IBM

Build a Business Process Solution Using Rational and WebSphere Tools

Explore IBM On Demand Business and business-driven development

Learn to use modeling, UML, and BPEL

Study implementation and integration

Peter Swithinbank
Hossam Badawi
Jenny He
Arisa Izuno
Parul Lewicke
Holger Schwarzer
Larry Yusuf

Redbooks

IBM International Technical Support Organization

Build a Business Process Solution Using Rational and WebSphere Tools

February 2006

First Edition (February 2006)

This edition applies to Version 5 of the WebSphere platform.

Contents

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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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 illustrates 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. 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 IBM's application programming interfaces.

Trademarks

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

AIX®	IMS™	SupportPac™
CICS®	MQSeries®	Trading Partner®
ClearQuest®	Netfinity®	TXSeries®
Cloudscape™	Rational Rose®	VisualAge®
DB2®	Rational Unified Process®	WebSphere®
developerWorks®	Rational®	XDE™
@server®	Redbooks (logo) ™	xSeries®
@server®	Redbooks™	z/OS®
Holosofx®	Requisite®	
IBM®	RUP®	

The following terms are trademarks of other companies:

Enterprise JavaBeans, EJB, Java, JavaBeans, JDBC, JSP, JVM, J2EE, Solaris, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

Intel, Pentium, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Preface

This IBM® Redbook is based on the experiences of a team in the IBM Hursley laboratory. They built an auto-claim insurance solution to put the WebSphere® software platform through its paces. The team worked with WebSphere developers to use the experience of building the solution to improve the design of WebSphere version 6 platform products.

They thought it would be valuable to share their experiences with a wider audience. The result is a tour de force, showing how the team went about using IBM's software development platform to understand business requirements and then architect, design and build the solution.

Their experiences will help you plan, design and build a business driven development solution using products from IBM's WebSphere Business Integration portfolio.

This redbook is written from the perspective of three types of developer: the business analyst, the software architect, and the IT specialist. Individual chapters in the book show how each member of the team developed their part of the solution, and how the team integrated the solution together.

This redbook helps you plan, design, and build a business-driven development solution using products from the IBM WebSphere Business Integration portfolio.

Unusually for an IBM Redbook team, we are not primarily focused on a product or technology. Instead, this book is focused on using a business-driven development methodology and applying it in a practical way to a specific scenario drawn from the insurance industry.

The solution on which this redbook is based is defined and built by the IBM System House development team in their Hursley laboratories in England, with help from IBM insurance clients. The subject of the solution is not important in itself. What is important, is that it is has been developed similar to a real solution. Normally IBM tests what individual products are built to do. But building a solution with our methodology forces our attention onto what products *haven't* been built to do. Building the solution reveals real cracks in the ways software needs to be used together to solve business problems.

The System House team was formed as part of an extensive effort within the Software Group at IBM to experience the real issues of using the many different products in IBM's software portfolio before they are released. The System House

team identified improvements for future releases of software (many of which are reflected in the changes to version 6 or the WebSphere discussed in Chapter 13, "Points to consider" on page 487), and worked with development and service teams to produce fixes and work-a-rounds for integration problems found in the current release of products - such as problems integrating long running processes using the WA0D SupportPac™ described in Section 12.2, "SupportPac WA0D overview" on page 471.

This IBM Redbook closely follows the work of the System House team and can be of value to you in a number of ways:

1. We write about a significantly large part of the development process, from the business requirements, through modeling the requirements, choosing an architecture for the solution and then designing and building the solution. If you are involved in planning a development project, then you should find this account useful, if not in providing a blueprint, at least in providing an example of one way to go about developing a solution.

2. If you have been thinking about the ideas contained in IBM's Software Development Platform introduced by Alan Brown, in papers such as "*Realizing the IBM Software Development Platform*", found at

   http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/SDP_WP_Final.pdf

   If you would like an example of the ideas put in to practice you will find this book interesting.

3. Many of the chapters deal with using one or other of the software tools to implement the solution. They show how we used the tools to build the solution. Nothing is left out.

   The materials we used are available on the IBM Redbooks™ Web site along with this book. If you are using one or other of these tools and are stuck, or think there might be another way of doing something, then find the relevant chapter for the tool in this book, and see what we did. It might help.

   There is a four-day workshop that is offered by ITSO based on this book, and that gives you an opportunity to go through many of the steps in building the solution.

The companies in this redbook

We emphasize that the company names used in this redbook are entirely imaginary, and do not refer to any real company, past, present, or future. There is no connection with the firms called DirectCar in Brazil, Spain or elsewhere, nor with Liberty Global (LGI).

The team that wrote this redbook

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

The team that wrote this redbook

Peter Swithinbank is a Project Leader at the International Technical Support Organization, Raleigh Center. He edits Redbooks and teaches IBM classes worldwide building business integration solutions. Before joining the ITSO last year, Peter worked in other capacities at IBM for 27 years. He has a diploma in software engineering from Oxford University and a Master of Arts in Geography from the University of Cambridge.

Arisa Izuno is a WebSphere technical sales in Japan. She has two years of experience in the field of business integration. Her areas of expertise include business process modeling. She has written extensively on WebSphere Studio Application Developer Integration Edition. She holds a degree in sociology.

Hossam Badawi is a software engineer in Egypt. He has four years of experience in software engineering and development. He holds a Bachelor of Science in Systems and Biomedical Engineering from Cairo University and is pursuing a Master of Science in signal processing. He worked for two years in the development labs in Cairo in which he contributed to the development of WebSphere Business Integration Workbench and WebSphere Business Integration Modeler. His areas of expertise also include modeling and deployment of WebSphere MQ Workflow processes. Hossam is also a Microsoft® certified .NET application developer for Web services.

Jenny He is a Solution Test Specialist in the United Kingdom. She has three years of experience in solution test area across various software and platforms. She holds a Ph.D degree in the field of Optical Networks from University of Essex in the United Kingdom. She has been actively involved in modeling and monitoring business processes since she joined IBM three years ago. Her areas of expertise include WebSphere MQ Workflow, WebSphere Business Integration Modeler, WebSphere Business Integration Workbench and WebSphere Business Integration Modeler. Jenny is a member of IEEE.

Parul Lewicke is a Software Engineer in the United States. She has three years of experience in z/OS® Integration Test, specifically testing WebSphere Business Integration Message Broker and WebSphere MQ. She holds a degree in Computer Science from Clarkson University in Potsdam, New York and is looking for new opportunities in Minneapolis and St Paul, Minnisota.

Holger Schwarzer is a Senior IT Architect in Switzerland. He has over sixteen years of experience in multiple facets of information technology and is a consultant for a large number of financial services projects in Switzerland, Germany,and the United States. His previous experience includes software development Tools of Smalltalk Server on z/OS, Unix and MS. He holds a degree in Master of Business Administration from School of Business: Fachhochschule in Muenster, Germany.

Larry Yusuf is a Solution Designer with Software Group Strategy and Technology based at the Hursley Labs in the United Kingdom. He has four years experience in Business Integration and modeling, with a particular focus on Business Process Management, Event and Solution Management, and Integration patterns. He has written and presented extensively on these topics.

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

Jim Amsden, Model Driven Development, Raleigh, made early versions of the Modeler integration with Rational® Software Architect available to us, made such an excellent job of the integration and contributing Appendix B, "Integration considerations" on page 505.

Andre Weiser, WebSphere MQ Workflow Development in Boeblingen, championed the need for the WA0D Supportpac to integrate WebSphere MQ Workflow and WebSphere Business Integration Server Foundation simply, and then worked long hours to get it working properly with long running processes.

Many people in the Hursley System House and Application Integration Middleware Service team, who built the initial solution, helped to solve technical problems getting the solution in the redbook to work, and helped with the development methodology. **Pete Edwards**, **Mick Lickman** (WebSphere Business Integration Message Broker), **Keiron Scott**, **Nick Maynard**

(WebSphere Studio Application Development Integration Edition and WebSphere Business Integration Server Foundation), **Alan Chivers, Mike Starkey** (IT Architects), **Milena Litoiu** (Rational products), **Andy Gibbs**, **Paul Versheuren** and **Jean Pierre Paillet** (Process Integration Design Approach) and **Sue Horn**, **Rosanne de Vries**, **Mohammed Abdula** and **Louise Cheung** who sponsored and managed the System House teams involved in the solution.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

► Use the online **Contact us** review redbook form found at:

> `ibm.com`/redbooks

► Send your comments in an email to:

> `redbook@us.ibm.com`

► Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. HZ8 MP 206
P.O. Box 12195
Research Triangle Park, NC 27709-2195

Part 1

Background

In the first part of this redbook, we discuss the background behind building the new business solution to automate the claims assessment process in a large auto insurance company.

Chapter 1, "Business context" on page 3 introduces the insurance companies, LGI and DirectCar, which have recently merged. The chapter explains the goals of the merger, and why having merged the policy and claim systems, LGI are now considering automating the process of sending claims *adjusters* in the U.S., or claim *assessors* in the U.K. and elsewhere, to inspect vehicle accidents.

Chapter 2, "Current architecture" on page 13 describes the merged policy and claim systems, the solution architecture, and the products that were used to implement the solution.

Together, the business and system context are the background for Part 2, "Modeling" on page 37, where we explain more detail behind the existing claim system and then develop the new external claim assessor automation solution.

1

Business context

In this chapter we introduce the business scenario on which this Redbook is based. We tell the history of the company involved in this scenario and introduce the roles of the people involved. We discuss the business goals that shape the scenario, as well as the IT goals that drive the implementation.

The integration issues described in this Redbook are not unique to the insurance industry or to company mergers. For any company, continual business change and organizational consolidation can result in a myriad of pieces of hardware, software components, and applications that need to be made to work together to implement a new solution rapidly. The insurance industry integration scenario we present is an example that can be applied to other industries.

1.1 Setting the scene

Continual business change and organizational consolidation through a merger, an acquisition, or other event often results in a myriad collection of hardware, software, and applications that rapidly must work together as one solution. From the IT perspective, the response to business integration has focused traditionally on consolidating data and applications. Although this is a valid approach, it is a long-term, high-cost activity that must be balanced with the need for a more immediate return on investment.

A faster approach that provides rapid return involves exploiting a combination of process management and enterprise application integration. Rather than rewriting existing applications to combine their functionality, we use tools to understand the processes that current applications execute and link them together with application integration software. The goal is to provide a single, integrated view of a process to customers and business partners across multiple companies with minimal disturbance to the existing applications. The term *Business Driven Development* is sometimes given to this approach. Business Driven Development is applied specifically to IT projects with a major incentive to achieve business goals measured by a revenue or profit growth in a specific line of business.

In this Redbook, we present you with a scenario focusing on an imaginary insurance company, ITSO Lord General Insurance, hereafter referred to as LGI. LGI is a 50-year-old, well-established business with existing, mainframe-based IT systems.

1.1.1 Company history

LGI recently acquired a new company, DirectCar, with an Internet-based infrastructure. Both LGI and DirectCar had their own processes and very different back-end systems. To appreciate some of the challenges that LGI faced when integrating DirectCar into its IT infrastructure, we first need to understand the two companies.

LGI

LGI has been in business for 50 years and focuses on auto, home, and home contents insurance and has over five million policyholders. Prior to the acquisition of DirectCar, contact with customers was through the traditional channels of independent agents and a recently outsourced call center. Thus, the only way that customers could buy, update, or make claims against LGI policies was through an LGI agent or by calling the call center. Prior to the merger with DirectCar, no internet-based channels existed. LGI systems are

mainframe-based. LGI has not invested in mainframe-based internet channels, and has no plans to develop new mainframe-based applications.

DirectCar

DirectCar had less than one million policyholders, but was an expanding new company focused on auto insurance through the Internet. It interacted with customers solely through a direct customer Internet channel. DirectCar used desktops and server-based systems, rather than mainframe-based, and supported J2EE™, open, architecture. It had an e-business focused infrastructure based on WebSphere Application Servers, Oracle databases, and TXSeries®.

Motivation for the merger

As a traditional personal lines insurance company, LGI realized it needed to expand its market share by establishing a direct customer channel to complement the existing agent and call center channels. To this end, it acquired an insurance newcomer, DirectCar, to provide a quick entry into the direct insurance market and to utilize this company's Internet-based IT skills and infrastructure. The merged company continues to be known as LGI.

1.1.2 Scope of the scenario

This merger required LGI to re-engineer significantly its processes and integrate many of its existing applications with DirectCar to provide a unified view of the business to customers, business partners, and employees, all while minimally disrupting the vastly different existing systems.

One business system that was merged was the claim system. The merged claim system is not a brand new implementation, but an evolution of the existing LGI and DirectCar claim systems. In this Redbook, we assume that this merged claim system has already been developed and is fully operational. We do not discuss the integration of the LGI and DirectCar claim systems.

Even though it has successfully merged the two claim systems, LGI wants to further reduce cost of assessing claims by automating the process of getting loss adjustments from external assessors. We focus on using WebSphere Business Integration to automate this portion of the claim system. The scope of this scenario is limited to automating getting assessment reports from claims adjusters, or claim assessors as they are known outside the United States. The goals of providing both a business and an IT view of the process to monitor the performance of the claims process is something we hope to include in a later redbook based on this same scenario.

1.1.3 Claim system

We use the term *claim system* to refer to the set of processes and IT systems that are used to process a claim filed by an auto insurance customer or agent of the merged LGI company.

Claims process overview

When an LGI customer has a car accident, he or she reports the accident and details surrounding the accident to LGI, the insurance provider. This is what we consider filing claim. The LGI Claim handler then reviews the information and can request more information such as a police report, medical report, or a report from an external assessor. External assessors are not LGI employees and must be contacted each time their services are needed. They make a visual inspection of the automobile in question, assess the damage, and report back to LGI. LGI can then make a decision on how to settle the claim with the customer. More on the claims process can be found in Chapter 3, "Our method" on page 39.

Although this might seem like a straightforward procedure, there are many activities and parties involved. Executing the external assessor's portion of the claims process manually is expensive and time consuming.

External assessor and the claims process

Though LGI has merged its claims processes with DirectCar, resulting in a single claims process, LGI still feels its view of the claims process is fragmented. There are still parts of the flow which are outside of LGI control and cannot be monitored from one single point. In analyzing the claim system, LGI has identified a particular problem in the external assessors portion of the claims process where costly delays have been identified in the selection and follow-up of external claim assessors. LGI would like to automate this selection process and get a more comprehensive view of the progress and efficiency of the assessors.

Currently, after a claim handler reviews a a particular claim and determines that an external assessor report is needed, he or she goes through a manual process to determine which of the external assessors that LGI works with are in the same geographic area as the vehicle to be inspected and have expertise in assessing that type of car. This is time consuming and expensive, making it an ideal task for automation.

The Automated Assessor Management system that LGI would like to develop will be responsible for identifying and selecting an assessor based on geographic

location, availability and knowledge of the type vehicle. The system must have the following capabilities:

► Manage the communications between the LGI and the external claim assessor through the initial selection process and the stages of the assessment.

► Hold details of the available external assessors and their contact information.

► Link to the main claim system and allow authorized LGI claims personnel to monitor the progress of an claim assessment

► Hold details of a claim assessment through its life cycle.

► Allow event points to be established. These event points can feed event information, including status, to the main claims process for display on a business monitor or dashboard.

► Provide a rules engine that can be updated with business rules to determine which external claim assessor to select for a particular claim.

Using business integration to automate the external assessor process will provide the following benefits:

► LGI claims personnel can concentrate on other tasks.

► LGI with have the ability to monitor and analyze the external assessment process in more detail,

► LGI will be able to make process improvements based information collected by the claims monitoring system.

1.2 Business goals

Design and development of the Automated Assessor Management system is driven by a number of business goals and constraints. These goals can be divided into four main categories:

► Reduce total business cost of obtaining external assessment reports.
► Increase customer satisfaction by reducing administrative delays on claims.
► Provide a complete view of the external assessor business process to LGI.
► Incorporate existing systems and resources into the Automated Assessor Management system.

Because the project is regarded as technically innovative and therefore high-risk, the choice of the Automated Assessor Management System must not jeopardize the business by too rapid deployment. The solution must be brought in incrementally, gradually replacing the existing manual process.

We discuss each of these business goals in more detail in the following sections.

1.2.1 Reduce cost

A major and obvious goal of any business change is to reduce costs. LGI wants to reduce the administrative costs of obtaining assessment reports from external assessors. Currently, claim handlers use a manual process to match claims with external assessors who are in the same geographic area and who specialize in the type of vehicle in question. Automating the process will allow claim handlers to focus on other tasks, thus driving down the overall cost of getting external assessment reports.

1.2.2 Increase customer satisfaction

Another important goal of the Automated Assessor Management system is to increase customer satisfaction by reducing the total time for claims to be processed. Delays often occur because the assessor for a particular claim either cannot be reached, or the claim handler is backed up trying to find assessors for other claims.

LGI also expects the Automated Assessor Management system to increase satisfaction by reducing administrative delays on claims queries. Currently, when a customer contacts LGI to check on the status of his or her claim, it can take days before the exact status of the claim can be determined, due to the many steps involved in selecting an external assessor. Automating the system will allow Claim handlers to easily pinpoint exactly where in the process a claim is, thus allowing them to keep the customer better informed.

1.2.3 Incorporate existing resources into the new solution

It is important to LGI to incorporate current resources into the new automated solution. These resources include not only hardware and software, but also the people who currently perform the manual, external assessor process. The goal is not to replace the Claim handlers with an automated external assessor process, but rather to incorporate the solution into the Claim handler's existing job. The workload on the Claim handler should be reduced, allowing them to focus on other claims and customer tasks.

Additionally, LGI does not want to scrap their existing systems that deal with external assessors, but rather build an automated solution that takes advantage of these systems. For example, LGI currently has several systems and applications that hold basic information about the external assessors, past assessment reports, and business rules used to select an assessor for a particular claim. LGI wants to reuse these systems in the Automated Assessor Management system, resulting in minimal impact to the existing systems and maximizing ease of development of the new system. The new Automated

Assessor Management system also needs to be responsive to business needs so that future enhancements to it will require minimal rework and redesign.

1.2.4 Provide a complete view of the external assessment process

LGI currently feels that it has a fragmented view of the external assessment process. There are still parts of the flow that are outside of LGI control and cannot be monitored from one single point. The Automated Assessor Management system must give LGI a complete view of the external assessment process, including activities performed by external parties such as the assessors.

For example in the current system, if a customer inquires about a claim that is awaiting assessment by an external assessor, all that the Claim handler can tell the customer is that the claim is stalled at this point in the overall process. It is difficult for a Claim handler to give more detail, such as whether or not an assessor has begun the assessment report, been assigned to the claim, or even been chosen for this claim. A goal of the new solution is to remove this inability to give customers more details on the status of their claims.

Another reason that LGI wants a complete view of the external assessment process is so that in the future it can monitor the process more closely to measure the performance of assessors and identify any delays or weak spots in the process. Information from the external claims assessment process about past performance can be used by the Business Rules Engine to choose the best assessor to deal with a future insurance claim.

1.3 IT goals and constraints

In addition to business goals, IT considerations play a large part in the design and development of the Automated Assessor Management system. These IT goals can be divided into several categories:

► Minimize IT costs. Keeping IT costs low is an ongoing goal of LGI. With the Automated Assessor Management system, LGI hopes to minimize IT costs by reusing many of the existing systems that deal with external assessors.

► Deliver in the short-term. LGI wants rapid results, so they need the system implemented in one year.

► Minimize the impact of the Automated Assessor Management system on existing applications and processes. The new solution has to work with the existing claims process without requiring any changes to it.

► Reuse existing applications and back-end systems. The new solution must reuse the existing systems that keep track of assessor information, business rules, and prior assessment reports. Additionally, it must support external

assessors who have different interfaces and back-end systems (Web interfaces, WebSphere Application Server, CICS®, and so forth).

► Demonstrate development productivity improvements by using modern tools and technology. The new system will implement a Service-Oriented Architecture (SOA) based on Web services. It will be developed using Model Driven Development (MDD) and use open standards based technologies such as J2EE, BPEL, and UML2. The development project will serve as a reference for using these technologies in the future, showing how to achieve development productivity improvements.

► Use Automated Assessor Management system as a proof-of-concept for automation of other processes in the future. The Automated Assessor Management system must be built in such a way that it can be used as a basis for future efforts to enhance more business critical components of LGI's infrastructure, such as the policy management system or claims process as a whole. There are two main questions to answer:

   – What are the problems in integrating existing processes and applications with applications integrated or created in J2EE and Web services technologies?

      • Can existing FDL processes interoperate with BPEL processes?
      • What problems are there migrating from a message-oriented middleware backbone to a service-oriented architecture?

   – What are the best practices to adopt for an MDD approach to development and what benefits in terms of speed and cost of development does it deliver?

1.4 Roles

In the scenario for this Redbook, there are several roles that have to be filled. At LGI, there are the claim handler and external assessor. Additionally, to design and implement the solution, we need to understand the roles that the business analyst, solution architect, and application developer (or IT specialist) play.

A brief description of the more important roles in the solution follows in this section. Figure 1-1 on page 11 puts these roles into their business context. They are described in more detail in Chapter 3.2.1, "Roles and responsibilities" on page 46.

Business analyst

The business analyst is the specialist in the company's particular business domain (insurance, in the case of LGI) and knows the business inside out. The analyst fully understands the specific functions of the business for which they are

responsible and is expected to plan and deliver changes to meet business goals. Furthermore, the business analyst is proficient in the modeling tools used to document and analyze business processes.

Solution architect

The solution architect understands the business goals of the Automated Assessor Management system to be developed and has specialized understanding of particular aspects of the business, especially the claims process. The solution architect fully understands the current IT architecture and is responsible for mapping new business goals to technical solutions compatible with current IT structure. The solution architect also understands industry patterns and approaches related to the solution such as SOA and Web services.
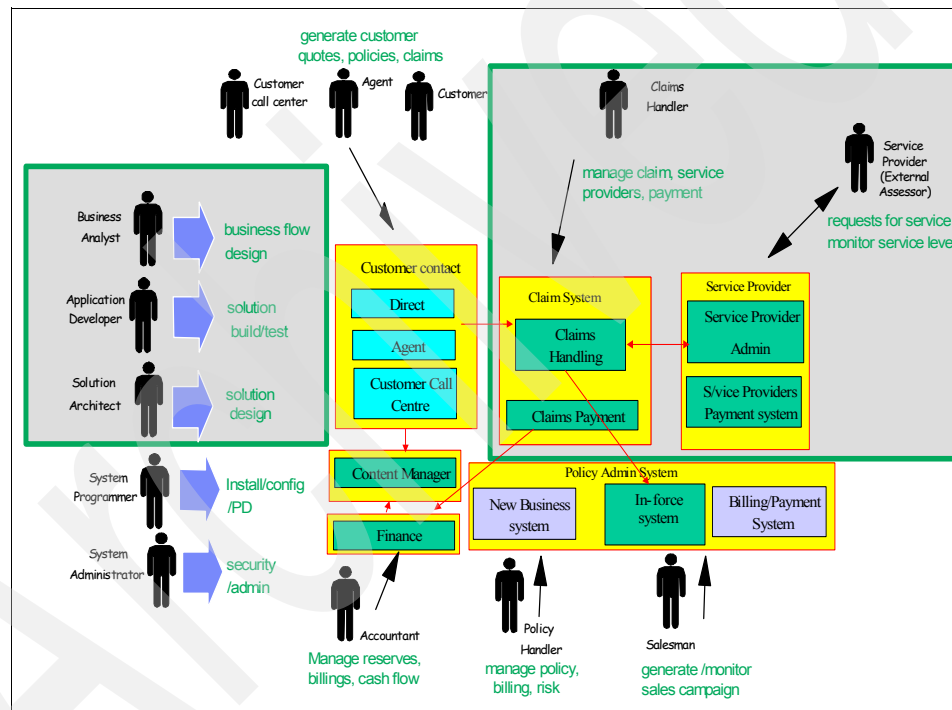
Figure 1-1   Business and development roles at LGI

Application developer or IT specialist

The application developer has a thorough understanding of the business components involved in the proposed solution and is able to program them as required, for example, using J2EE for new applications or working with existing data in existing formats.

LGI Claim handler

The LGI claim handler manages claim payment, policy administration, and generation of external reports as necessary to complete the claim. The claim handler currently talks, faxes or e-mails the external assessors to assign assessment tasks. The claim handler is also responsible for the overall completion of the claim.

External service provider (external assessor)

External assessors are not employees of LGI, but rather work for assessment firms external to LGI. They provide the important service of being impartial loss adjusters for vehicles being considered in LGI claims. After committing to assessing a particular claim, the external assessor is responsible for visiting the vehicle in question and determining the amount of damage. The external assessor then prepares a report of his or her findings and sends this back to LGI.

1.5 Summary

This chapter addresses integration issues that arise when a recently merged company wants to improve its IT infrastructure further. It focuses on describing the two companies in this scenario, their merger, their claims processes, and their decision to automate the external assessor management portion of the merged claims process. The following chapters lay out the Automated Assessor Management system in more detail, including business processes, solution architecture, and implementation.

The integration issues described in this redbook are not unique to the insurance industry or to company mergers. For any business, continual business change and organizational consolidation can result in a myriad of hardware, software, and applications that rapidly need to work together as one solution. The integration scenario that we present is meant to be an example that can be extended to other situations.

2

Current architecture

This chapter describes the architecture of the environment that supports the existing claim system. We also describe the interaction between users and components, and between the components and any other remaining components.

The system has been built by the System House eMerge Solution Test team in the development laboratories in Hursley[1]. The laboratories at Hursley are used to prototype new capabilities and to test the integration of new releases of the WebSphere platform. The solution test team run a customer partnership program that provides short residencies for clients to work alongside the developers in Hursley to prototype solution extensions. For details, contact your account representative.

The following topics are discussed in this chapter:

► Before the merger
► The merged solution context
► Integration solutions
► IT infrastructure

---

[1] Now called the Horizontal Integration team.

13

2.1 Before the merger

In this section, we look at LGI and DirectCar IT infrastructures before the merger, which are shown in Figure 2-1 on page 14.

The claim system for DirectCar is based around a three-tiered, net-centric architecture that lets clients register claims online and receive updates on the status of their claims through e-mail or traditional mail. The IT infrastructure that supports the claims processing consists of a cluster of application servers that handle both the transformation and collation of data provided by the client, and sends this data in the form of requests to an off-the-shelf claims application in the back-end system. Replies from the back-end applications are sent to the application servers, where they are presented dynamically to the client as Web pages. Other processing is performed manually or in the back-end system.

LGI has a message-based, hub-and-spoke infrastructure with all client applications sending requests to a central broker that handles the transformation and routing to the back-end applications or workflow systems. All replies from the applications are sent to the message hub for transformation and routing to the client application. A customer registers a claim by contacting a call center or their insurance agent where the claim agent collects the required information and uses EDI or a dedicated client to input the information to LGI. As with DirectCar, all the claims processing in LGI is done manually, or through dedicated client applications accessible by the claim handler and claims supervisor. LGI provides channels for business partners into the message broker.
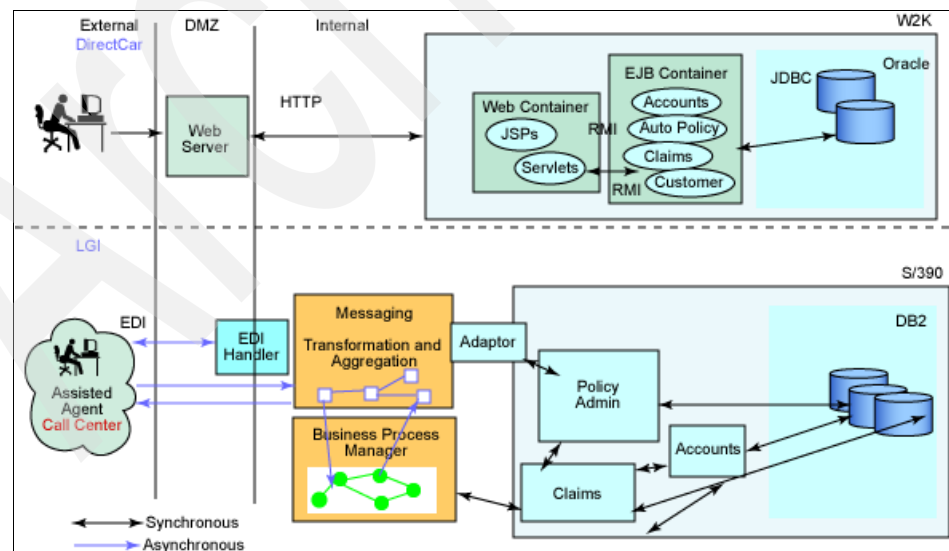
Figure 2-1   LGI and DirectCar IT infrastructure before merge

The business vision for the merger, as shown in Figure 2-2, was to create a one-company-for-all-channels view that hides the customer and staff involved in the claims process from the complexity of the LGI and DirectCar back-end applications.
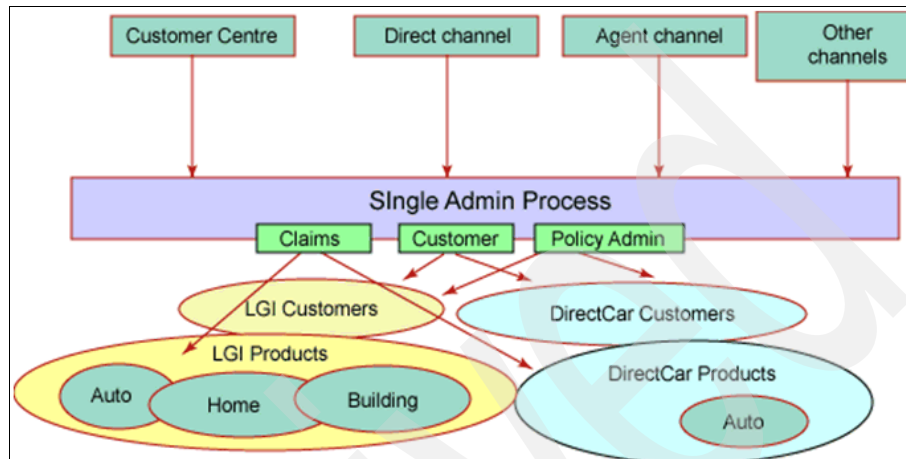
Figure 2-2   Business vision of merged companies process

2.2 The merged solution context

The solution context in Figure 2-3 on page 16 related to the merger scenario involves LGI and DirectCar.com billing, claims and policy processes and systems. The parts we are concerned with are outlined in red in the top right of the picture: the claims handling in LGI and interfacing to the external claim assessors. To put this in context, we give a brief description of the other components of the solution.
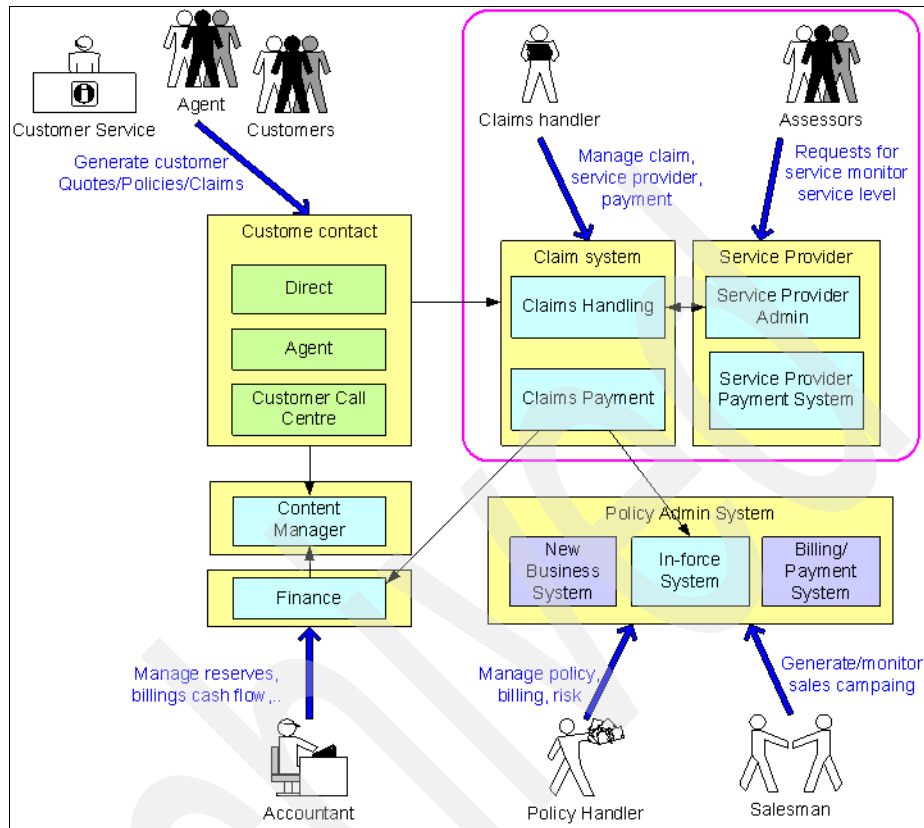
Figure 2-3   Mergers & acquisitions scenario solution context

Customer contact

The responsibility of this subsystem is to enable customers to obtain policy quotations, sign up for policies, check their policies, and submit claims for processing. LGI claims come from several sources. A customer may directly enter a claim using the Internet. A Web application is required to provide this functionality. Alternatively, claims can be submitted indirectly by agents or through EDI (Electronic Data Interchange), or a call center (EDI/ XML) where the call centre agent uses a specialized client-server application.

Quote and policy administration

The policy administration system is the central repository for policy and customer information. It provides facilities to record and update policy and customer details.

Claims handling

The claims handling system is a separate subsystem that is responsible for supporting all aspects of a claim. There are extensive interactions between the business process manager and the claims handling system. It is assumed that updates to claims are always initiated by the business process manager.

Both the policy administration and claims handling systems record customer related information. The policy administration system records information about policy holders and their policies. The claims handling system records information about other third parties such as other insurance companies, other claimants, licensing authorities and so forth. The policy administration and claims handling systems share data about policy holders.

Claims payment

This subsystem is responsible for initiating payment to a customer, repairer, or third party. It could be part of the claims handling system or the financial system. In auto insurance, repair payments are usually made directly to the repairer, and only payments for personal items lost or damaged are paid directly to the customer.

Financials

The financials subsystem is responsible for maintaining all financial information and managing payments and receipts. The business process manager will communicate directly with the financial system information about potential payments (reserves). Information about payments will be communicated indirectly using the claims handling and claims payment systems.

Service providers

Assessors, repairers, and other third parties are all various types of service providers.

- ► Assessors are specialist companies that provide estimates and recommendations concerning vehicle repairs.
- ► Repairers perform the actual repairs on a vehicle.
- ► Service providers could be organizations that provide vehicle recovery or car hire services.
- ► Other third parties can include organizations such as the police, and vehicle licensing authorities.

In general, the claims handling system supports the exchange of information with such parties with a variety of means.

2.3 Integration solutions

The integration solutions developed for the merged LGI are briefly described in this section. They are based on the merger and the development of common claims and policy administration processes that support the integration across the two companies. The solutions fall into two broad areas: quote and policy administration and claims. We briefly review the quote and policy solution. We then discuss the existing claim system and the proposed new external claim assessor integration solution.

2.3.1 Quote and policy administration

The goal is to integrate disparate applications into a single insurance quote and policy accept system which selects the best quotation from different insurance applications for a customer to accept, and presents a single view of policies to both customers and the rest of the insurance system. The componentization of the solution enabled the creation of a single company Web application to work alongside LGI's existing EDI and dedicated call centre applications.

After the customer's information is validated, the merged quote and policy accept solution accesses the policy quote systems of both LGI and DirectCar using a message broker. Based on rules, the broker selects and returns the best insurance quote.

When a customer accepts the insurance quote, the next step is to verify the customer's previous history. This involves requesting and receiving the prospective customer's motor vehicle information (MVR) and credit check rating from external agencies. This part of the solution involves building a subprocess to handle the multiple requests for external information and integrating these steps with the existing LGI or DirectCar quote and policy systems. The process handles the timing issue of delayed responses from the external agencies, while continuing the process to the point of acceptance.

2.3.2 Claims Integration

The claims process is composed of four main steps or processes that are executed sequentially.
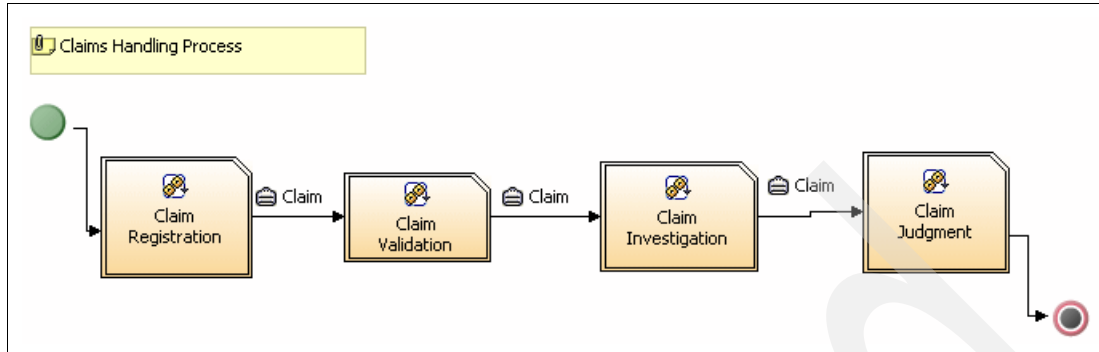
Figure 2-4   Basic claim handling process

> ► Register claim
>
>   The client provides details about their policy and the claim.
>
> ► Validate claim
>
>   The insurance company validates the claim . Is the policy paid up, and is the claim appropriate on the policy?
>
> ► Investigate/Assess claim
>
>   The insurance company investigates the claim, gathering information from external parties such as police and medical reports, as well as assessing the claim.
>
> ► Judge claim
>
>   Based on the results of the external reports and the assessment, the claim handler makes a decision whether the claim is valid or the claim is rejected. The claims supervisor might be involved if there are unusual aspects to the claim.

A process engine, WebSphere MQ Workflow, is used to integrate the LGI and DirectCar claim systems into the overall claims handling process. It gives visibility to the overall progress of a claim, whether it is in DirectCar or LGI. Integrating LGI and DirectCar claim systems with a common process engine also enables the development of new common functions in the claims process such as the gathering of claims assessments from external claim assessors.

We briefly describe the claims processes of the two companies here. But, from the perspective of the External Claim Assessors, the important point is that the existing claims process integration has made possible the simple extension of a common process to provide a new level of automation in the handling of external assessors.

LGI claims process

LGI already has an infrastructure for the claim system with all client applications sending requests to a central message hub that handles the transformation and routing to the required back-end applications or workflow systems. All replies from the applications are sent to the message hub for transformation and routing for the required client application.

The LGI claims processes are:

1. Register the claim.

   The client contacts the call center or an independent agent. A claim handler records accident details, manually completes the required forms, and enters the required information in the claims database. The claim handler then gives the client a claim reference number. This step can be considered as a manual step.

2. Validate the claim.

   The raised claim is authenticated to confirm that the client's policy is valid and not expired, that the details provided are accurate, and that the driver is insured on the auto policy.

3. Investigate the claim.

   In the investigate claims process, the claim handler requests external reports from a number of external agencies or companies as shown in Figure 2-5.
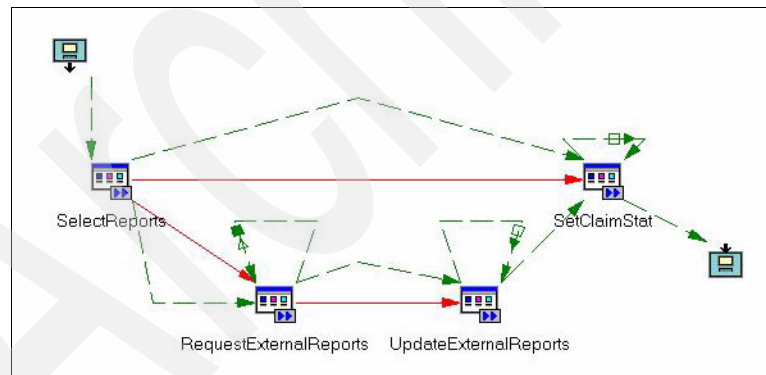
Figure 2-5   Investigate Claim sub-process

One of these is the external assessors claims assessment report. The LGI claim handler manually selects an assessor from the assessors database and requests an assessment report as shown in Figure 2-6 on page 21.
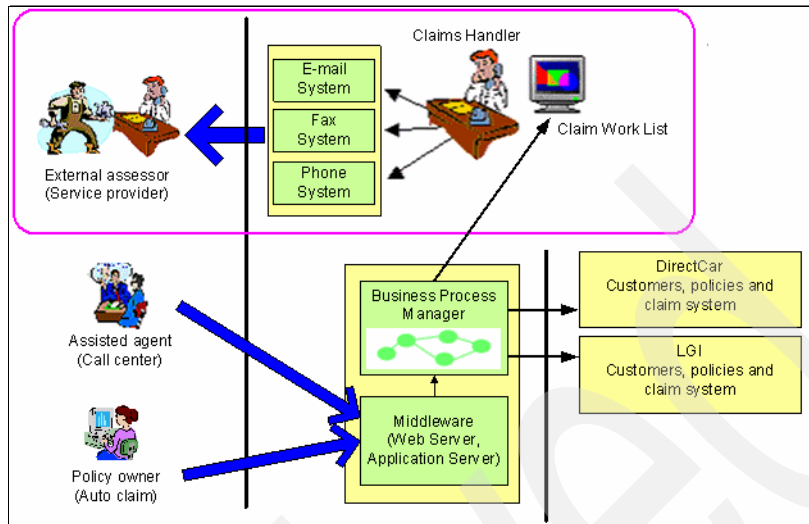
Figure 2-6   Manually requesting assessments reports from external assessors

4. Judge the claim.

   Based on the results of the assessment, the claim handler decides whether the claim is valid or the claim is rejected. The claim handler might decide to involve the supervisor in the decision, if there are any unusual aspects to the claim.

Figure 2-7 and Figure 2-8 on page 22 show the use cases in the LGI existing claim system.

Figure 2-7   LGI claim registration process

Figure 2-8   LGI Claim handler and claim supervisor activities

DirectCar claims process

The claim system for DirectCar is based around a three-tiered, net-centric architecture that lets clients register claims online with their Web browsers and receive updates on the status of their claims through e-mail or traditional mail. The IT infrastructure that supports the claims processing consists of a cluster of application servers that handle both the transformation and collation of data provided by 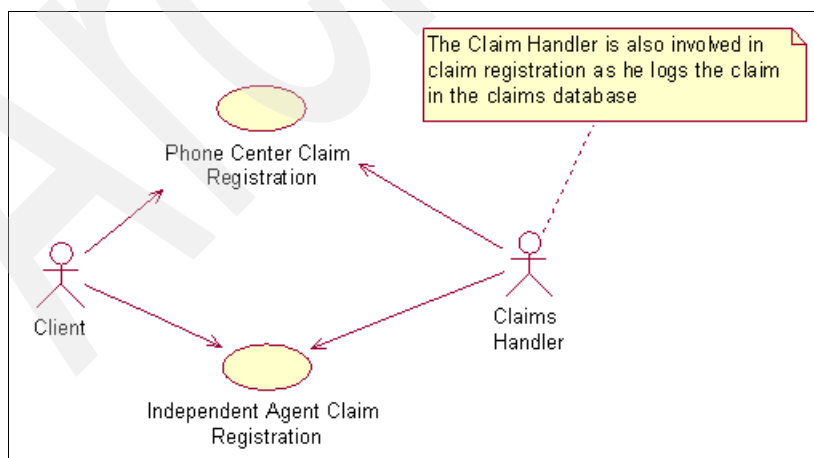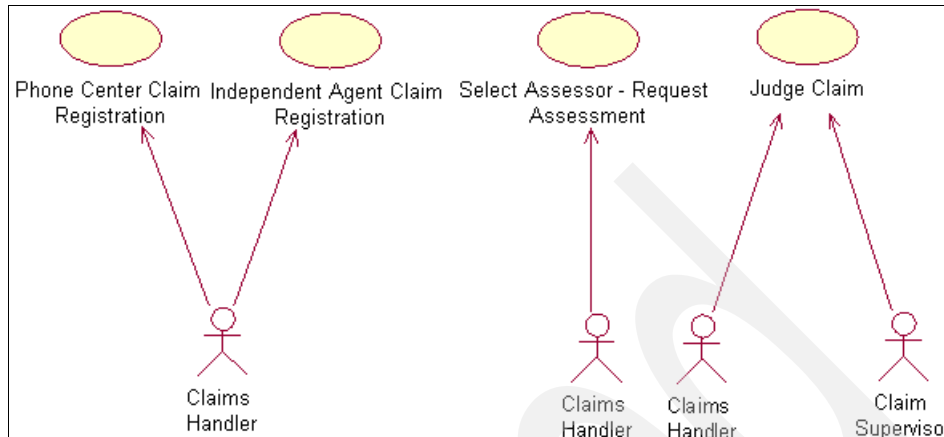the client, and sends this data in the form of requests to an off-the-shelf claims application in the back-end system. Replies from the back-end applications are sent to the application servers, where they are presented dynamically to the client as Web pages. Other processing is performed manually or in the back-end system.

The DirectCar claims processes are:

1. Register the claim.

   The client logs on to the DirectCar Web site and registers a claim online. The client is recognized as a policy holder, then a claim reference number is presented to the client. No actions are required from the claim handler at this point. This is an automated process.

2. Validate the claim.
3. Investigate the claim.
4. Judge the claim.

   These steps are the same for DirectCar as for LGI.

Figure 2-9 and Figure 2-10 on page 23 show the use cases of DirectCar existing system.
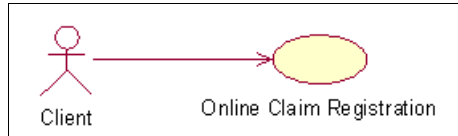
Figure 2-9   DirectCar claim registration process
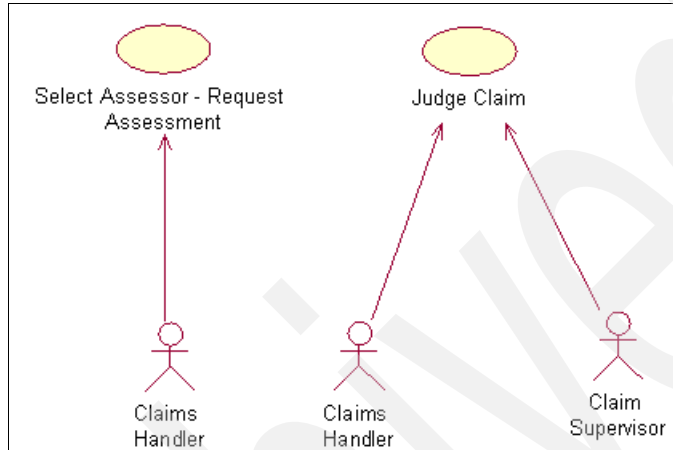
Figure 2-10   DirectCar Claim handler and claim supervisor activities

In the existing LGI and DirectCar systems, certain applications are used in various steps of the process. These applications are different and used only for storing data and tracking claims status. Much of the interaction with the client and between departments is performed manually, on paper using the mail or fax.

2.4 IT infrastructure

In this section, we describe the high-level architecture of the environment that supports the merged claim system. We also describe the interactions between users and components.

The solution is built assuming the reuse of the existing infrastructure and claims applications from LGI and DirectCar while rebuilding the DirectCar Web application for LGI. This choice is driven by the following reasons:

► Business and strategic reasons:

  – LGI strategy in acquiring DirectCar was to acquire market share and the Web channel, rather than effect cost reduction by rationalization of the LGI and DirectCar back offices.

- The IT architecture must support the possibility of a future demerger.
- The merger of the two companies must be associated with a change of the look and feel of the combined Web site. However, the same insurance policies will continue to be offered, though now to both LGI and DirectCar customers.

► Technical reasons:
- Current DirectCar IT hardware infrastructure is not sized for the new expected workload: DirectCar has less than one million policies while LGI has more than five million.

  i. Claims from LGI policy holders must be serviced by the existing LGI backend.

  ii. The existing DirectCar Web application will need modification to handle the new LGI policies as well as to handle the increased work volume.

Architecture overview diagram

Figure 2-11 shows the proposed architecture overview diagram for the merged claim system.
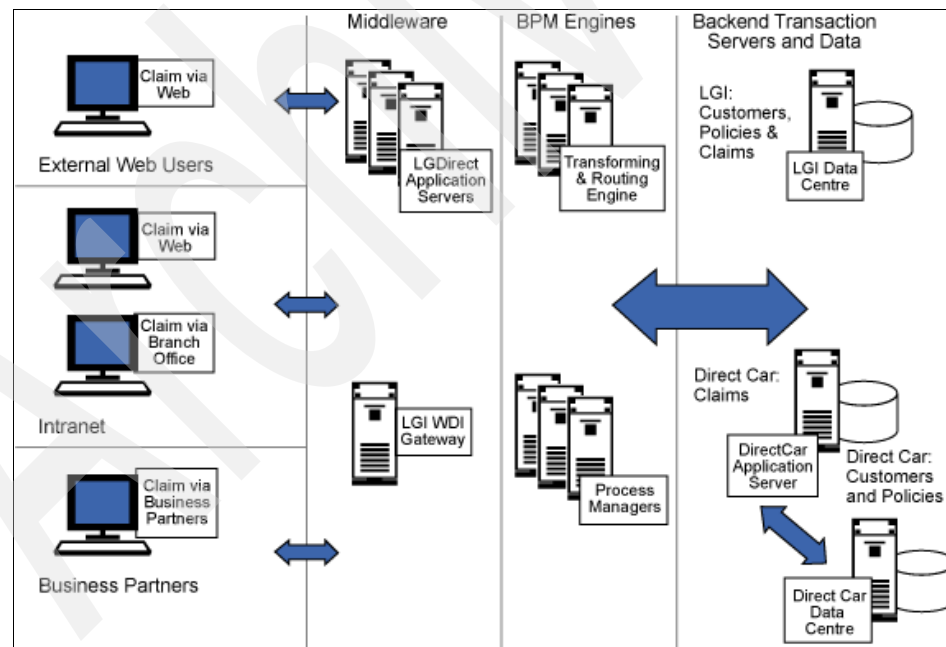

Figure 2-11   Merged claim process architecture overview diagram

As shown in the figure the solution architecture is based on the following decisions:

► Separation of the presentation layer from the backend applications by routing and process management engines

  – The presentation layer does not access backend services directly. Interactions with backend services are mediated by the transformation and routing engine and orchestrated by the process engine. The mediation engine connects a presentation service to the process engine, and the process engine to a backend service.

► Complete reuse of both back-end components such as business logic, data access and connectors

► Removal of the DirectCar Web layer (presentation and controller).

Figure 2-12 explains the products and technologies used to implement each of the claim system components.
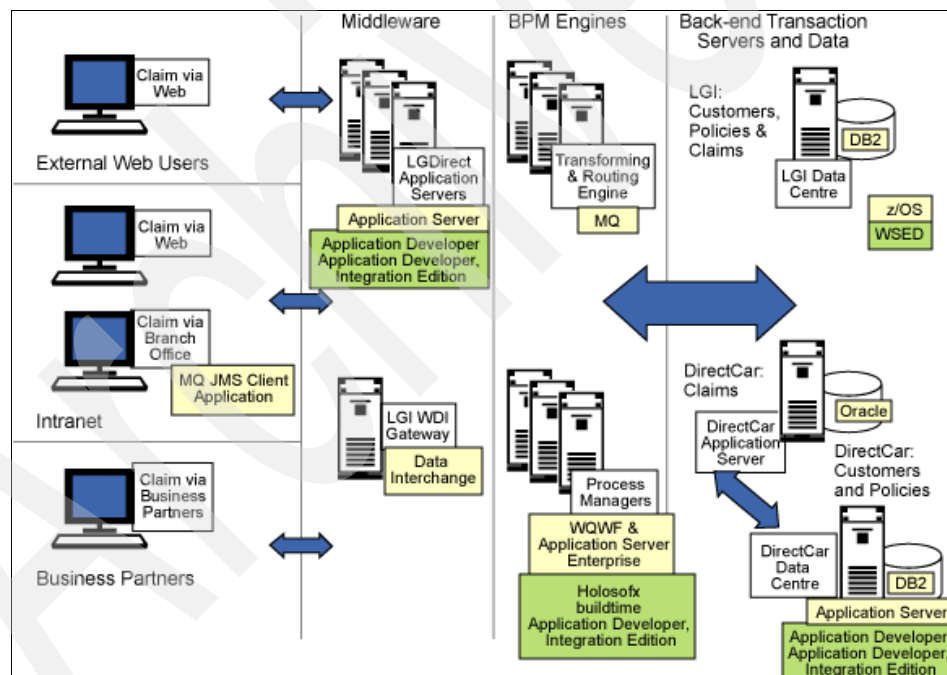

Figure 2-12   Products used to implement the claim system architecture

We decided to retain the existing IBM WebSphere MQ infrastructure of the LGI Company and extend it to incorporate the new common front-end Web application and existing DirectCar back-end systems. The WebSphere MQ infrastructure consists of multiple WebSphere MQ clusters configured to transfer

data among the different application components running in different products on various operating systems. This takes advantage of WebSphere MQ's assured delivery mechanisms and its wide platform coverage.

Each component displayed in Figure 2-12 on page 25 is described in more detail below.

2.4.1 User interfaces

User interfaces include the Internet, intranet, branch offices, and business partners. The combined claim system provides a user interface to register an auto insurance claim in the form of:

- ► An intranet Web interface for DirectCar customers.
- ► A Java™ Swing application for the LGI branch offices and call centers

  The application uses WebSphere MQ clients to transmit the information to and from the transformation and routing engine.

- ► An EDIFACT interface for business partners. The application sends EDI 835 data to the Data Interchange gateway.

The decision that leads us to choosing the products is that existing interfaces used by DirectCar customers, LGI call center, business partners, and branch offices must be retained unchanged.

Runtime
The products used for runtime of the user interfaces part are listed in Table 2-1.

Table 2-1 Products used in user interfaces runtime

Product	Platform	Version
Java runtime engine (JRE)	Microsoft Windows® 2000	1.3, 1.4.x
Web browser to view Web pages (Java Server Pages (JSP™) pages and HTML)	Windows 2000	N/A
WebSphere Data Interchange (Data Interchange)	Windows 2000	3.2.1
WebSphere MQ Client	Linux®, Windows 2000	5.3.x

Tools

The products for tools to create user interfaces part are listed in Table 2-2.

Table 2-2   Products used as the tools for developing user interfaces

Product	Platform	Version
WebSphere Studio Application Developer (Application Developer)	Linux	5.0.x, 5.1.x
Usage: For application development of Java Swing user interface		

2.4.2 Application Servers

Our middleware includes the LGI and Direct Application Server and Data Interchange Gateway. The middleware performs the following functions:

► The middleware provides a newly consolidated Internet Web site for DirectCar and LGI policy holders to register an auto insurance claim, based on the existing DirectCar applications. Uses the HTTP server to host static HTML pages with the Application Servers hosting JSP pages, servlets, and EJB™ components. The EJB components have been modified to issue JMS API calls to pass requests to and from the transformation and routing engine.

► Provides an intranet Web site for the company's claim handlers using HTTP server and Application Servers to interact with the process manager.

► Network Deployment Edge Server components provide load balancing across multiple Application Server nodes for a scalable solution with failover capabilities.

► The Data Interchange gateway puts the EDI 835 data it receives from the business partner onto a WebSphere MQ queue, where it is consumed by the transformation and routing engine.

The decisions that lead us to choosing the products for middleware are:

► Application Server allows the merged company to expand into J2EE technologies, using the skills and technology possessed by the DirectCar Company.

► In addition to supporting WebSphere MQ as a JMS provider, Application Server meets the company's quality of service requirements.

► Data Interchange is required for the continued support of the business partners' channels.

Runtime

The products used for runtime of the middleware part are listed in Table 2-3.

Table 2-3   Products used in middleware runtime

Product	Platform	Version
IBM HTTP Server (IHS)	AIX®, Solaris™, Windows 2000	1.3.x, 2.0
WebSphere Application Server (Application Server)	AIX, Solaris, Windows 2000	5.0.x, 5.1.x
WebSphere Application Server Network Deployment (Network Deployment)	AIX	5.0.x, 5.1.x
WebSphere Data Interchange (Data Interchange) Gateway	Windows 2000	3.2.1
WebSphere MQ	AIX, Windows 2000	5.3

Tools

The products for tools to create middleware part are listed in Table 2-4.

Table 2-4   Products used as the tools for developing middelware

Product	Platform	Version
WebSphere Studio Application Developer (Application Developer)	Windows 2000	5.0.x, 5.1.x
Usage: For application development of servlets and Enterprise JavaBeans™ (EJB) components		
WebSphere Studio Site Developer (Site Developer)	Windows 2000	5.0.x, 5.1.x
Usage: For application development of HTML and JSP pages		

2.4.3 Message Brokers

WebSphere MQ Integrator and Message Broker implement a transformation and routing engine that transforms the data received from the application servers or process manager in one XML format to either another XML format required by

the DirectCar company's existing backend claims application, or to an existing communication area (COMMAREA) format for the LGI company's systems. A *COMMAREA* is a CICS area that passes data between tasks that communicate with a given terminal. The area can also be used to pass data between programs within a task.

After the data has been transformed, it is routed to the appropriate backend system based on the content of the WebSphere MQ message. For requests from business partners, the EDIFACT data is converted into XML using a DTD imported as a message set.

The WebSphere Business Integration Message Brokers are configured in a domain for handling large workloads and providing fail-over support.

We use the Message Brokers because of:

► The continued support of the messaging infrastructure and reuse of the existing WebSphere Business Integration Message Broker infrastructure, already in use to expose LGI's backend systems.

► Support for all the data formats required by the applications: XML, COMMAREAs, EDIFACT data, and the multiple operating systems used by the merged company.

Runtime

The products used for runtime of the BPM engine part are listed in Table 2-5.

Table 2-5   Products used in the runtime of transformation and routing engine

Product	Platform	Version
WebSphere Business Integration Message Broker	AIX, Solaris, Windows 2000, z/OS	2.1.x
WebSphere Business Integration Message Broker	Windows 2000	5.x

Tools

The products for tools to create BPM engine part are listed in Table 2-6.

Table 2-6   Products used as the tools for developing BPM engine

Product	Platform	Version
WebSphere Business Integration Message Broker Control Center	Windows 2000	2.1.x
Usage: For application development of message flows		
WebSphere Business Integration Message Broker Workbench	Windows 2000	5.x
Usage: For application development of message flows		

2.4.4 Process managers

This section describes the process managers in our scenario. Process managers control processing of claims through the validate, investigate, and judgement stages calling the backend systems, where appropriate, through a WebSphere MQ message to the transformation and routing engine. They also allow conditional staff intervention based on meeting certain criteria.

The decision of choosing the specific products are based:

► Builds on the existing skills and products already in use by the LGI company.

► MQ Workflow provides support for role-based staff activities in long running interruptible flows.

► The tooling provides the ability to rapidly modify or redevelop new business process flows.

Runtime

The products used for runtime of the process managers part are listed in Table 2-7.

Table 2-7   Products used in the runtime of process managers

Product	Platform	Version
WebSphere MQ Workflow	AIX, Windows 2000	3.4, 3.5

Tools

The products for tools to create process managers part are listed in Table 2-8 on page 31.

Table 2-8   Products used as the tools for developing business processes

Product	Platform	Version
WebSphere Business Integration Modeler	Windows 2000	4.2.x
Usage: For application development of business process flows		
WebSphere MQ Workflow Buildtime	Windows 2000	3.4, 3.5
Usage: For application development of business process flows		

2.4.5 Backend transaction servers and data centers

This section covers the backend transaction servers and data centers for DirectCar and LGI.

DirectCar application server and data center

The DirectCar application server is configured to use WebSphere MQ as a JMS provider. Message Driven Beans (MDBs) were developed to consume messages from a WebSphere MQ queue using a message selector, then invoke the appropriate existing session beans (EJB components). The session beans call entity beans to access an Oracle database.

During claim validation, checks are performed against the policy details held in the database. In this case, the session beans use an RMI-IIOP call to an EJB interface that uses JCA services to route requests to TXSeries through the CICS Transaction Gateway (CTG). CICS COBOL applications within TXSeries access DB2® where the policies are held.

Secure connections using the Secure Sockets Layer (SSL) protocol over WebSphere MQ links have been established between LGI and DirectCar.

Reuse of existing systems leaving working applications unchanged. TxSeries and DB2 were part of an off-the-shelf insurance policy management application. The DirectCar company had chosen J2EE technologies with Application Server to develop their own claims handling system in conjunction with an Oracle database. The only new addition here was WebSphere MQ to integrate with the consolidated system.

LGI data center

The LGI data center is an existing system based on CICS. The WebSphere MQ-CICS bridge is used as the interface between the wider WebSphere MQ infrastructure and the CICS system. WebSphere MQ messages are transferred through theWebSphere MQ-CICS bridge that invokes CICS COBOL

applications. These issue SQL statements to access the DB2 databases containing the customer's policy and claims information.

Runtime

The products used for runtime of the back-end part are listed in Table 2-9.

Table 2-9   Products for runtime of the back-end transaction servers and data centers

Product	Platform	Version
DirectCar data center		
DB2	Windows 2000	7.2, 8.1.x
Oracle	Windows 2000	9.1
TXSeries	Windows 2000	5.0.x
WebSphere Application Server	Windows 2000	5.0.x, 5.1.x
WebSphere MQ	Windows 2000	5.3.x
LGI data center		
CICS Transaction Server for z/OS (CICS)	z/OS	2.2, 3.1
DB2	z/OS	7.2, 8.1.x
WebSphere MQ	z/OS	5.3.x

Tools

The products for tools to create back-end part are listed in Table 2-10.

Table 2-10   Products for developing back-end transaction servers and data centers

Product	Platform	Version
DirectCar data center		
VisualAge® for COBOL	Windows 2000	3.6
WebSphere Studio Application Developer	Windows 2000	5.0.x, 5.1.x
Usage: For application development of session and entity beans		
WebSphere Studio Application Development Integration Edition	Windows 2000	5.0.x, 5.1.x

Product	Platform	Version
Usage: For application development of Java Connection Architecture (JCA) EJB components		
LGI data center		
WebSphere Studio Enterprise Developer	Windows 2000, z/OS	5.0.x, 5.1.x
Usage: For application development of CICS COBOL programs		

Note: Product versions that are used in the System House solution are evolving with time. The versions listed here are correct as we write this book.

2.5 Extending the architecture

In "1.2, "Business goals" on page 7" and "1.3, "IT goals and constraints" on page 9" we describe the objectives for the merger between LGI and DirectCar. The implementation of these objectives is spread over at least three phases.

Phase one: completed

The following steps have been completed before the actions described in this redbook:

1. We automated steps of the process which improve the speed and predictability of handling claims.

2. We gave claim handlers a common set of interfaces to deal with both LGI and DirectCar claims and policy systems. This improved the flexibility with which staff handled claims on policies belonging to either company.

3. We automated the interaction with some external agencies such as licensing authorities. This automation was relatively simple and only involved a single interaction, such as checking a vehicle registration.

For further information, see the series of articles in IBM developerWorks®, *Merging disparate IT systems: Build a single integrated view for users quickly and with minimal disruption*, found here:

http://www-128.ibm.com/developerworks/ibm/library/i-merge.html

Phase two: automating external transactions

Th rest of this redbook focuses on e second-phase implemenations:

- ► Automate more complex interactions with external agencies, such as automating the tasks involved in the assessment of a claim:

    a. Issuing contracts with loss adjusters
    b. Selecting the loss adjuster to take on a particular case
    c. Starting the assessment process
    d. Monitoring its progress and the performance of the adjuster
    e. Receiving the adjustment report
    f. Making payment for the assessment report

Automating the external assessment of a claim

LGI, as a matter of corporate IT strategy, has decided to qualify the external claim assessor automation project as a pilot for moving its technology base towards greater use of open standards for its business process applications. They have already used Java 2 Enterprise Edition (J2EE) for the presentation layer, and have used Web services for simple interactions with external agencies.

The challenge now is to focus on the skills they need to build business process applications using BPEL on a J2EE platform, identify any technology gaps in introducing BPEL technology to the business, and evaluate the benefits of the new technology. Key among these are new tools for modeling solutions in BPEL and UML. Will these tools work together and speed up the development and deployment of the solution?

There are business and technical reasons for choosing the claim process for the pilot.

- ► From a business perspective, this decision should prove to be a valuable investment. It does not involve the whole business at once because it can be incrementally installed, gradually replacing the existing manual process.

- ► From a technical perspective, moving towards open standards such as Web services and BPEL is a high priority for processes involving business partners. The claim assessor process will also pilot integration of WebSphere MQ Workflow and WebSphere Business Integration Server Foundation, and the integration of a BPEL process using Web services with LGI's existing messaging infrastructure.

The new process must also use LGI's WebSphere MQ backbone to make use of services hosted on LGI and DirectCars systems. The backbone also provides the gateway to connect to services provided by business partners, and services provided by LGI to business partners. LGI's backbone has been evolving from a message bus connecting application components to becoming a service bus using SOAP/JMS and SOAP/Http as the main transport protocols.

Phase three: monitoring claims data in the future

The future actions that will have to be performed are:

1. Track the progress of a claim. Tracking a claim will provide information to clients who have questions about the progress of their claims. Potentially, it also could provide information for a Web-based, self-service claim application.

2. Collect information about the performance of the claims process to improve the performance of claim assessors and to provide information for improvements in the process itself.

Monitoring the claims process

Phase three requires monitoring the progress of a claim. The information from the process will be used in two ways:

1. Build a business dashboard to provide a real time business view of the claims process.

2. Improve the claims process by providing performance data. The data can be used by:
   – Business partners to review their performance,
   – The business analyst to investigate process improvements based upon real performance data input to the claim assessor simulation model.

We want to return to these topics in a future redbook to show how to use the business event data that the WebSphere family of products collects.

The rest of this and the second part of the book is divided in the following ways:

► The next chapter discusses the method and tools that we use to develop the new solution which is important to LGI in meeting their IT goals for better development productivity, better alignment of IT solutions with business goals, and getting solutions into production faster.

► In Chapter 4, "Business Process" on page 75 we use WebSphere Business Integration Modeler to develop the new business process.

► In Chapter 5, "System Architecture" on page 153 and Chapter 6, "Solution Architecture" on page 187 we use the Patterns for e-Business and Rational Software Architect to develop the new solution and extend the existing IT architecture.

2.6 Summary

In this chapter we covered the existing system architecture and application components. The design of the new external claim assessor automation system will be based on the existing skills and systems.

Part 2

Modeling

In this section we have four chapters to explain how to build a solution model and demonstrate how to build the model for the External Claim Assessor solution.

Chapter 3, "Our method" on page 39 describes the method used to build the solution.

Chapter 4, "Business Process" on page 75 shows how to build the new business process with WebSphere Business Integration Modeler and validate it using a simulation.

Chapter 5, "System Architecture" on page 153 uses the Patterns for e-Business and Rational Software Architect to build the system architecture.

Chapter 6, "Solution Architecture" on page 187 refines the architecture by defining all the interactions, interaction sequences and interfaces required to implement the solution.

3

Our method

One of our business objectives in 1.3, "IT goals and constraints" on page 9 is to:

► Demonstrate development productivity improvements by using modern tools and technology. The new system will implement a Service Oriented Architecture (SOA) based on Web services. It will be developed using Model Driven Development (MDD) and use open standards based technologies such as J2EE, BPEL, and UML2. The development project will serve as a reference for using these technologies in the future, showing how to achieve development productivity improvements.

In this chapter we describe the method we use to build the External Claim Assessor solution. Our approach is influenced by a number of current thoughts about software development. At times it feels that we could carry the project past our goal of building the solution. So we have choose from the tools that are available today. In this, we face the same practical difficulties as any other development project. As a result, we hope the choices we make are instructive.

39

3.1 Building the On Demand Business

What is an On Demand Business? IBM defines it as one whose leaders can see and manage their company as an integrated whole. This means that all sectors of the business must engage each other in a dynamic transformation of formerly isolated departmental operations into full business processes integrated across the company and outside to their customers.

An On Demand Business has four essential characteristics:

▶ *Responsive*, intuitively responsive to dynamic, unpredictable changes in demand, supply, pricing, labor, and competition

▶ *Variable,* flexible in adapting to variable cost structures and processes associated with productivity, capital, and finance

▶ *Focused*, concentrated on core competency, differentiated tasks and assets, with tightly integrated strategic partners

▶ *Resilient,* capable of managing changes and threats with consistent availability and security

For further information about the On Demand Business, see:

http://www-306.ibm.com/e-business/ondemand/us/overview/overview.shtml

LGI is transforming itself into an On Demand Business by:

▶ Transforming their IT infrastructure into one based on open standards that make LGI flexible and more responsive to business needs.

▶ Developing solutions focused on new or more efficient business processes cutting across customers, departments and suppliers

▶ Recognizing software development is a strategic business process that can drive their business success and should be driven by the same type of horizontal integration that is driving their other business processes

3.1.1 The on demand operating environment

The on demand operating environment has two goals: IIT simplification and business flexibility. It is a set of integration and infrastructure management capabilities that customers and partners can utilize, in a modular and incremental fashion, to enable the transformation to On Demand Business. It has two parts, an infrastructure management part and an integration part that addresses IT simplification and business flexibility goals. Look for the IBM Redbook, *On Demand Operating Environment: Creating Business Flexibility*, SG24-6633, which provides a roadmap to creating business flexibility in the on demand operating environment.

Infrastructure management is about enabling access to and creating a consolidated, logical view of resources across a network. Integration is about connecting people, processes and information in a way that allows companies to become more flexible to the dynamics of the markets, customers, and competitors around them. Figure 3-1 shows these concepts and lists the capabilities that are required to implement them.
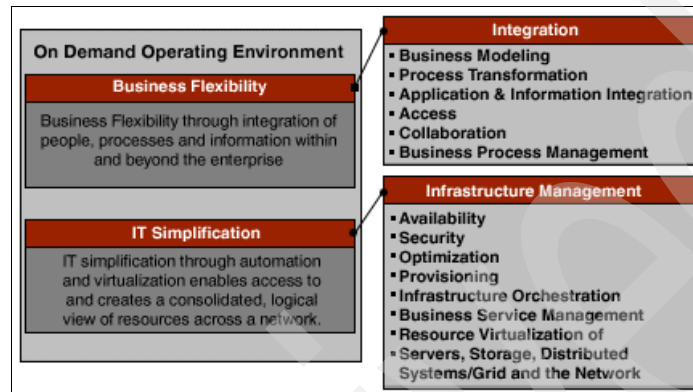
Figure 3-1   On demand operating environment

In the External Claim Assessor solution, LGI uses Business Modeling, Process Transformation and Application Integration. In the future, LGI plans to use Business Process Management to monitor day-to-day operations and analyze process performance. In the on demand operating environment these capabilities are realized using a Service Oriented Architecture (SOA). SOA is both about simplifying the operating environment by making it easier to mix and match capabilities, and about providing an integrated software development environment for building solutions[1] from services using service-oriented modeling[2].

3.1.2 Service-oriented modeling

Interest in service-oriented modeling is growing from a realization that:

Existing development processes and notations such as Object-Oriented Analysis and Design (OOAD), Enterprise Architecture (EA) frameworks, and Business Process Modeling (BPM) only cover part of what is required to support the architectural patterns currently emerging under the SOA

---

[1] Mike Perrow, *"Building the On Demand Business: Four Imperatives for Improved Software Development"*, found at
http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/imperatives-04.pdf
[2] Ali Arsanjani, *"Service-oriented modeling and architecture"*, found at
http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

umbrella. Thus, there is a need for an enhanced, interdisciplinary service modeling approach[3].

OOAD, EA and BPM are positioned by these authors in Figure 3-2.
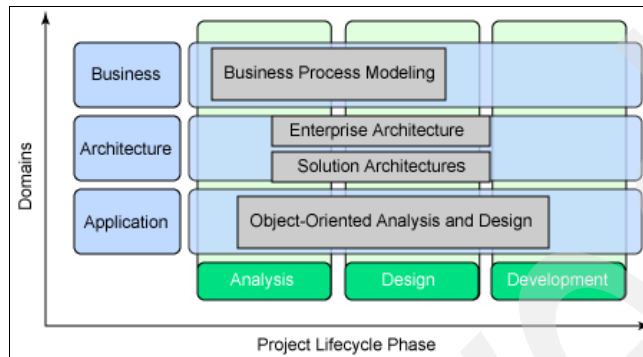
Figure 3-2   BPM, EA and OOAD positioning
(Zimmerman et al., Service-Oriented Analysis and Design)

The methods are performed by different people. EA is the province of the enterprise, solution and infrastructure architects, BPM of the business analyst, and OOAD of the IT specialist or application programmer. Service-oriented modeling requires the integration of the work of these practitioners plus extension of UML based modeling tools to deal with the artifacts of a service-oriented architecture, specifically:

► Services (WSDL)
► Service choreography (BPEL)
► Service bus (ESB)
► Services patterns

The authors' point is that OOAD, EA and BPM each provide some elements required to build a service-oriented architecture. However, they are not sufficient individually; the methods require integration. In effect, we have in software development the same departmental stovepipes we find in the use of software itself. Rotate the diagram 90º and you can see the similarity!

3.1.3, "The IBM Software Development Platform" on page 43 describes the strategy behind the IBM Software development platform to integrate the different roles and activities involved in developing a software solution.

---

[3] Olaf Zimmermann, Pal Krogdahl, Clive Gee, *Elements of Service-Oriented Analysis and Design*, June 2004 found at the IBM developerWorks Web site:
http://www-128.ibm.com/developerworks/webservices/library/ws-soad1/

3.1.3 The IBM Software Development Platform

IBM regards software development as a strategic business process. Software development benefits from horizontal integration just as other business processes do. IBM bases its software development tools on a common software development platform (SDP) in support of the concept that software is a business process, in an analogous way that the supply chain or customer relationship management are business processes (see Figure 3-3.[4]), the purpose of which is to support business transformation that occurs by integrating and automating horizontal business processes.
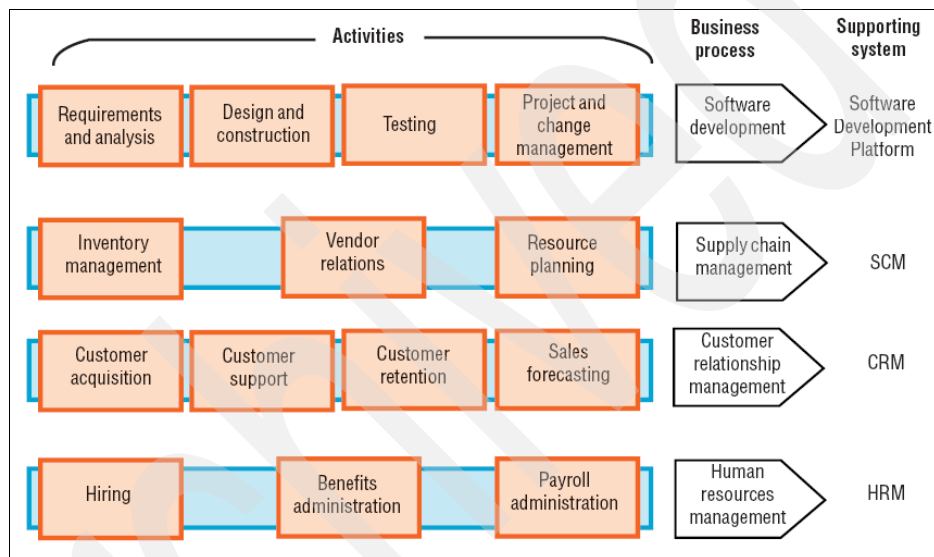
Figure 3-3   Software development: A strategic business process

The IBM software development platform provides broader tooling for an on demand operating environment than individual software development methodologies. The software development platform enables the integration of different departments, processes, and deliverables involved in software development as well as provides and integrates the specific tools that are required.

The goals of the IBM software development platform are to:

► Connect business needs with IT solutions.
► Enable teams of practitioners.
► Provide end-to-end visibility, cost containment and risk management.

---

[4] Alan Brown, *Realizing the IBM Software development platform*, April 2004:
`http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/SDP_WP`
`_Final.pdf`

Connect business needs with IT solutions

Business process modeling, information modeling, and the Rational Unified Process® are the main tools for connecting business needs with IT solutions.

We are focussed on using business process modeling to connect business needs to the IT solution in the External Claim Assessor scenario.

Enable teams of practitioners

The Eclipse software platform along with the Eclipse Modeling Framework (EMF) are the key technologies in the SDP to enable teams of practitioners to work together on a common solution model based on UML2 (Figure 3-4).
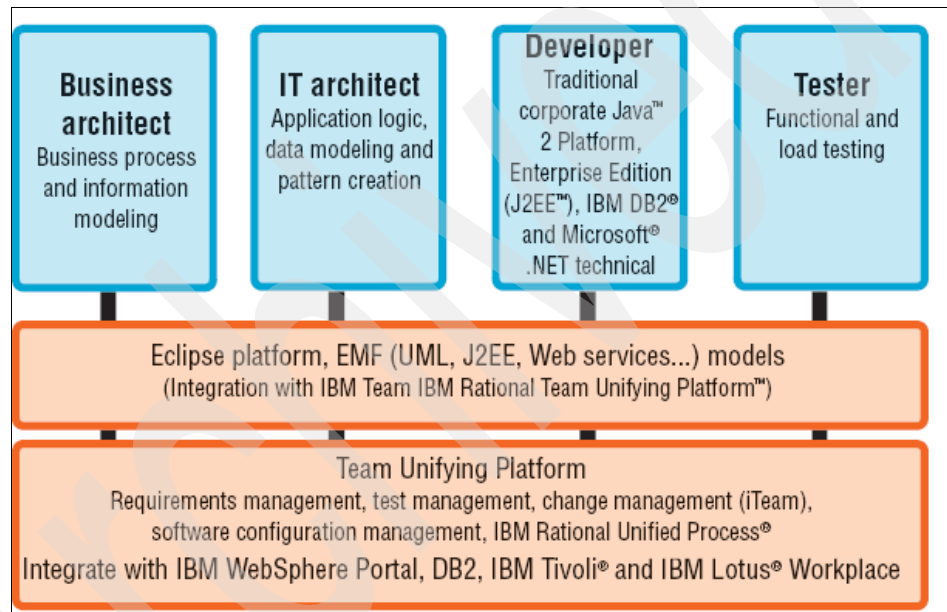
Figure 3-4   Software development platform (Alan Brown, op cit.)

The tools are designed around the needs of the different role players, but integrated on a common tools platform, able to share artifacts by using a common meta model, and unified by a common software development process including requirements, test, change and configuration management.

The tools we use all run on the Eclipse platform and share various characteristics. In particular, we focus on the following integration tasks:

1. Connecting the business process model created by the business analyst using WebSphere Business Integration Modeler with the architecture created by the solution architect using Rational Software Architect.

WebSphere Business Integration Modeler and Rational Software Architect have recently been integrated using UML2 and a method of accessing a common business model[5].

2. The IT process integration specialist builds the executable business process with WebSphere Studio Application Development Integration Edition using the process model from WebSphere Business Integration Modeler and interface definitions from Rational Software Architect.

3. The IT workflow specialist maintains the FDL process with WebSphere MQ Workflow Buildtime. The workflow specialist uses the process model from WebSphere Business Integration Modeler and WebSphere MQ Workflow and interface definitions from Rational Software Architect.

4. The application developer reengineers the claims applications created using WebSphere Studio Application Developer to use Web services by using the WSDL definitions and Web service and UML architecture being defined by the solution architect.

5. The service bus integration IT specialist builds the mediations that are deployed on the enterprise service bus by WebSphere Business Integration Message Broker using WSDL and schema interface definitions maintained in Rational Software Architect.

End-to-end visibility, cost containment, and risk management

The Rational Unified Process (RUP®) is the main tool for providing end-to-end visibility, cost containment, and risk management in the software development process.

In this project we have not used the RUP to define and manage the development process, Requisite® pro to manage requirements, ClearQuest® and Clearcase to manage changes, or any of the testing or performance profiling tools that are available in the software development platform from Rational.

3.2 Building the External Claim Assessor solution

The approach used by the System House scenario team who developed the solution for this redbook was based on methods used by IBM service teams in real-life projects. The scenario team modeled the business process in workshops to collect requirements, document the as-is business process, and define the to-be business process. They then took the model from the business process analysis as the starting point of a business driven development approach to design and implement the solution.

---

[5] See Appendix B, "Integration considerations" on page 505 for details.

The method is described in the following sections:

1. Section 3.2.1, "Roles and responsibilities" on page 46 describes all the roles who participated in building the solution.

2. Section 3.2.2, "Responsibilities and contract-based development" on page 52 describes how the project is organized and what deliverables are produced.

3. Section 3.2.3, "Gather business requirements though modeling workshops" on page 55 discusses the value of workshops in the process.

4. Section 3.2.4, "Establish a Reference Architecture" on page 56 shows the model for the staging the refinement of architecture into design which was followed in the development of the solution.

5. Section 3.2.5, "The Patterns for e-business layered asset model" on page 58 introduces the Patterns for e-Business stages which are used in the first step of the architectural refinement.

6. Section 3.2.6, "A process for using the Patterns for e-business asset model" on page 59 provides more detail about the method of applying the Patterns for e-Business layered asset model to a process integration solution.

7. Section 3.2.7, "Use a Model Driven Development approach" on page 67 discusses the value of the solution development team using a model driven development (MDD) approach to take the solution from initial business process model to implementation.

8. Section 3.2.8, "Tool chains" on page 70 describes how to plan the MDD approach by understanding how the tools used by the different roles involved in the development are integrated in the software development platform.

3.2.1 Roles and responsibilities

The development of the External Claim Assessor solution is a team effort which involves business sponsors and business analysts, IT architects and specialists, and end users. See the business context diagram - Figure 1-1 on page 11. Understanding which roles are involved is important from at least two perspectives:

1. It establishes who is responsible for the connection between business and IT goals. This is important because the development of software for the On Demand Business is regarded as a business function, not as purely a technical software engineering procedure.

2. It identifies who the technical specialists are, what their requirements on the software development platform will be, who will be dependent on input from whom, and thus what artifacts need to be exchanged using the software development platform.

The roles who are involved are:

► Business sponsor is the business executive responsible for auto claims. This person should be at director level.

The business sponsor's role is to set the overall goals of the claim assessor project, and to ensure members of the claims department are able to spend enough time with the project team to be effective in describing their requirements and reviewing process changes.

The sponsor will have a limited involvement in day-to-day development. They will use graphics of the business process and reports from the Modeler to create presentations to justify the business case and to explain the proposed process changes to executives in the rest of the business

► User representatives

One or more representatives such as a claim assessor, claims supervisor and a claim handler, will be present from the user community. They will be particularly valuable in documenting how the existing claims process works, what possibility for improvement exists, and to assist with the definition of and phasing in of the improved process.

The user representatives will use graphics from the tool to create presentations to explain and discuss changes with their colleagues.

► Project manager

The project manager is responsible for the delivery of the claim assessor solution on time and budget and meeting its requirements.

The project manager will work with the business analyst and use reports from the Modeler to identify tasks and responsibilities.

► Workshop facilitator

The facilitator's role is to manage the workshop sessions. The facilitator may be the project manager, business analyst or process specialist, or someone with special experience in running workshops. The important point is that someone has responsibility for driving workshops through and getting the results documented.

The person selected to document the workshops will use WebSphere Business Integration Modeler or Microsoft Visio® to capture process models. Progress on defining the process model can be written up after each workshop. Alternatively, if it suits the participants, it is often more effective to work with the tool interactively on a projector or screen sharing in a remote meeting.

► Business analyst and process specialist

The Business analyst is responsible for the LGI insurance company's business processes. The analyst is responsible to the business sponsor for meeting the goals laid out for automating the claim assessor process.

The Business Analyst performs tasks such as:

– Advising on current status and future directions of business transactions in relation to business goals

– Participating in purchasing decisions

– Working closely with the Solution Architect in designing new solutions

– Advising the development team throughout the solution development phase, with respect to detailed requirements or business trade-offs for the solution

– Working closely with the Solution Tester to develop a plan to validate the solution after delivery and establish test scenarios

– If necessary participating in the certification of solutions as meeting the expected business goals and being ready for production

– Defining or modifying business rules within an existing application

– Documenting business processes and is responsible for certifying the processes as compliant with statutory or voluntary requirements

– Providing content for Web sites describing business processes

The business analyst will use the WebSphere Business Integration Modeler. The business analyst will also use the tool to build and document the process, define business measures related to the goals of the solution, and simulate the process to verify the process will meet the goals set for it.

► IT architect*s*

– Solution architect

The IT solution architect is responsible for mapping the new claim assessor process into an IT solution. The solution architect is responsible for mapping the business goals of the solution into the IT goals (commonly known as Service Level Agreements, or SLAs) of the solution. These include measures of quality of service, response times, costs, time to develop and increasingly mapping performance of the solution as a whole to business measures set by the business analyst.

The solution architect performs tasks such as:

• Designing a new solution, application or function based on the specified functional and non-functional business requirements

• Defining the interface of new business services

- Writing application design specifications, including clear specifications of functional and nonfunctional requirements for the solution

- Modifying existing services to react to new requirements

- Planning solution upgrades and rollout of new capabilities (applications and products)

- Defining expected behavior of a service in terms of performance, service levels and customer satisfaction

- Defining the task flows for enabling the business service

- Using analysis tools to define information and task flow such as BPR (Business Process Re-engineering) tools like Holosofx® or OOA (Object Oriented Analysis) tools like Rational Rose®.

- Interacting with business people to understand a particular business process or business requirement

- Interacting with application developers to interpret business requirements in technical terms

The principle tool to be used by the solution architect is the Rational Software Modeler or the Rational Software Architect rather than WebSphere Business Integration Modeler. The architect will use WebSphere Business Integration Modeler to understand the business process model and share artefacts with the Rational tools.

– Enterprise/Infrastructure architect

The principle responsibilities of the enterprise architect are setting IT infrastructure standards and the business's strategic IT goals. The infrastructure architect is responsible for realizing the standards and goals in the IT infrastructure and ensuring new solutions can be realized meeting those standards and goals. Beyond the scope of this scenario the architect's responsibilities include capacity planning, meeting response time or service level goals, maintenance of the IT infrastructure, disaster recovery and system back up and restore.

– Data architect

The data architect is responsible for data modeling in the enterprise and the definition of database tables and the relationships between them. They will have deep industry expertise as well as technical knowledge about relational data models.

We have omitted any consideration of data models from this study because we focused our resources on how to perform process-based integration. When we need to chose the tools to model and build the solution, consideration of the data will be a high priority.

– Security architect

We have omitted security, privacy and conformity with data protection and other legislation from the scenario.
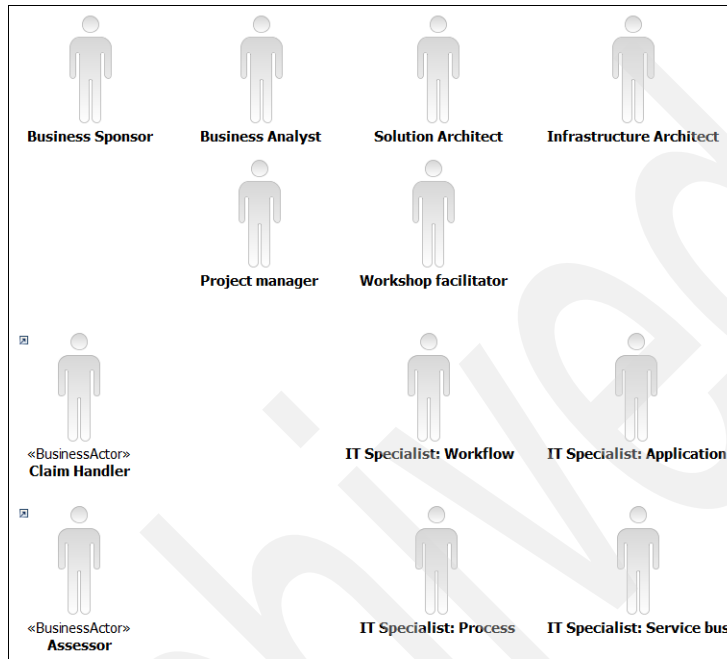
Figure 3-5   The roles involved in building the solution

► IT Specialists

IT specialists can be called into the early stages of the development by the IT architect to clarify how the existing system works. Having specialists attend the early process design meetings to represent developers proves its worth when the specialists are refining the design. The specialists can check the process model has sufficient detail from which to work.

The specialist's principal responsibility is to export the business analyst's process model (BPEL) from WebSphere Business Integration Modeler and the IT architect's model (UML) from Rational Software Architect and then to implement their components using the products for which they are responsible.

Application developers perform tasks such as:

– Developing the business services according to the architect's model

– Deciding on the specific implementation of required task flows (workflows, message flows, and so forth)

- – Writing EJBs to wrap internal and external resources or implement new services
- – Writing message flows and message sets to implement mediations between services
- – Configuring WebSphere MQ Workflow to execute FDL and integrate it with manual and automatic activities
- – Transforming BPEL from a process description to a process execution definition and integrate the BPEL flow with manual and automatic activities
- – Unit testing each application component against specifications and testing the validity and basic performance of the developed application components
- – Developing user interfaces, EJB, or servlet frameworks to contribute to shared services for re-use within other solutions

We had four IT specialists. We combined the development and runtime roles because our focus was on development, and the only runtime task was to deploy the solution.

a. WebSphere MQ Workflow specialist

The WebSphere MQ Workflow specialist is responsible for taking the new (*to be*) claims process defined by the Business Analyst and modifying the existing (*as-is*) claims process running on WebSphere MQ Workflow to call the new claim assessor subprocess.

The specialist's principle tool is WebSphere MQ Workflow Buildtime. However, this specialist will also need to refine the to-be process in WebSphere Business Integration Modeler and export FDL suitable for the workflow Buildtime.

b. WebSphere Business Integration Message Broker and WebSphere MQ specialist

This specialist is responsible for the underlying WebSphere MQ infrastructure and providing message flows to connect service requesters and providers. This specialist's principle tools are WebSphere Business Integration Message Broker workbench, and WebSphere MQ explorer.

c. WebSphere Business Integration Server Foundation and WebSphere Studio Application Development Integration Edition specialist

This integration specialist is responsible for taking the to-be claim assessor BPEL process and implementing it using WebSphere Studio Application Development Integration Edition.

d. WebSphere Application Server and WebSphere Studio Application Developer specialist

The WebSphere Application Server specialist is responsible for building and deployed the services used in the claims automation. We used WebSphere Studio Application Development Integration Edition to write the services.

3.2.2 Responsibilities and contract-based development

The method involves these multiple roles working together at each stage of development. It is punctuated by producing deliverables such as the business process model, and the solution architecture document. These documents form contracts between the people in the team and mark the progress of the project as a whole. This is not strictly a waterfall process, because work on the business process model proceeds in parallel with solution architecture and some of the implementation. But the work is also structured in stages with deliverables and formal approval to the completion of each stage. To the extent the software development platform can support iterative development, though mechanisms such as refactoring, then there is greater scope for working in parallel, and to the extent that it is difficult to reverse changes back into the process model the development needs to proceed in stages.

As we discuss in 3.2.8, "Tool chains" on page 70, there are limitations today to the extent the software development platform supports refactoring across all the tools. As a result, truly iterative development is costly and the work on the model does have to proceed in stages.

It is important to recognize there must be some friction in the refinement process resulting in development proceeding in a series of stages. Staged development is not simply a consequence of the difficulty of exchanging models between tools. It reflects the need for changes to be agreed between members of the development team before each member goes off to refine their part of the solution. Making changes to design decisions always has a wider impact than anyone supposes. Therefore, the impact of the changes needs to be assessed before committing to the project. It is part of the project manager, business analyst, and IT architects' jobs to put a brake on proposed changes that emerge during refinement that will change requirements and architecture, and consequently affect other parts of the implementation. The benefit of being able to exchange models between tools without friction is not to eliminate the need for development stages, but to make the implementation of agreed upon changes less costly.

The approach we took in working together was for the business analyst to document the business process using WebSphere Business Integration Modeler and to regard the model as a programming specification. The architect and IT specialists were free to use, extend, and modify the model to meet the objectives

of the implementation. But the model is not a hard and fast design of how the process works.

We treated the business process model produced by the analyst as the definition of what the business process is, and not a specification of how the process actually operates. At some stage the Business Analyst should review the IT model, and agree it is a reasonable implementation of the business model. This is similar to the approach of traditional program development, and is still some way from the goal of model-driven development.

Figure 3-6 illustrates three different ways of understanding the relationship between the business model and the IT model.

► The business analyst develops a process model, and exports it (A in the figure). The IT architect views the process model as a specification of process requirements, and may import none, some or the whole of the model to assist in the development of the IT model.

► The business analyst develops a process model, and exports it (B in the figure). The IT architect imports it, then develops the model further. The Business Analyst might be able reimport the IT model, because it is all in the common BPEL language, and pick up the IT refinements. There is a risk the refinements might obscure the key elements of the model for the business analyst.
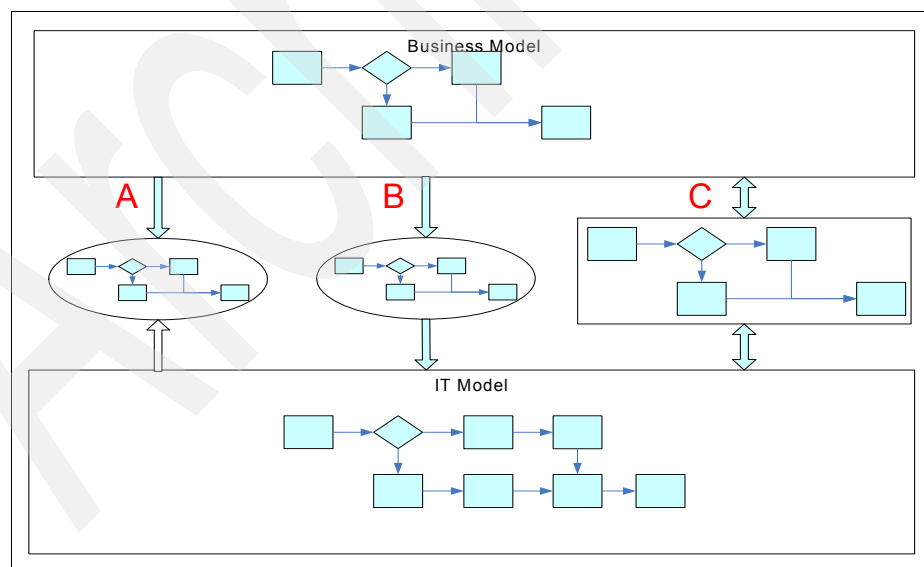
Figure 3-6   Different approaches to implementing a business model

► In approach C, we have a shared representation of business components and processes on which the business analyst and IT architect work jointly. The goal is to be able to find different abstractions of the same business processes and components that meet the modeling needs of both the business analyst and the IT architect.

This is almost a maturity model for the progress in automating business process modeling technology. (A) represents commonly used graphical techniques, from backs of envelopes, through Visio, to structured business process modeling tools such as WebSphere Business Integration Modeler. (B) illustrates the state of current tools that are based on executable process languages such as BPEL, and represents the current state of the art. (C) perhaps represents where the technology is moving to next.

Three contracts

We identified three main phases of architectural development based on the refinement of the solution model (Figure 3-7). This resulted in having three contracts. The first phase is covered in Chapter 4, "Business Process" on page 75 and is the development of the business process model. The business model is called a Computationally Independent Model, or CIM. It is described in BPEL, used as a process description language.
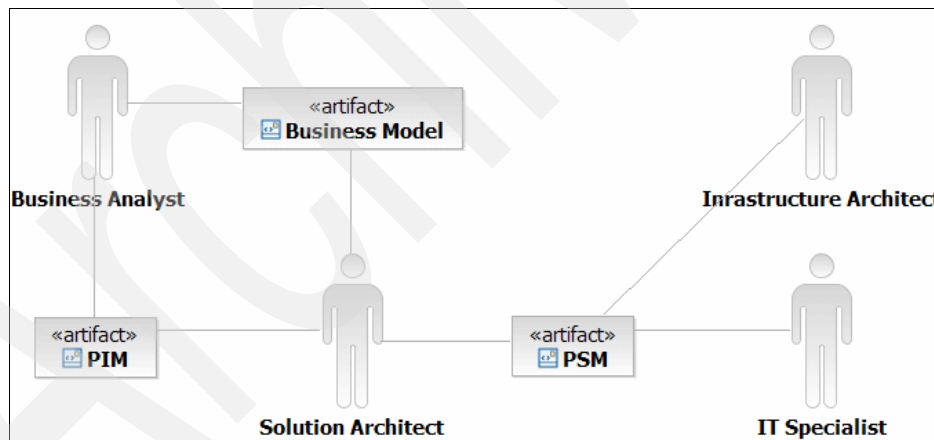
Figure 3-7   Contracts between roles involved in process implementation

1. The business model (CIM) forms a contract between the business analyst and the solution architect who is responsible for developing the solution architecture.

   Here we determined which parts of the business model are candidates for software automation. This is negotiated between the decision maker,

business analyst, and IT architect. The decisions were based on costs, time-to-market, and strategic goals (see 1.2, "Business goals" on page 7).

The architecture is developed in Chapter 5, "System Architecture" on page 153 and Chapter 6, "Solution Architecture" on page 187. It shows how the solution architect incorporates the business model defined in WebSphere Business Integration Modeler into the platform independent model (PIM) of the solution developed in Rational Software Architect.

The PIM model should be reviewed by the business analyst to check that the refinement has not identified new process issues or misinterpreted the intent of the business process.

2. The PIM forms a contract between the solution architect and the business analyst who verifies that the solution architecture will implement the business model.

The architect, with the assistance of IT specialists and the infrastructure architect, maps the PIM to a platform specific model (PSM) making use of the technologies available at LGI. Mapping the solution to the PSM forms the remainder of Chapter 5 and the whole of Chapter 6. The PSM should be approved by the IT infrastructure architect and reviewed by the IT specialists.

3. The PSM forms a contract between the solution architect, the infrastructure architect and the IT specialists who are responsible for developing the solution implementation.

3.2.3 Gather business requirements though modeling workshops

Using modeling in workshops involving both business and IT roles is an extremely effective way to elicit requirements. Because there are no right process or architectural answers (though there are wrong ones!) the discussion around applying process and pattern models is extremely effective in building a common understanding of the requirements and of the proposed solution. In this it draws heavily on the best practices followed by business analysts involved in business reengineering.

Another advantage of a workshop-based approach is that it plays an important part in bringing together the whole team who are involved in sponsoring, building, and eventually using the solution. Building a solution is not just about building software that works and meets its requirements. It is about building a solution that works and delivers the benefits required of it. There are some well written accounts of the pitfalls of letting the concerns of software engineers hold too much sway in building solutions in *The Inmates Are Running the Asylum: Why High Tech Products Drive Us Crazy and How to Restore the Sanity* by Alan Cooper ([6].Early involvement of everyone concerned with a business process change is key to success.

The involvement needs to be structured and understood by the representatives of all the affected parties for it to be effective. There are a number of books specifically on reengineering business processes that focus on how to capture a business's existing processes and how to undertake the task of improving them. An accessible place to start is the redbook, *"Continuous business process management"*, SG24-6590, found here:

`http://www.redbooks.ibm.com/abstracts/sg246590.html?Open.`

This book describes one method of developing business processes specifically for the WebSphere Version 4 platform.

Designing a new business process, and automating it, is a only part of the overall task of business reengineering. The development of a new business process needs to be part of the larger task of implementing the change. The process model is an effective means of structuring discussions about the business process with the sponsors and users of the business process[7].

3.2.4 Establish a Reference Architecture

Establishing a high-level architecture begins during the earliest stages of the solution design and delivery process. As illustrated in the model in Figure 3-8 on page 57, architectural specifications go through a transition over time from the initial very high-level conceptual representations to detailed, specified, and physical-level representations. The stages we followed are numbered 1 to 4.

---

[6] Published by Macmillan, 1999, ISBN 0-672-31649-8.

[7] There is a well written case study provided by Chris Booth of Strategic Thought in Charles Brett's book *Five Axes of Business Application Integration*. It is based on his work with Criterion Assurance and shows the value of using process models in requirements workshops on the overall success of a project.
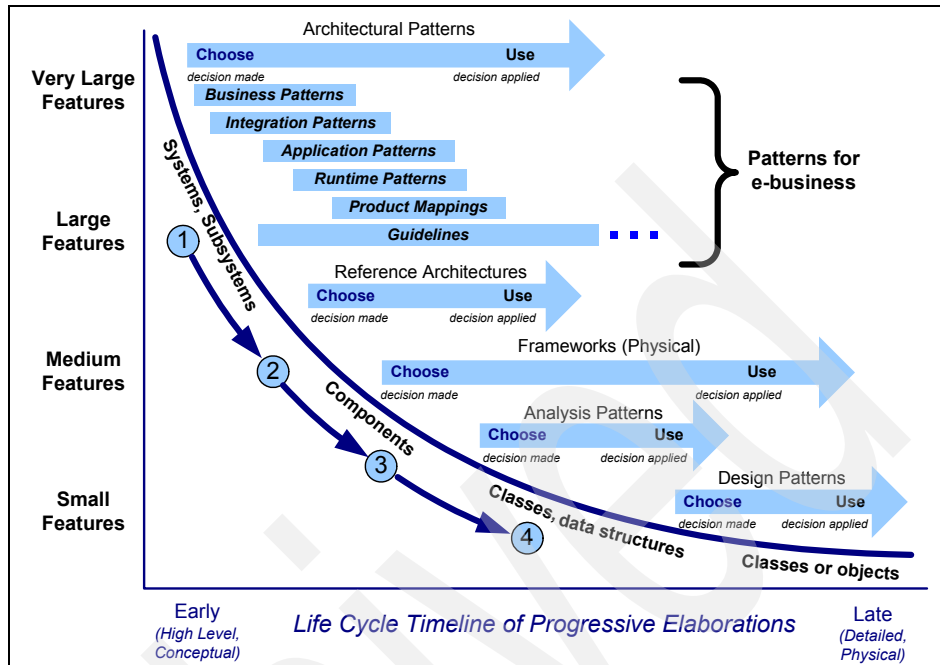
Figure 3-8   Pattern and asset use across the life cycle

1. Our architectural process started with using Patterns for e-business process to define a reference architecture.

   Patterns for e-business are very useful in establishing the high-level architecture. A well-defined pattern selection process is a key feature of e-business patterns. The pattern selection process accelerates the design work and provides traceability from business requirements to architectural decisions. The pattern selection process begins with analyzing a small number of architecturally significant requirements and drivers. That information yields high-level application and runtime topologies that support the business and integration needs of the solution. The use of the Patterns for e-Business method is described in the next section and performed in chapter 5.1, "Selecting the architectural patterns" on page 154.

2. Document the reference architecture using UML using Rational Software Architect. The reference architecture identifies the technologies to be used as well as the components to be built and deployed. This information is required by the infrastructure architect to plan the physical deployment and by the IT specialists to identify which technologies to use to build the components.

3. The third step is to analyze the interactions and interfaces to provide the IT specialists with a UML model of the solution, WSDL interface definitions and

BPEL descriptions of the business process. We do this refinement in Chapter 6, "Solution Architecture" on page 187.

4. The design steps are carried out by the IT specialists in the subsequent chapters on the implementation.

3.2.5 The Patterns for e-business layered asset model

A major part of the architectural design was done using the Patterns for e-business layered asset model.

The Patterns for e-business approach enables architects to implement successful e-business solutions through the re-use of components and solution elements from proven successful experiences. The patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in a way that each level of detail builds on the last. These assets include:

► Business patterns that identify the interaction between users, businesses, and data

► Integration patterns that tie multiple business patterns together when a solution cannot be provided based on a single business pattern

► Composite patterns that represent commonly occurring combinations of business patterns and integration patterns

► Application patterns that provide a conceptual layout describing how the application components and data within a business pattern or integration pattern interact

► Runtime patterns that define the logical middleware structure supporting an application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes

► Product mappings that identify proven and tested software implementations for each runtime pattern

► Best-practice guidelines for design, development, deployment, and management of e-business applications

These assets and their relationships to each other are shown in Figure 3-9 on page 59.
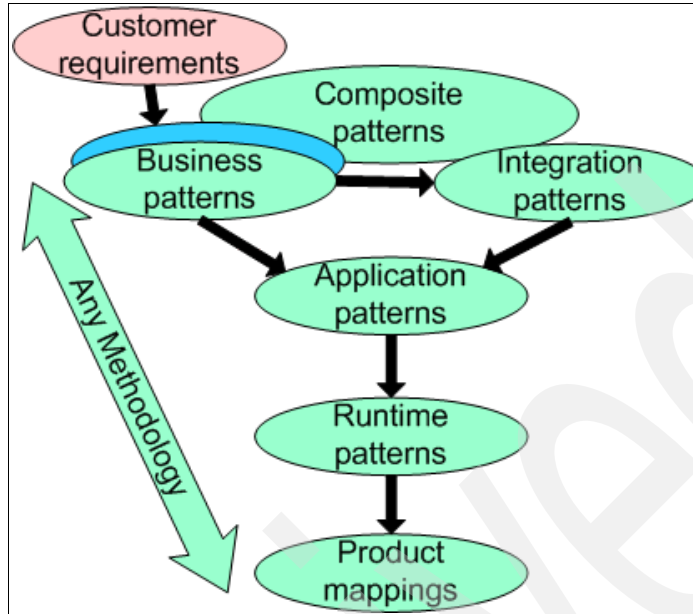
Figure 3-9   The Patterns for e-business layered asset model

Patterns for e-business Web site

The Patterns for e-business Web site provides an easy way of navigating
through the layered patterns assets to determine the most appropriate assets for
a particular engagement.

For easy reference, see the Patterns for e-business Web site:

    http://www.ibm.com/developerWorks/patterns/

3.2.6 A process for using the Patterns for e-business asset model

The approach we follow in Chapter 5, "System Architecture" on page 153 for
designing the external claim assessor solutions uses the Patterns for e-business
pattern selection process (Figure 3-10).

A multistage sequence of steps is used, each of which has the objective of
analyzing a specific subset of key requirements and yielding a result that is
based on selecting from a specific subset of the patterns using those
requirements as selection criteria. Each step provides input to the next step, in
effect providing guiding constraints that progressively narrow the selection
process for the next subset of patterns to be considered. Ultimately, this results
in architectural-level topologies and product mappings that form the basis for a
solution architecture.
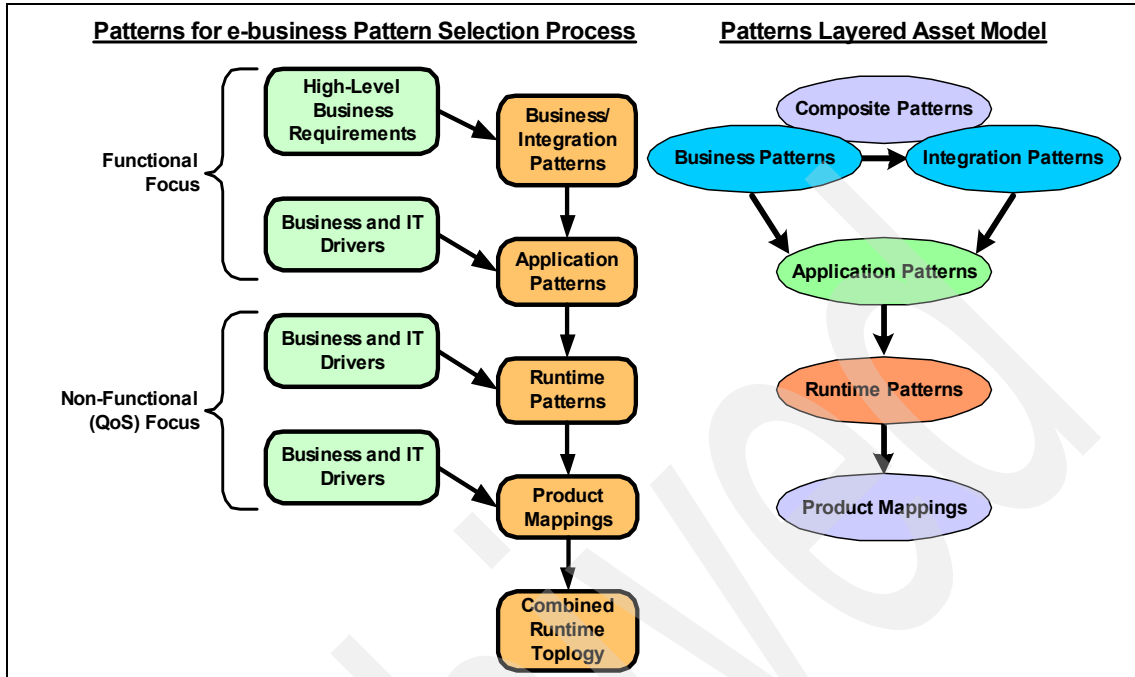
Figure 3-10   Patterns for e-business pattern selection process and layered asset model

Recently a method based on UML2 for applying the pattern selection process for process integration patterns has been developed.

Principles of the process

Figure 3-11 on page 61 summarizes the Process Integration Design Approach [PIDA][8]. There are several distinctive aspects to this approach:

► It focuses primarily on integration aspects of the solution architecture.

► It is based on UML2 and is particularly focussed on collaboration and interaction analysis

► It places special emphasis on behavioral aspects of the solution: the things that happen when components collaborate and interact with each other.

► It puts a lot of emphasis on the use of non-functional requirements in the component selection process.

---

[8] PIDA has been filed as a patent with the UK patent office (GB920040072GB1 - METHOD AND APPARATUS FOR INTEGRATING ELECTRONIC SYSTEMS). See also the presentation by the author of this patent, Paul Verscheuren, *"Business Integration Patterns"*, found at, http://www-106.ibm.com/developerworks/patterns/library/w19.pdf

► Its analysis technique can be applied iteratively and recursively to account for the fact that integration aspects range from simple to complex, providing a way to iteratively decompose and recompose an integration related scenario until it has been understood to a sufficient level of detail. (Note the symbols representing iteration in Figure 3-11.)

We chose this new method of doing integration design for a number of reasons:

1. The team involved had previously used the Patterns for e-Business approach for designing integration architecture and found it effective.
2. Conversely we have found UML modeling difficult to apply to model integration of existing solutions. The literature tends to focus on building new solutions from designing components and classes from scratch and there is little practical guidance on using UML to model adding or changing one or two capabilities in a complex system.
3. By combining Patterns for e-Business with UML, PIDA provides an approach to using UML to model integrating existing complex systems.
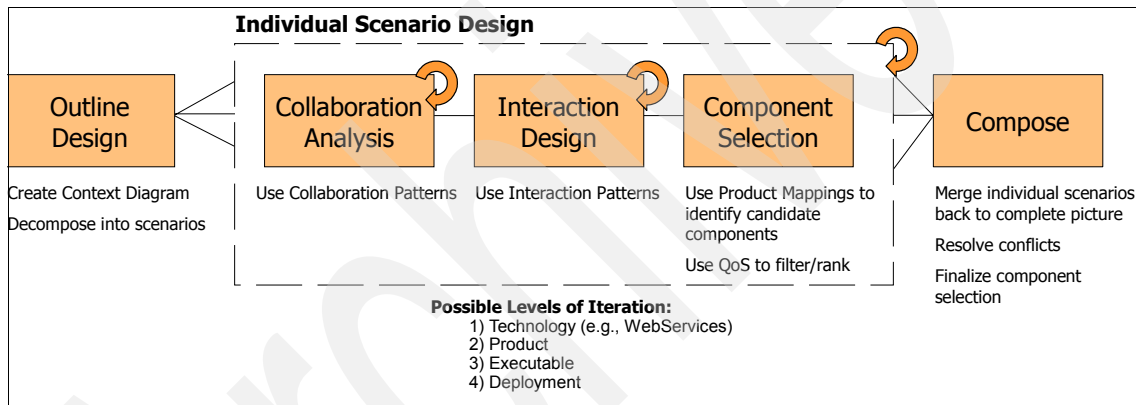
Figure 3-11   Approach to designing process integration solutions

Collaboration and interaction

When we first envision a system, we approach it from the point of view of its desired behavior. The desired behavior of a system is the result of several components collaborating to support a well-defined business need. Because we approach the system from an overall vision of its business function, we do not know the detail of interactions and have only a generic understanding of the kind of large-scale collaborations we expect.

Starting with the *business story* underlying the system, the analysis consists of identifying the collaborations required in the system and decomposing them, while gaining better understanding through studying their interactions. At some point, we reach a view of the collaborations and their interactions that enables us

to identify which IT components we will need and map them to prebuilt implementations in products available off the shelf.

Eventually, our knowledge of the components and interactions in the system-to-be has to be precise. However at the beginning of the analysis, when we do not know all the detail, and later, when we want to select the detail to communicate, we want to be able to control the level of detail being represented so we do not become overwhelmed at the expense of understanding the larger picture. The solution is to regard a system as being composed of parts which can be simply represented at any level of detail, without being wrongly represented.

This very powerful notion is termed *fractal*. It enables us to model and describe systems consistently at whatever level of detail is appropriate for the purpose. For example, we can show the collaboration between two components of the system as a simple line representing their relationship, or we can decompose the line into more and more components and their relationships which are all represented by that simple line. This principle is illustrated in.Figure 3-12.
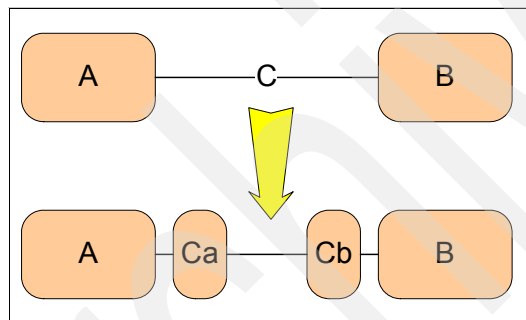
Figure 3-12   Fractal decomposition of a collaboration

The important point is that the representation at the top is no less correct than the representation at the bottom. They both represent the same model in different ways. Conceivably, a UML editor could be implemented that would switch between these views.

Accordingly, any concept we use in the behavioral analysis of such systems must be *scale-invariant;* must be usable regardless of whether we are addressing a very large system or any one of its parts. This principle lies behind describing the architecture of a system in a model which can then be giving , or handed-off, to IT specialists to implement, while retaining exactly the model originally produced by the architect.

Important terms

The following concepts are important in understanding Patterns for e-business:

Collaboration	A Collaboration represents the execution of any set of computational operations distributed across a particular space (computer network) and time.

> **Note:** The execution of the collaboration is governed by software artifacts (programs, objects, and components), but these artifacts are not the collaboration.

Interaction	An Interaction is a collaboration originating from a single operation or event.
Context	The data (including state data, programs, and executable rules) associated with a collaboration forms its (data) context.
Quality of Service	A collaboration may be subject to a set of Quality of Service (or QoS) requirements that constrains its execution.
Fractal	A Collaboration is a fractal if both compositions and decompositions of collaborations are collaborations.

As-is and to-be issues

It is tempting to think of the development of an IT system as starting from nothing at all and proceeding in a straightforward manner through phases of analysis, design, and construction, whether in a single pass or many passes. However, very few IT projects are *green-field*, smoothly running linear projects. Almost always, there are some preexisting elements (or even a whole system) that need to be modified or simply incorporated. This is almost axiomatically true for process integration projects, which by definition consist of making existing systems work together, typically in a new way. It is also true for the External Claim Assessor solution which must not only use existing components, but must be integrated with existing processes.

Therefore, a central concern in architecting and designing for process integration must be to keep track of the evolving distinction between existing, or as-is, and intended, or to-be, components and configurations.

This distinction is supported by the notation used in the Patterns for e-business that clearly shows the existing and new components of a solution, and also provides for external context elements (such as those which do not take part in the operation of the system but do interface with it, often in well-defined ways).

Quality of service

The distinction between *functional* and *nonfunctional* requirements has long been recognized in IT methodology. However, the term nonfunctional is not very

helpful because of its negative characterization. (covering those requirements that cannot be declared in terms of a particular business function). In fact, nonfunctional requirements are often the major influences on the choice of runtime and product mappings, and can influence the topology of the system. The characteristic feature of such requirements is that they pervade the system: they are generally attached to aggregates rather than to any particular interaction, are sometimes measurable with global statistics, and appear as an overall quality rather than a binary (yes/no) property. Therefore, the term QoS has emerged to designate those features in a system rather than non-functional requirements.

A QoS criterion: Governance

Perhaps because of its obviousness, an often underestimated QoS characteristic is governance. Governance describes how a system is composed of parts under different jurisdictions, both inter- and intracompany. Governance has a major influence on the design of a system from many perspectives. Not only will it influence the componentization (because it is rare for a single component, to be shared between jurisdictions) but also by determining which parts can be changed together in a new version of the solution, and which parts must coexist in different versions, has a major influence on the choice of technologies.

Identifying functional requirements

As to the functional requirements, they are defined by means of Use Cases at the System design level; the use cases describe the desired behavior of a system that is expressed at the Business level by Business scenarios and Business processes. In a process integration context, the functional content is primarily expressed by interaction relationships, which, in turn, determine the *topology* of the system. So instead, we call functional requirements *topological* requirements.

There are two kinds of requirements for a system:

Topology: The way in which connectivity and interaction between various components is supported in the system

QoS: Defined by means of statistical measures or as properties of aggregates. These properties may be attached to the system as a whole, to specific components, or to specific scenarios.

Work products and work method

These principles have been pulled together into a description of a set of work products that explain a method of working. We used this method while developing the External Claim Assessor solution.

The work products and work method that are described here are not intended to be a process that must be followed. The idea behind this approach is that it plugs in to your particular methodology and informs how to actually do various tasks, such as selecting the appropriate e-business pattern if you are following the Patterns for e-business design approach. In our case, we used it in the context of applying the Patterns for e-business layered asset model.[9]

The method involves the following activities:

1. Establish the boundaries of the solution in capability, behavioral and technical terms:

    – Define the target context, where we want to be in the future.
    – Define the situational context, where we are today.

2. Focus on the collaborations in the context in which they occur. The major activity of process integration is to rearrange the collaborations in a new context, rather than create new collaborations.

3. Exploit the fractal nature of the model in the refinement process:

    – Study the relationships between components. To look in more detail, decompose a component into constituents and study the relationships between constituents. The components themselves are black boxes.

    – It is the boundary of the components, called the interfaces to the components, that is important and is handed over at each stage of refinement. It is the architect's responsibility to scope the level of refinement and decide what is going to be handed over to the IT specialist or application developer for further refinement.

    – The hand over takes the form of a contract following the notion of design by contract.

4. From this understanding of the nature of the problem, the architect's work can be organized according to a simple work pattern known as *Distribute-Recurse-Converge*.

    a. Start with the pair of as-is and to-be context diagrams.

    b. Distribute: Apply various scenarios and use cases to the system to bring out the required collaborations.

        i. List the collaborations for further analysis.
        ii. Establish the overall topology of the solution.

---

[9] There are some internal IBM studies available, showing how the approach works with the IBM Global Services Method, the Rational Unified Process (RUP) and with the Enterprise Architecture methods such as the Technical e-Business Architecture Method (TeAM). Contact swithers@uk.ibm.com

c. Recurse: Analyze each collaboration for interactions. The first level collaborations will resolve into more detailed collaborations until we can identify single-event origins, which yield interactions

d. At any level of analysis attach the QoS to each collaboration.

e. Converge: Bring together the many strands of analysis and verify they are compatible and the resulting system is well balanced in meeting its various requirements.

Fractal thinking

Not everything that you need to know at a given level of abstraction you can know until you deconstruct the system further. On the other hand, you can know some things early on, and eliminate many alternative designs as unrealistic. The ideal situation is to find a balance between keeping design options open by not fixating on a particular solution, nor proceeding too deep into the analysis and design without taking into account what is actually available. Ignoring service considerations too long runs the risk of allowing the analysis and design into positions from which it is impossible to provide the qualities required.

This reality checking must also be present to take into account the existing (*black box*) components. Keeping track of available products is another set of boundary conditions. Do not wait until the end of the analysis and design to make sure that your work on each collaboration converges into a consistent whole.

In short, fractal thinking encourages us to hold all considerations at all scales (levels of analysis) at once.

Layer decisions

This process is not intended to be fully iterative, or performed with multiple passes. The architect must find the right points of handover in order not to revisit architectural decisions during implementation. It is for this reason that the physical realization of the architecture into product mappings takes place as soon as it is feasible, and long before the detailed design has been established. This is not a traditional approach in OOAD methods. The architect must take account of what is feasible in the implementation in order to minimize revisiting architectural decisions.

The aim is to delay specific decisions about the design as long as possible. Make decisions only when:

► All relevant elements are known.
► An external boundary constraint arises that forces a decision.

Decisions then occur at the first point in the analysis at which they can be fully justified. This point can be:

- ► Very early on, for example when the customer has chosen WebSphere Application Server to build the integration.
- ► At top collaboration level, for example when the business requires two different modes of operation, direct consumer access and batch.
- ► At product selection time, for example when we cannot use product X because we need compensating transactions.
- ► Even during final convergence, for example when we must use clusters to support high availability requirements.

The workshop method and design-by-contract supports this approach.

3.2.7 Use a Model Driven Development approach

Note: The information in this section and the next refers to the IBM Redbook *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105:

```
http://www.redbooks.ibm.com/redbooks.nsf/RedbookAbstracts/sg247105.html?Open
&S_TACT=105AGX46&S_CMP=SPLT
```

The use of modeling and metadata-enabled tools to move through the steps of architectural refinement and manage the complexity of software development is an aspect of Model Driven Development (MDD)[10]. In a recently published Redbook, Model Driven Development is defined as[11]:

A style of software development in which the primary software artefacts are models rather than code.

MDD is an approach in which:

– Application domain oriented models are the primary focus when developing new software components, not code or other platform artifacts.

---

[10] Compare the method we used for the External Claim Assessor scenario with a similar approach described in a series of articles *"On demand business process life cycle: Build reusable assets to transform an order processing system"*, found at
`http://www-128.ibm.com/developerworks/webservices/library/ws-odbpsum.html`
[11] Taken from Tracy Gardner, "MDD and Patterns Overview" in *Patterns: Model Driven Development using Rational Software Architect*, SG24-7105, found at
`http://www.redbooks.ibm.com/redbooks.nsf/RedbookAbstracts/sg247105.html?Open&S_TACT=105AGX46&S_CMP=SPLT.`
The next section on MDD benefits is also taken from this Redbook.

– Models are used not just as sketches or blueprints but as primary artifacts from which efficient implementations can be generated.

MDD proposes an approach that allows business processes to be captured in a Computation Independent Model (CIM) by a business analyst. The IT architect builds a Platform Independent Model (PIM) and then, working with the IT specialist, a Platform Specific Model (PSM). The PSM is mapped to code by the developers. The different steps are incremental and iterative and shown in Figure 3-13 on page 69.

Modeling and metadata standards are necessary for the tools to interchange software artifacts and to deploy the code and other artifacts such as configuration specifications on to the runtime platforms.

The artifacts produced at each stage have to be decided upon by the development team using a process model such as the Rational Unified Process (RUP). While this book focuses on the development path, other perspectives are equally important, such as testing, usability and systems management, as well consideration of how the solution will be adopted by the user community. Is the solution to be sold, provided as part of a service, or is it a custom development within an enterprise? People working on these aspects of the solution need to be continually involved with the development process to ensure development, testing, and usability are being employed on the most important aspects of the solution.
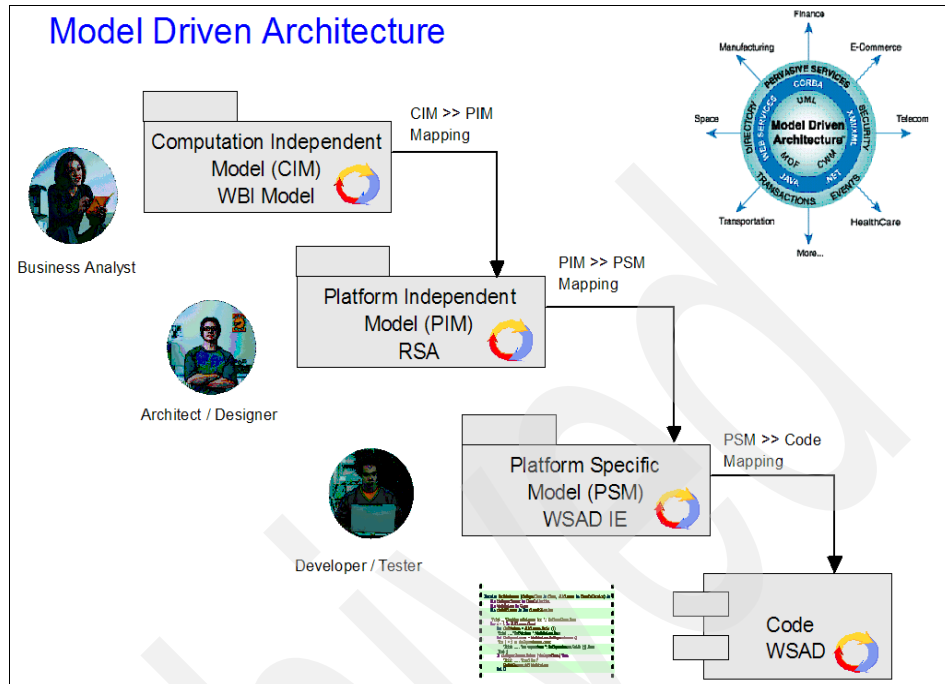
Figure 3-13   Model Driven Architecture

The key benefits of a thorough MDD approach are:

► Increased productivity

Model-driven development can reduce the cost of software development by generating code and artifacts from models and thereby increasing developer productivity.

► Flexibility

A change in the technical architecture of the implementation can be performed by updating a transformation which can then be reapplied to models to produce implementation artifacts following the new approach.

► Adaptability

When adding or modifying business function, only the behavior specific to that capability needs to be developed, the rest of the information needed to generate implementation artifacts has already been captured in transformations.

► Consistency

Model-driven development ensures that artifacts are generated consistently.

- ► Repeatability

  MDD is especially powerful when applied at a program or organization level. The use of tried and tested transformations increases the predictability of developing new function and reduces risk since the architectural and technical issues have already been resolved.

- ► Improved stakeholder communication:

  Models are much closer to the problem domain and easier to communicate. Improved stakeholder communication helps in the delivery of solutions that are better aligned to business objectives.

- ► Improved design communication

  Models facilitate understanding and reasoning about systems at the design level. The fact that models are part of the system definition, rather than documentation, means that the models can never be out of date and are reliable.

- ► Expertise capture

  Models capture expertise. Explicit expertise capture ensures that the knowledge of an organization is maintained even when experts leave the organization.

- ► Models as long-term assets:

  In MDD, models are important assets that capture the actions the IT systems of an organization perform. High-level models are resilient to changes in the state-of-the-art at the platform level, changing only when business requirements change.

- ► Ability to delay technology decisions

  When using an MDD approach, early application development is focused on modeling activities. This means that it is possible to delay the choice of a specific technology platform or product version until a later point when further information is available.

In the solution we are building, we only scratch the surface of the benefits MDD promises. But the experience LGI gains using modelling tools to describe the solution is the first step in a process of increasing maturity in the approach that will put them at an advantage in the future.

3.2.8 Tool chains

The Model Driven Development is an approach to software development in which the focus and primary artifacts of development are models (as opposed to programs). The models will be transformed from step to step until skeleton of

code can be generated. Figure 3-14 is an example from the Object Management Group (OMG) of the transformation from a PIM to a PSM.
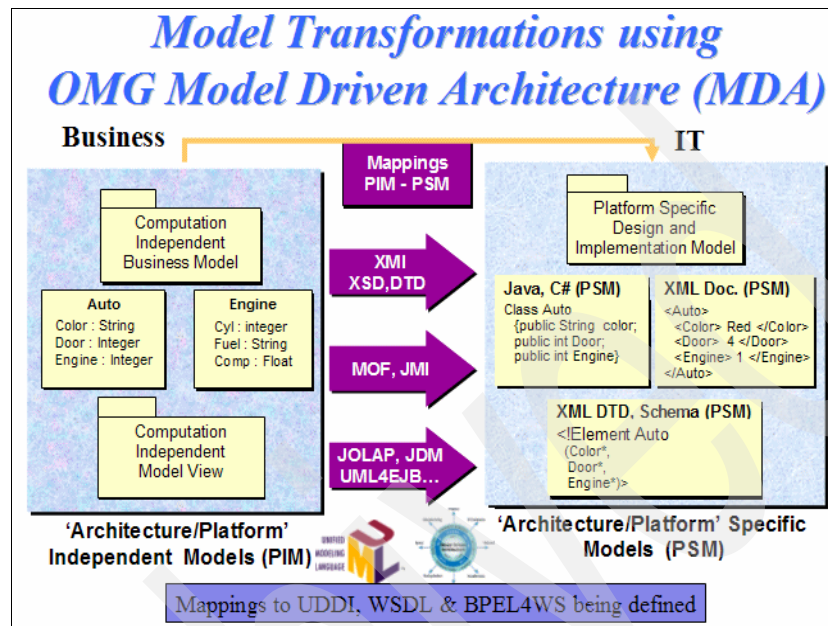
Figure 3-14   Model Transformation

To transform our business problem into a solution using a Model Driven Development approach we will be using different tools. We need to understand what type of tool is required for each role in the development, and then we need to understand how the model will be moved from tool to tool and what limitations there are in moving the artifacts from one tool to the next.

► Is information lost?
► Is the model interpreted in a different way?
► Can the model be moved in either direction from tool to tool, or are there changes made to the model which might prevent it being reimported into WebSphere Business Integration Modeler, for example.

The ideal is to be able to round-trip the meta-model. A sample process might look like this:

1. Create the process model in WebSphere Business Integration Modeler.

2. Import the model into WebSphere Studio Application Development Integration Edition to implement the process model which will involve changes such as:

   – Renaming

- – Adding new elements
- – Rerouting connections
- – Changing existing elements

3. Re-import into WebSphere Business Integration Modeler and:

- – Compare with the original model
- – Re-run simulations
- – Modify the new model and repeat the development process.

Today, the complete tool chain only supports a one-way, or *waterfall*, approach to MDD. Round-tripping the process model is something for the future. So in deciding on how to use the tools, it is important to take into account that after the model has been exported from one tool to the next, it is going to take more time and expense to make changes in an upstream tool. The changes in an upstream model also have to be manually refactored back into the downstream tools[12].

This is one of the reasons why you need to be clear about what the responsibilities and work products are of each role in the development process. We have suggested that the contract model is a good way to manage the interactions between the business analyst and the IT architect, and between the IT architect and the IT specialists. See Figure 3-7 on page 54.
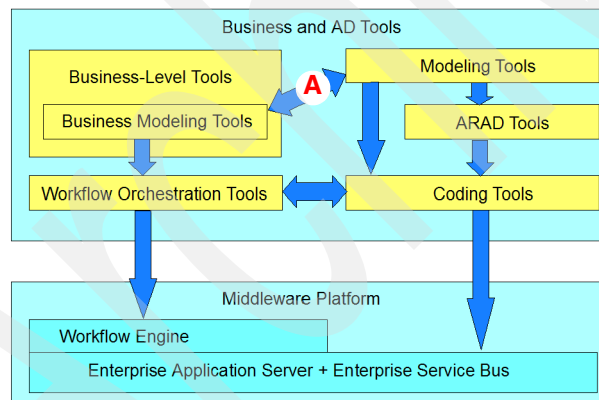
Figure 3-15   Tool Chain

Figure 3-15 shows the tool chain on which we based our development. The business process is captured in a business modeling tool by a business analysts. This model is used in two ways (see A in Figure 3-15). It is used to generate the process definitions for workflow orchestration and it is used to model the solution architecture. The solution architecture model contains patterns, interfaces, the deployment model, and sequence diagrams. The workflow orchestration builds

---

[12] The technology to solve this problem is model transformation.

the process and workflow architecture with the sequence, flows, and pick messages. The architecture models and the workflow orchestration are both used in the coding tools. Figure 3-16 shows the tool chain we used in this redbook.
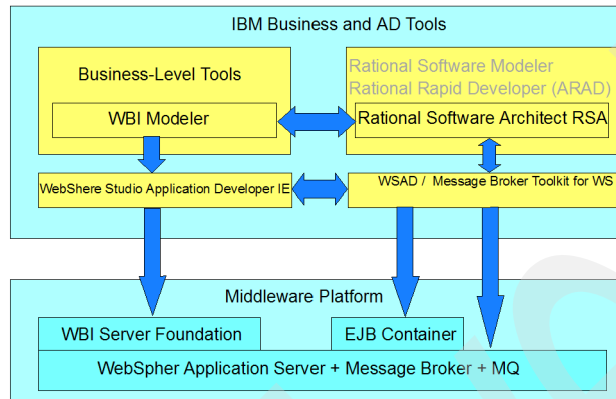
Figure 3-16   Tool Chain for this Book

The tool chain we used does not have the capability to automatically incorporate Patterns for e-business into the model. We used the tools on the Patterns for e-business Web site to select our patterns, and then drew them using Microsoft Visio. After we agreed on the patterns were agreed, then we drew the resulting deployment diagram in Rational Software Architect.

Here is a summary of the tools we used:

► The business process was developed with WBI Modeler.

► Existing systems, such as the Assessor Management System and the Document Handler were developed in WebSphere Studio Application Developer.

► ESB services, such as the distribution of availability requests to assessors were developed in WebSphere Business Integration Message Broker Workbench.

► Workflows in FDL were imported to WebSphere Business Integration Modeler from WebSphere MQ Workflow Buildtime.

► The Rational Software Architect was used to produce the system and solution architecture. Rational Software Architect has the following benefits for our project:

  – It has the capability to point to the artifacts which were developed in WebSphere Business Integration Modeler to produce the systems architecture.

– It can import existing applications from WebSphere Studio Application Developer as input for developing the new architecture.

Based on the input from the WebSphere Business Integration Modeler and interfaces from existing implementations, Rational Software Architect enables the architect to build the interfaces for all the applications, class diagrams of the various components, the sequence diagram and the deployment diagram.

The main artifacts that are exchanged with the IT specialists tools are BPEL, FDL and WSDL These interface specifications are the input for programming tools. We used WebSphere Studio Application Developer, WebSphere Studio Application Development Integration Edition, WebSphere MQ Workflow build time and WebSphere Business Integration Message Broker workbench. Which tool depends on the runtime platform the IT specialist is going to deploy to.

3.3 Summary

This chapter describes the approach we take to develop and integrate the new External Claim Assessor process with the existing Claims Handling process, and with the existing IT architecture.

Integration projects have proven even more difficult to complete to everyone's satisfaction than greenfield software development projects. A major reason for this is the difficulty in communication between the people involved representing many different aspects of the solution, all competing for attention from the very beginning of the project. The method described in this chapter and which we follow, aims to improve the communication between the different roles involved by focussing attention on what needs to be done to meet requirements represented by the business process, using a common software development platform to simplify the exchange of artifacts of the solutions, and using common modeling languages to improve the interchangeability of the artifacts. In "Chapter 4, "Business Process" on page 75" we start by looking at modeling the business requirements.

4

Business Process

In this chapter we demonstrate the steps of creating and analyzing a new business process using WebSphere Business Integration Modeler by taking on the role of business analyst.

We describe the following topics:

► Introduction to business process modelling
► Modeling the claim investigation process
► Simulate the process
► Developing the process implementation

Attention: Since writing this chapter of the Redbook, WebSphere Business Integration Modeler has shipped modification level 5.1.1.2 patch 3 and Modeler Version 6. We strongly recommend you start with at least 5.1.1.2 patch level 1 installed for compatibility with Rational Software Architect 6.0.0.1. There is a model patch shipped to migrate models created with 5.1.1.1 to 5.1.1.2. We haven't yet tested the tool chain for Modeler Version 6.

Some of the diagrams in this Redbook were captured using 5.1.1.1 and may look slightly different in 5.1.1.2

4.1 Introduction to business process management

In this section we introduce Business Process Management and WebSphere Business Integration Modeler v5 as a tool for modelling business processes. We discuss the expected users for it and the two editions.

4.1.1 Business Process Management

Business Process Management (BPM) is the concept of continuously creating, analyzing, and improving a business process (Figure 4-1).
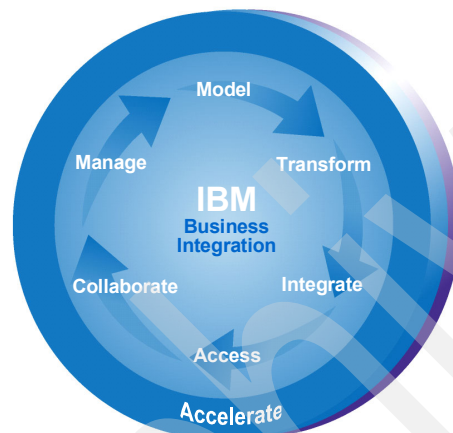
Figure 4-1   The continuous cycle of process improvement

Donald Light, *"Deriving insurance business value from business process management tools"* (see bibliography) suggests a BPM solution typically includes the following elements:

► Integrated Development Environment (IDE)
► Process Library
► Process Execution Engine
► Monitoring
► Execution Database
► Modeling and Optimization

Integrated Development Environment

The Integrated Development Environment allows business and technical users to design, simulate, document, test and deploy a claims process integrating it with existing processes, services, databases, applications and the IT infrastructure. The capabilities need to be tailored to the tasks and abilities of different kinds of users, such as business analysts, software architects and IT

specialists. The IDE needs to handle changes which affect the process and be able to refactor the changes into the business model.

Process library

The Integrated Development Environment needs to manage changes to and refinement of the process as it is designed and implemented, and as responsibility for the development of the process passes between the business analyst, the software architect, and the IT specialist. The records for the process should contain information about the requirements that have been incorporated into the process, dependencies and interactions with other processes, and auditable information such as the authors, reviewers and approvers and change activity.

Process execution engine

The process execution engine is the core BPM application. It controls and directs the flow of claims among processes and actors. It provides transactional capabilities in addition to directing the flow of claims and orchestrating calls to services, other processes, databases and people. It is able to execute claims processes and activities in a claims process in parallel as well as sequence individual steps of a claim.

Monitoring

Monitoring allows claims supervisors to view the status of claims, claim handlers, and other services used in the process. The claims supervisor should be able to view the duration of claims against expected targets and the workload on individual claim handlers. as well as switch work between claim handlers, and modify the sequence of actions on an individual claim to the extent this has been designed into the process.

Execution database

The execution database contains a record of all executing and executed processes. The events that are recorded are defined as part of the development process. The event records are queriable on the run time system, as well as being available export into other tools.

Modeling and optimization

Modeling allows the business analyst to use real execution data to calibrate the business process model and permit construction and testing of alternative processes and what-if scenarios. Optimization enables the analyst to explore the effects of removing different constraints. What is the optimal number of claim handlers for instance?

4.1.2 IBM suite of BPM tools

These elements of a BPM solution are brought together in IBM's suite of BPM tools. The redbook, *"Continuous business process management"*, SG24-6590, shows how to use the WebSphere Business Integration tools released in version 4 of the WebSphere platform to implement the continuous business process management cycle (see Figure 4-2).
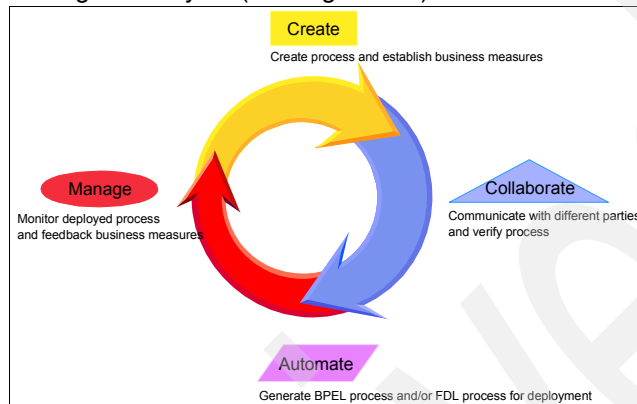
Figure 4-2   Continuous business process management cycle

WebSphere Version 4 BPM Solution

This version includes the following elements.

Create

IBM WebSphere Business Integration Workbench V4.2.4: Business Modeler is the tool to create business process which are published as Flow Definition Language (FDL) models.

Collaborate

IBM WebSphere Business Integration Workbench Server V4.2.4: Business Repository and Web Publisher are the tools to communicate the business process with other people.

Automate

WebSphere Business Integration Server V4.3 or IBM WebSphere MQ Workflow V3.5 are the runtimes for executing FDL models.

Manage

IBM WebSphere Business Integration Workbench V4.2.4: Business Monitor is the tool to monitor business processes

WebSphere Version 5 BPM Solution

Version 5 of the WebSphere platform has moved the BPM solution to open standards using Eclipse for tooling, Java 2 Enterprise Edition for the runtime and BPEL for modeling and executing business processes. We have based our solution on version 5 of the WebSphere platform.

Create

WebSphere Business Integration Modeler Advanced Edition V5.1.1.2 is the tool we used for modifying the claims process and creating the claim assessor process, and for optimizing the claims processes based on simulations.

Collaborate

WebSphere Business Integration Modeler has a report function to query business processes and create reports that can be shared with other people. We haven't used this capability in the present exercise.

Like other IBM Eclipse based tools it used the concurrent version system (CVS) for team collaboration. We used the tool in stand-alone mode, importing and exporting the artefacts with other tools. It can also be installed as a plug-in to WebSphere Studio Application Development Integration Edition.

Automate

After we had defined the claims and claim assessor processes in WebSphere Business Integration Modeler, we modeled the entire solution using Rational Solution Architect[1]. Based on this model, we defined the interfaces to the services we would be using to automate the activities in the BPEL model. We then used WebSphere Studio Application Development Integration Edition to refine the BPEL flows to execute in WebSphere Business Integration Server Foundation and we used WebSphere MQ Workflow Buildtime for developing the Flow Definition Language (FDL) processes that are deployed to WebSphere MQ Workflow. We also used WebSphere Business Integration Message Broker to route and transform the service requests and WebSphere Application Server to host some of the services.

Manage

We did not plan to design and build a monitoring solution for the claims scenario in this redbook. The IBM System House Scenario team will build the monitoring solution next year. The solution will use the Common Event Infrastructure (CEI), and monitoring tools that support the CEI.

---

[1] This is actually a version 6 product. We chose to use it because it is better integrated with WebSphere Business Integration Modeler than Rational XDE™ or Rational Rose (see Chapter 5, "System Architecture" on page 153).

For now[2], it is possible to use the CEI monitor packaged with WebSphere Business Integration Server Foundation to monitor events in the BPEL flow, and IBM WebSphere Business Integration Workbench V4.2.4: Business Monitor for monitoring WebSphere MQ Workflow.

4.1.3 Why business process modelling

Mapping an enterprise's business processes can be a daunting task. Communicating these processes and all they involve to others can be even more challenging. A business process model is a visual representation of a process and supporting information. People find working with visual process models the easiest way to understand the relationships between activities. We seem to be naturally adept at understanding the step-by-step relationship between activities. Noninformation technology personnel are comfortable working with visual process models as a way of discussing requirements and representing the way in which tasks are accomplished in an organization.

In the early stages of mapping an enterprise's business processes, flip charts or a generic presentation tool are all that are required. But if you want to document your processes, and later streamline them through automation or other initiatives, then a custom business process modelling tool offers a number of advantages compared to creating ad-hoc documentation using presentations and wordy documents. The tool should offer to:

► Capture existing business processes and guidelines in one place for easy reference and company-wide consistency.

► Analyze current processes in detail.

– Capture data about existing processes in a structured way.
– Zoom in from a high-level view of a complete solution to details about individual subprocesses.

► Answer questions about who does what, which organizations participate and which applications are used in the current process.

► Simulate capabilities to uncover your process weaknesses and highlight improvements.

► Generate process performance metrics.

► Generate an assessment of your company's return on investment regarding planned process changes.

► Provide a clearly documented contract in the form of a process model for IT professionals to implement.

---

[2] Version 6 of WebSphere Business Integration Modeler has extensive support for modeling business measures

We use business process models for many purposes, including:

- ► Documenting existing procedures
- ► Determining requirements for staff, systems and facilities
- ► Planning changes to existing processes and systems
- ► Testing and analyzing existing and proposed processes
- ► Identifying bottlenecks in your processes

4.1.4 WebSphere Business Integration Modeler

Version 5.1 is a completely new implementation of the WebSphere Business Integration Modeler. It provides a complete definition of the Enterprise from a business perspective, such as:

- ► Business artifacts (data, items, parts, etc.)
- ► Business processes, sub-processes, global task, artifact repositories
- ► Organizations
- ► Resources
- ► Timelines, locations, currencies

Compared to WebSphere Business Integration Modeler version 4.2.4, WebSphere Business Integration Modeler has new some useful new features:

- ► Eclipse-based implementation
- ► BPEL modelling and support for Web services
- ► Enhanced reporting and simulation capabilities
- ► New UML2-derived meta-model
- ► Team support

Attention: You can create the processes in this redbook using WebSphere Business Integration Modeler with no previous experience. A business analyst will probably need no more technical detail about the Modeler than provided in this book, and in the Modeler's information center. However, if you want to use the Modeler to create FDL or BPEL flows, or want to understand it in a more systematic way, then look at the redbook *"BPEL4WS Business processes with WebSphere Business Integration: Understanding, Modeling, Migrating"*, SG24-6381, particularly chapter 5.

4.1.5 Editions of WebSphere Business Integration Modeler

There are two editions for WebSphere Business Integration Modeler. They are Entry edition and Advanced edition. Table 4-1 on page 82 lists the features for these two editions.

Table 4-1   WebSphere Business Integration Modeler Entry edition and WebSphere Business Integration Modeler Advanced edition features at a glance

Feature	WebSphere Business Integration Modeler entry edition	WebSphere Business Integration Modeler advanced edition
User profiles	Basic, intermediate and advanced	Basic, intermediate and advanced
Technology modes	Operational	Operational, BPEL and FDL
Team support	Yes	Yes
Simulation	NO	Yes
Static and dynamic analysis	NO	Yes
Report generation	Yes	Yes
Query	Yes	Yes
Basis report templates (available)	Yes	Yes
Printing	Yes	Yes
Modeler project import and export	Yes	Yes
Delimited file import and export	Yes	Yes
ADF import	Yes	Yes
XSD import and export	No	Yes
UML export	No	Yes
FDL and BPEL export	No	Yes
FDL import	No	Yes

4.2 Using WebSphere Business Integration Modeler

In this section we discuss how to use WebSphere Business Integration Modeler both in the context of introducing a new or changed process to the business, and in the context of developing the process as part of the wider software solution.

4.2.1 Who uses WebSphere Business Integration Modeler?

Figure 4-3 shows typical users of WebSphere Business Integration Modeler v5.x.
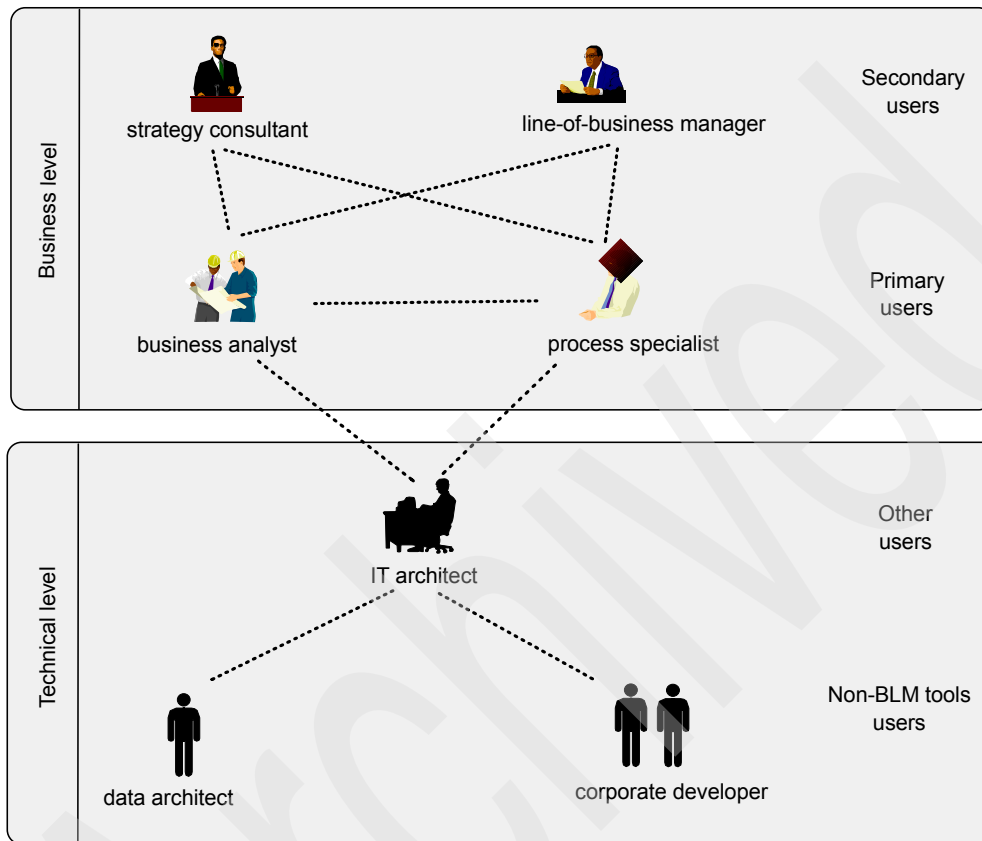
Figure 4-3   Users of WebSphere Business Integration Modeler v5.x

- ► Primary users are business analysts (BA) and process specialists, IT Process Specialists.

  Business analysts and process specialists use WebSphere Business Integration Modeler to construct business processes, identify process improvements and evaluate return-on-investment.

  IT Process Specialists, corporate developers in the diagram, use WebSphere Business Integration Modeler to refine a BPEL process before exporting it to process automation tool.

- ► Secondary business users are Strategy Consultants and Line-Of-Business Managers. They need to understand the models produced by WebSphere

Business Integration Modeler to assess and approve funding for the project. We refer to these users as the business sponsors.

► Other users of WebSphere Business Integration Modeler are IT architects. They refer to and share artifacts created in the business model, such as organizational structures, activities, and the roles of activities in the business process.

4.3 Modeling the claim investigation process

In this section, we act as the business analyst to work on the claim investigation process using the advanced edition of the WebSphere Business Integration Modeler.

In the scenario, the claim investigation process has been modelled and runs as part of the WebSphere MQ Workflow claims process in LGI. We have exported the claims process from WebSphere MQ Workflow and have a Flow Definition Language representation of the claim investigation process from which to start.

4.3.1 Start WebSphere Business Integration Modeler

If you start WebSphere Business Integration Modeler for the first time, the **QuickStart** wizard opens. Click **Cancel** to ignore it. WebSphere Business Integration Modeler starts with a 2-pane layout and Business Modeling Perspective as shown in Figure 4-4 on page 85.
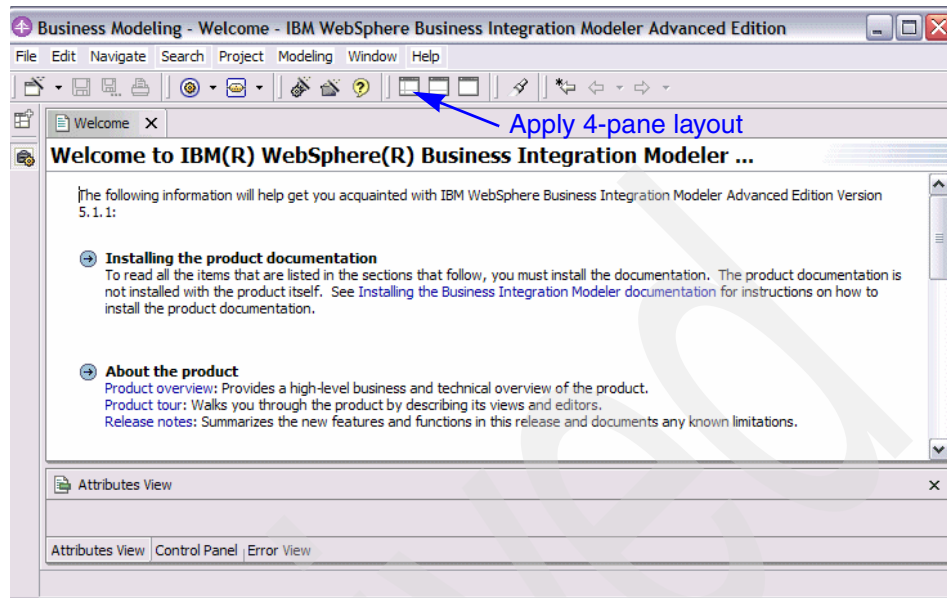
Figure 4-4   2-pane layout in WebSphere Business Integration Modeler

You can switch to 4-pane layout by clicking the **Apply 4-pane layout** icon. The following four panes are displayed:

► Process editor
► Outline view
► Project tree
► Attributes view

As business analysts, most of the time we work with the basic user profile and operational mode to concentrate on business process logic. These are the defaults. The basic user profile focuses on creating and displaying sequence flows and does not expose low-level technical details of process and data modeling. Later, we will use the more technical profiles to prepare the model for export. The operational mode is the least restrictive editing mode and most suitable for a defining a business process independent of any implementation platform.

When you switch from the operational mode to another mode, the following changes occur:

► Some options become disabled
► Some notational elements become disabled

► A previously valid model might now be invalid, because the BPEL and MQ Workflow FDL technology modes are more restrictive and have additional validation rules.

Figure 4-5 shows a 4-pane layout.
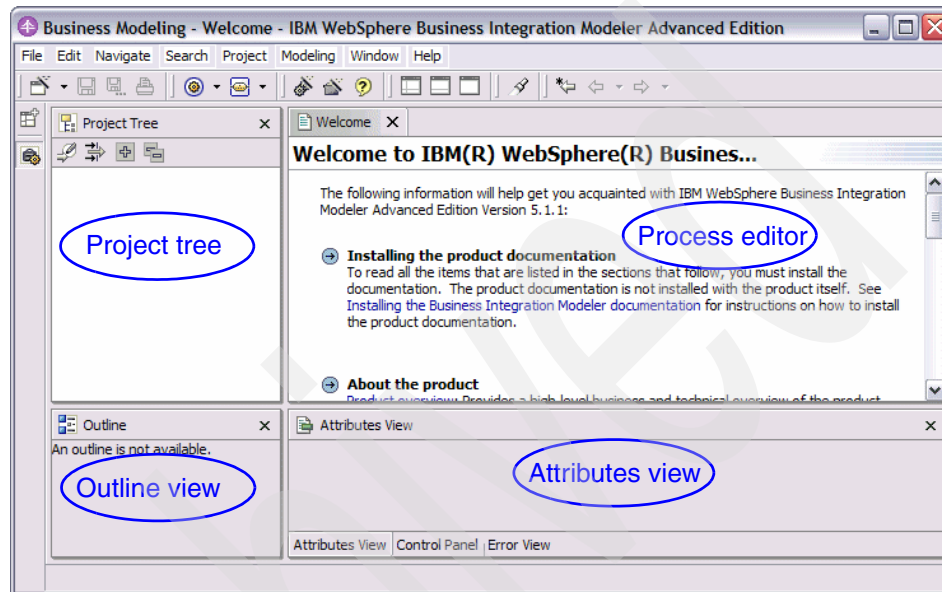
Figure 4-5   4-pane layout in WebSphere Business Integration Modeler

4.3.2 Import AS-IS process

To import an as-is process, follow these steps:

1. In WebSphere Business Integration Modeler, click **File** → **Import**, a window with title **Import** appears.

2. Choose **WebSphere Business Integration Modeler Import** → **Next** → **WebSphere MQ Workflow** → **Next**, you get WebSphere Business Integration Modeler Import Page as shown in Figure 4-6 on page 87.
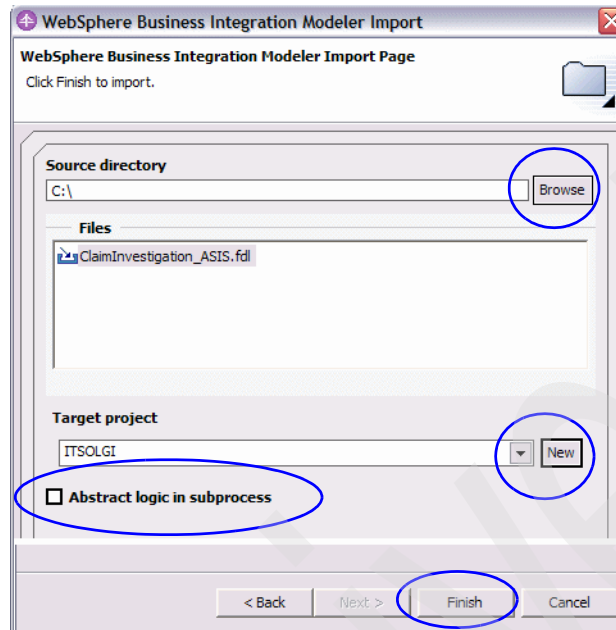
Figure 4-6   WebSphere Business Integration Modeler Import Page

3. Click **Browse** to choose the directory where the to-be imported FDL file is located. There is an example stored in
   `.\SG24-6636\Modeler\Other\ClaimInvestigation_ASIS.fdl` which you can e find in the additional materials folder for this redbook.

4. Because the ClaimInvestigation as-is process is fairly simple and contains only four activities as shown in figure Figure 2-4 on page 19, we *deselect* **Abstract logic in subprocess** to avoid WebSphere Business Integration Modeler generating subprocesses for it. However for FDL processes that have more activities and subprocesses, we recommend selecting this option to allow WebSphere Business Integration Modeler to generate the same number of nodes as the original process.

5. Click **New** to create a new project for ClaimInvestigation process to import into. Specify the project name `ITSOLGI` and deselect other options as shown in Figure 4-7 on page 88.
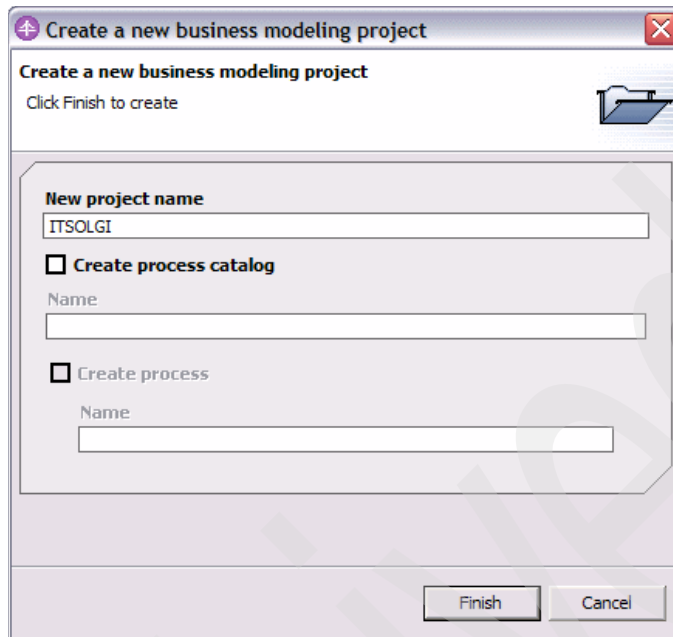
Figure 4-7   Create a new business modeling project window

After importing is finished, an information window similar to Figure 4-8 is shown. If there are any errors or warnings, click **Details** to display them.
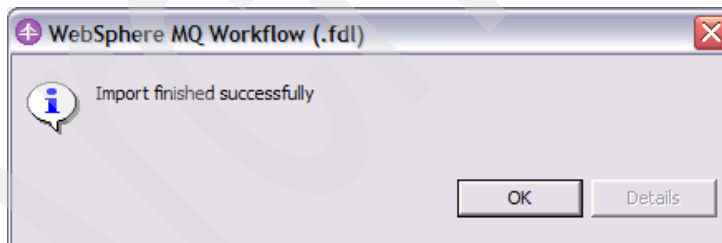
Figure 4-8   Successful import of the ClaimInvestigation FDL process

4.3.3 Analyzing the as-is process

In this section, we examine how WebSphere Business Integration Modeler represents the as-is claims investigation process imported from WebSphere MQ Workflow.

Elements mappings

By exploring the ITSOLGI project in Project Tree, we see the element mappings from WebSphere MQ Workflow to WebSphere Business Integration Modeler as shown in Figure 4-9:

► Elements in Data structures are imported as Business items with the same names.
► Process catalog and process are imported with the same name.
► Persons and Roles are imported with the same name but belong to Resources.
► Organizations are imported with the same name.
► Programs are not imported.
► Elements in Network tab are not imported.
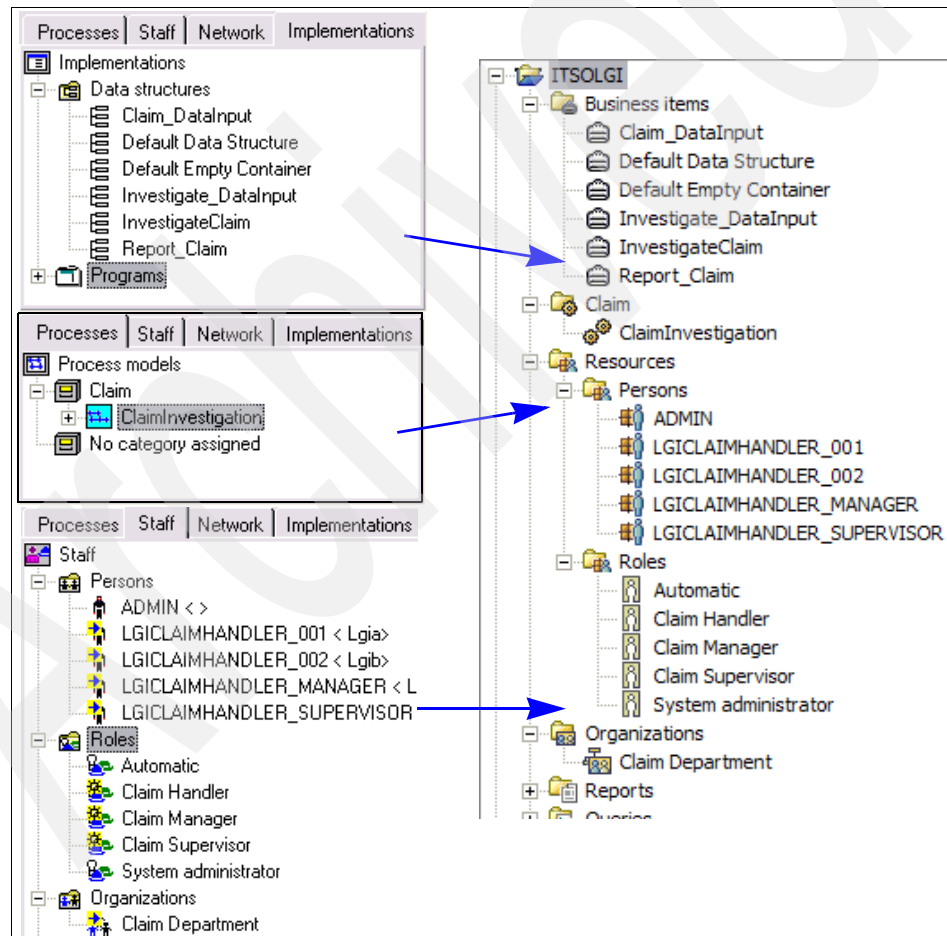
Figure 4-9   WebSphere MQ Workflow to WebSphere Business Integration Modeler

Process mappings

Examine the process mappings between WebSphere MQ Workflow and WebSphere Business Integration Modeler.
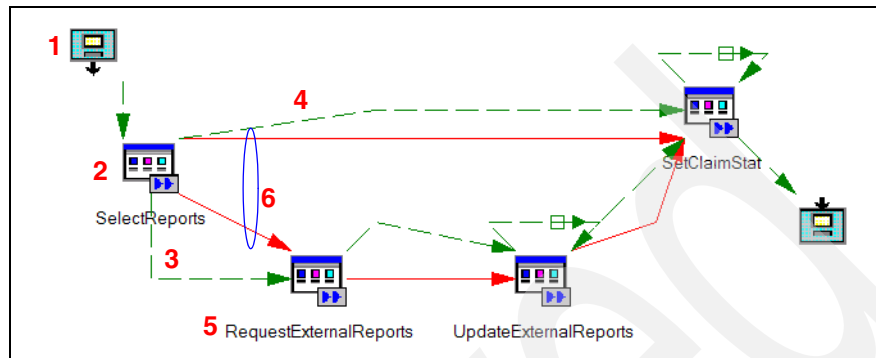
Figure 4-10   WebSphere MQ Workflow claims investigation process mapping (1)

► The source node **1** (Figure 4-10) is mapped to the input **1 (**Figure 4-11**)** of the whole process.

► The SelectReports **2** task is mapped to the local task SelectReports **2**.
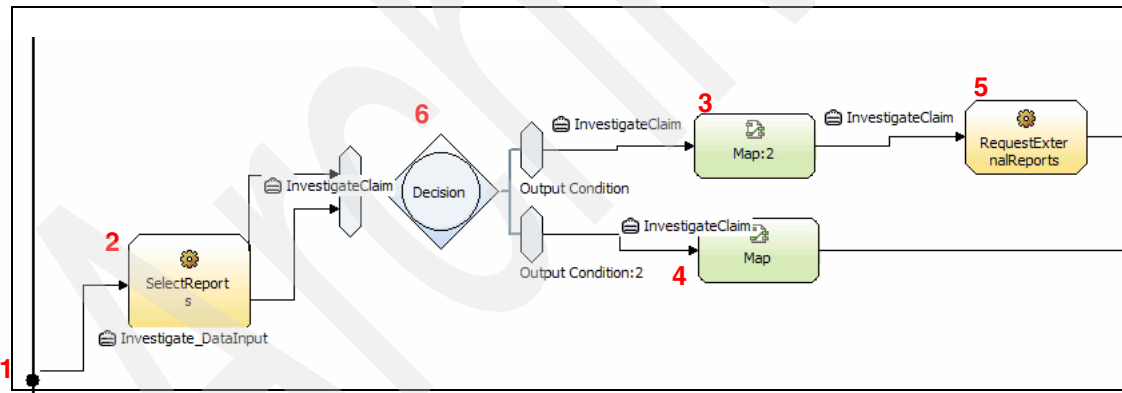
Figure 4-11   WebSphere Business Integration Modeler claims investigation process mapping (1)

► The data mappings for the two data connectors from SelectReports to RequestExternalReports and SetClaimStat are implemented with the Map elements **3** and **4** respectively by WebSphere Business Integration Modeler. Figure 4-12 on page 91 illustrates the implementation details for Map element **4.** Note that WebSphere Business Integration Modeler ignores the Default value field of a data element.

Figure 4-12   Details of data mapping

► The RequestExternalReports task **5** is mapped to the local task RequestExternalReports **5** in WebSphere Business Integration Modeler.

► The Decision element **6** is used explicitly by WebSphere Business Integration Modeler for the two Control Connectors from SelectReports in WebSphere MQ Workflow. The conditions on the two output branches correspond to the transition conditions of the two Control Connectors.

One of the transition condition mappings is shown in Example 4-1 and Example 4-2 on page 91.

Example 4-1   Mapping of transition conditions 1

```
(Investigate_DataInput.Claim_DataInput.Status="V") AND (Report_Claim.AssReqDate
<> "null")
```

Example 4-1 is mapped to Example 4-2:

Example 4-2   Mapping of transition conditions 2

```
'RootProcessModel.Claim.ClaimInvestigation_ASIS.ClaimInvestigation_ASIS.
Decision.InputObjectPin.Investigate_DataInput.Claim_DataInput.Status'
```

```
is equal to "V" AND
'RootProcessModel.Claim.ClaimInvestigation_ASIS.ClaimInvestigation_ASIS.
Decision.InputObjectPin.Report_Claim.AssReqDate' is not equal to "null"
```

In Figure 4-13 and Figure 4-14:
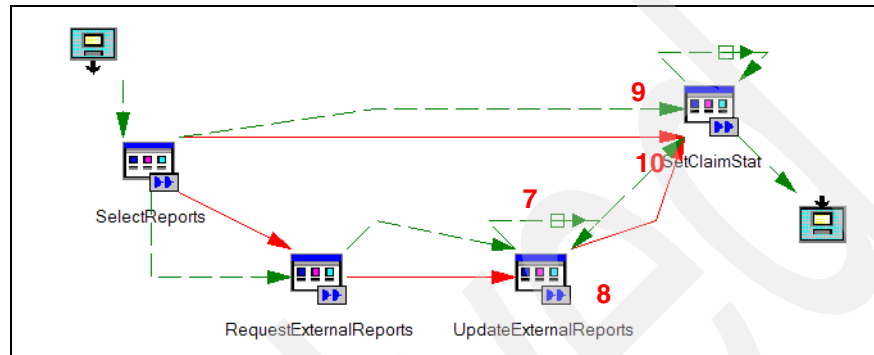
Figure 4-13   WebSphere MQ Workflow claims investigation process mapping (2)

► A Fork element is used to split the input prior to UpdateExternalReports in WebSphere Business Integration Modeler.

► The connection **7** corresponds to the Data Default Connector **7** in WebSphere MQ Workflow.

► There is no business item associated with the output of UpdateExternalReports **8** in WebSphere Business Integration Modeler. This is because UpdateExternalReports **8** is an asynchronous User-Defined Program Execution Server (UPES) implementation.

► A Merge element is generated in WebSphere Business Integration Modeler for combining the inputs **7**, **9**, **10** to the next.
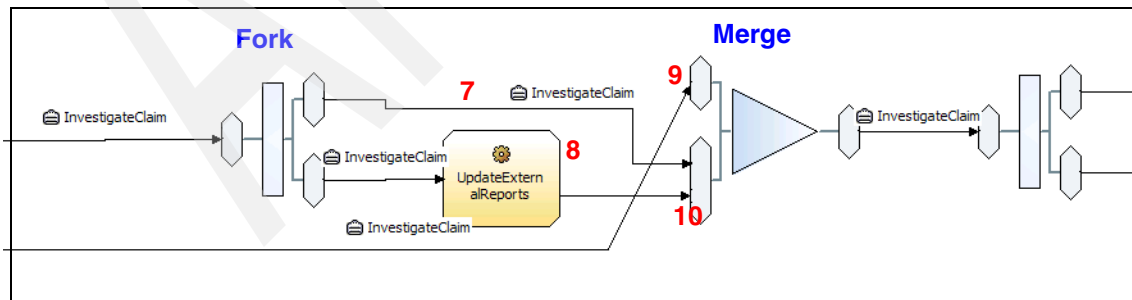
Figure 4-14   WebSphere Business Integration Modeler claims investigation process mapping (2)

Figure 4-15 and Figure 4-16 show the last part of the process mapping:
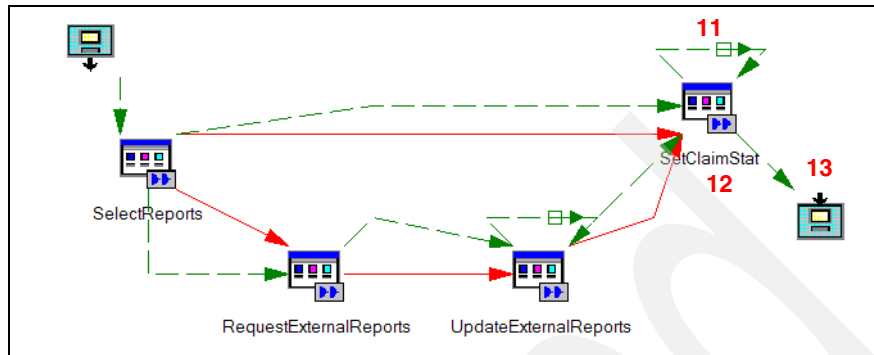
Figure 4-15   WebSphere MQ Workflow claims investigation process mapping (3)

► The connection **11** in WebSphere Business Integration Modeler is equivalent to the Data Default Connector **11** for SetClaimStat in WebSphere MQ Workflow.

► SetClaimStat **12** is mapped to local task SetClaimStat **12**. There is no business item associated with the output of SetClaimStat **12** in WebSphere Business Integration Modeler because of the asynchronous UPES implementation in WebSphere MQ Workflow.

► Sink Node **13** in WebSphere MQ Workflow is mapped to the output labeled **13** in WebSphere Business Integration Modeler.

► A Stop element **14** is added automatically by WebSphere Business Integration Modeler. A stop node is required in order for any simulations to work correctly.
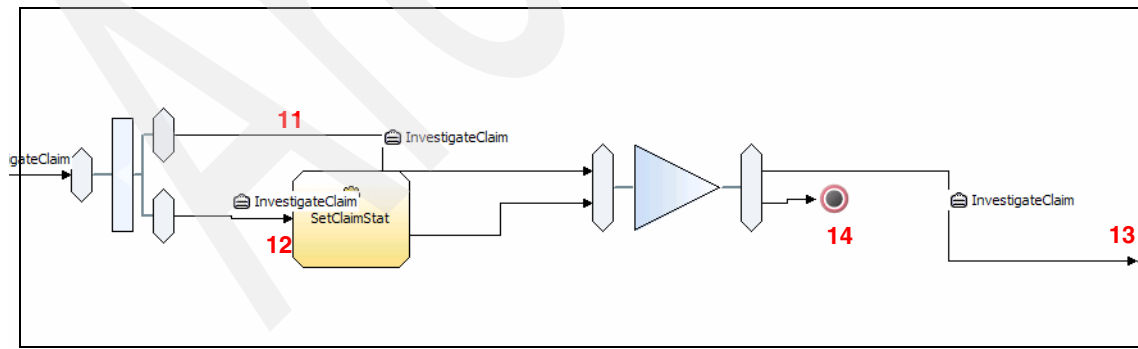
Figure 4-16   WebSphere Business Integration Modeler claims investigation process mapping (3)

► As illustrated in Figure 2-6 on page 21 the RequestExternalReports task is a manual task. To confirm this, open ClaimInvestigation in Process Editor and select **RequestExternalReports**. In the Attributes View, click the **Resources** tab.

Figure 4-17 shows the resources mapping from WebSphere MQ Workflow (the two diagrams in the upper part) to WebSphere Business Integration Modeler (the one diagram in the lower part). We can see that RequestExternalReports requires staff resources who have the role of Claim handler and are in the **Claim Department** organization. Values in other fields are generated automatically by the importing process of WebSphere Business Integration Modeler. We can ignore the value in the **Time Required** field for now.

Figure 4-17   Resource requirements: RequestExternalReports in as-is ClaimInvestigation

4.3.4 Create the to-be process

The outline of the to-be process was designed in a workshop using Microsoft Visio. It comprises a small modification to the ClaimInvestigation process and a new subprocess called RequestExternalReports that replaces the manual task of RequestExternalReports in the ClaimInvestigation process.

In this section, we go step-by-step through the procedure of creating the new RequestExternalReports subprocess and modifying the existing as-is claim investigation process by working with the workspace we have just created by importing the claim investigation process from WebSphere MQ Workflow. But first, we look at the complete RequestExternalReports subprocess as we produced it in Visio before going through the step-by-step procedure to refine it in WebSphere Business Integration Modeler.

To-be process logic
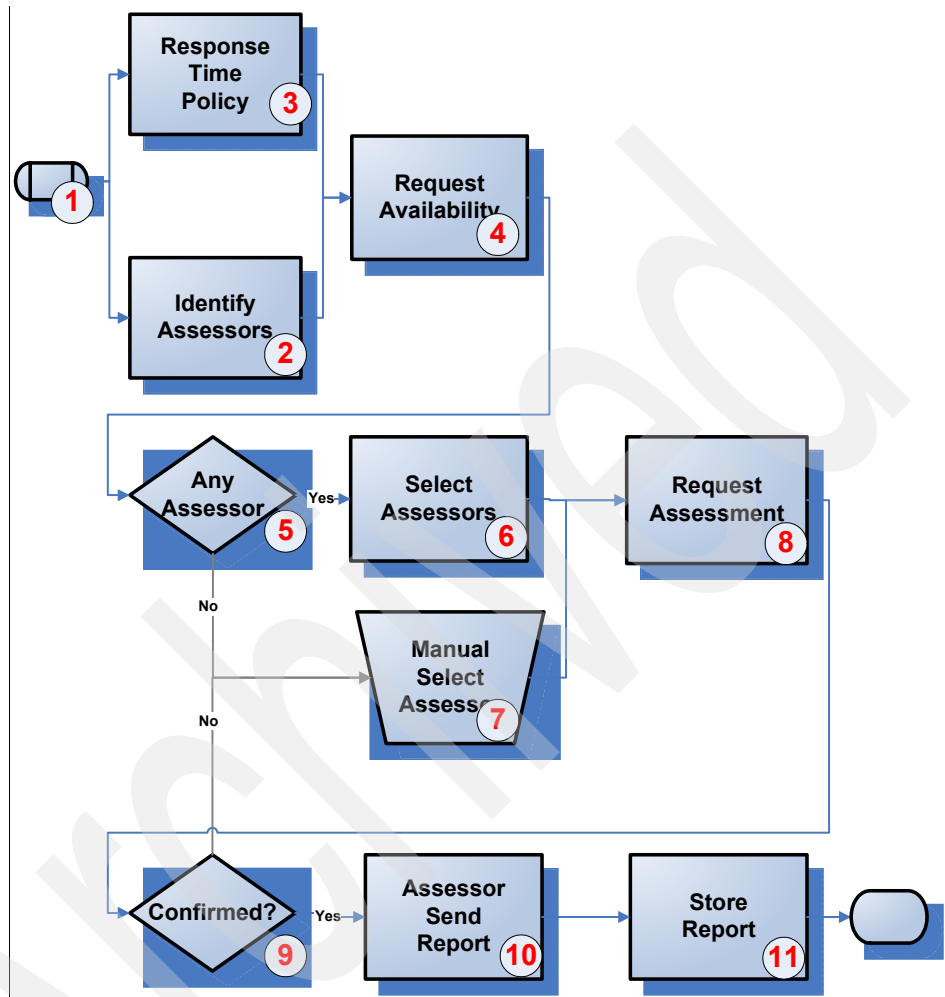
Figure 4-18   RequestExternalReports process

► The RequestExternalReports process is initiated from ClaimInvestigation and starts by forking into two parallel paths.

► On one path, the list of assessors who are eligible to assess the claim by reason of their geography and familiarity with the vehicle make is constructed.

► On the other path the customer's insurance policy is retrieved and depending on what has been agreed in the policy the expected time for the assessment is recorded.

- When these two activities are complete, all the eligible assessors are sent the claim details and the date by when an assessment is required. They are asked to respond within a fixed time whether they want to do the assessment.

- The process then waits for responses, discarding any responses that fall outside the fixed response time.

- From the list of assessors who want to perform the assessment, a single assessor is selected according to a number of criteria including cost, quality of assessments, reliability and so on.

- In the event there are no assessors able to take on the assessment, control passes to a manual task. The claim handler needs to manually select an assessor, perhaps by widening the area, or increasing the time before an assessment is due.

- The single request to perform an assessment is sent.

- An acknowledgement awaited confirming the assessor will perform the assessment.

- The assessor submits the assessment report and

- The report is filed in the document management system before control is returned to the ClaimInvestigation process for the claims handler to judge the claim.

Version control

WebSphere Business Integration Modeler uses Concurrent versions system (CVS) for version control. For more information about CVS support, refer to the **Team Support** part of WebSphere Business Integration Modeler information center:

> http://publib.boulder.ibm.com/infocenter/wbihelp/index.jsp).

We do not demonstrate CVS implementation for WebSphere Business Integration Modeler in this book. Instead we separate different versions of the process with different names. **ClaimInvestigation_ASIS** and **ClaimInvestigation_TOBE** and catalogs of **ASIS** and **TOBE**. The Project Tree structure is shown in Figure 4-19 on page 98. The steps are:

1. Expand the Project Tree. Right-click **ClaimInvestigation process** → **Rename** → specify the name with **ClaimInvestigation_ASIS**.

2. Right-click **Claim** process catalog → **New** → **Process Catalog** → specify the name with **ASIS**. This creates a catalog of ASIS.

3. Right-click **ClaimInvestigation_ASIS** → **Copy** → right-click **ASIS** process catalog → **Paste**. This copies ClaimInvestigation_ASIS to the catalog of ASIS.

4. Repeat step 2 to create another catalog of **TOBE** for business analyst.

5. Repeat step 3 to copy ClaimInvestigation_ASIS to **TOBE** and rename it to ClaimInvestigation_TOBE.
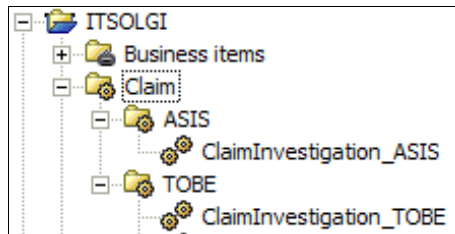
Figure 4-19   Renamed ClaimInvestigation processes

4.3.5 Build a new process

Because there are several tasks required to imitate the functionality of the manual task of RequestExternalReports, we create a new process called RequestExternalReports. There are two reasons for it:

▶ Keep the ClaimInvestigation process diagram as simple as before.
▶ The new process is reusable in other processes.

We have two ways of building the RequestExternalReports, either from scratch or by importing the Visio presentation created in the workshop. We have provided you with the Visio presentation in the additional materials supplied with this redbook. See Appendix A, "Additional material" on page 501 for further information about obtaining this material.

First, we start with how to create the basic process from scratch, if you want to import the one you made in Visio, see "Option 2 Import the process from Visio" on page 104.

Option one: Create the process from scratch

To create the process from the beginning, follow these steps.

1. In the Project Tree, right-click **TOBE** process catalog → **New** → **Process** → specify the name as **RequestExternalReports** and any description for it → **Finish**.

   RequestExternalReports is opened automatically in Process Editor. Ensure that you are in **Basic** profile. Delete **Start** Node because the process is started from the parent process.

The first part of the process

The first steps in the process are executed in parallel. On one side, the customer's policy is retrieved and, depending on the type of policy, the expected

response time is noted. On the other side, the list of assessors is sifted to extract those who are potentially able to handle this claim based upon the location and type of the vehicle.

1. Choose appropriate elements from the palette illustrated in Figure 4-20 and follow the instructions to assist in wiring them together.
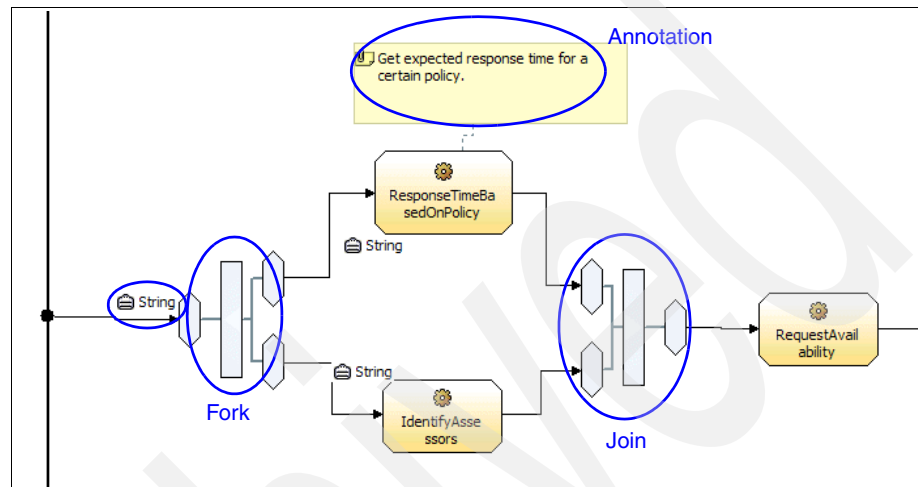
Figure 4-20   RequestExternalReports process part 1of 3

2. We use a **Fork** element to connect the input of the process to two tasks of ResponsetimeBasedOnPolicy and IdentifyAssessors.

Tip: Click the small arrowhead in the top left corner of the join/fork/merge icon in the process editor palette to toggle the element.

3. Information about each task can be added to the **Description** field shown in Figure 4-21. We also find it useful to put descriptive information into **Annotation** and link it to the task.

Figure 4-21   Description field for a task

4. We use a **Join** element to combine the two outputs from
    ResponsetimeBasedOnPolicy and IdentifyAssessors. Features of Join
    include:

    – A Join recombines and synchronizes parallel processing paths.

    – A Join waits until all the required inputs have received by each of its
       incoming branches and then sends them out as output at the same time.

    – If you want the data items consolidated into one input, you must add a
       specific task to do that.

5. After you have populated the canvas with the six elements, select
    Connections → the **outer frame** of the process. Click the **input data
    connection** of the **fork** to identify the fork as first element of the process. You
    are prompted with a pop-up box as in Figure 4-22. Select **Input** as the
    F**ork:Target dataconnection** → **OK**.
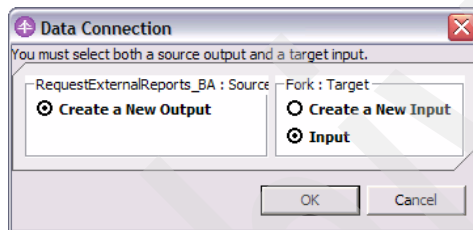
Figure 4-22   Setting the process input

6. Connecting the frame to the input connection of the fork now shows the String
    icon next to the connection as in Figure 4-23.
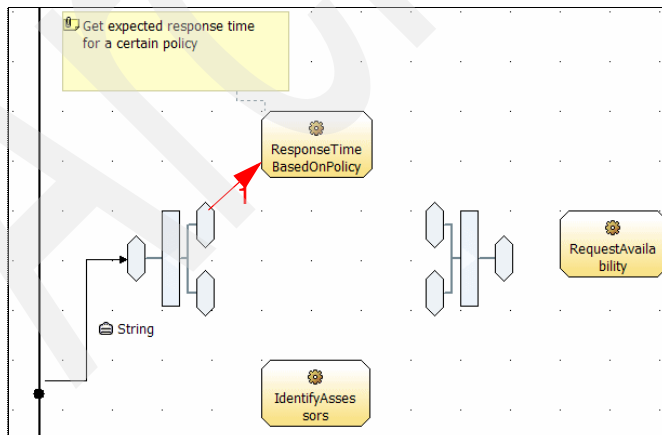
Figure 4-23   Connecting the RequestExternalReports process to its first element

7. Now start connecting the elements. Start with connection 1 in Figure 4-23.

You are presented with a pop-up box as in Figure 4-24. Select
Output:String → **OK**
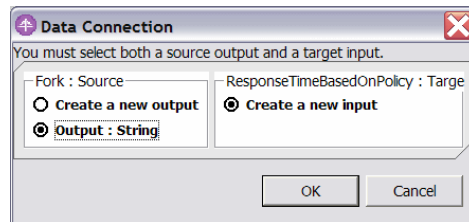
Figure 4-24   Selecting the data connection for output from Fork:Source

8. Continue to connect the elements. You might be prompted again for a choice of targets. Always select the existing target data connection rather than creating a new one.

> **Tip:** Periodically save your process. An asterisk next to its name in its tab handle indicates it is not saved. Saving your process will check for some errors. You can also run a static analysis of the process to check for other errors. See Figure 4-25.
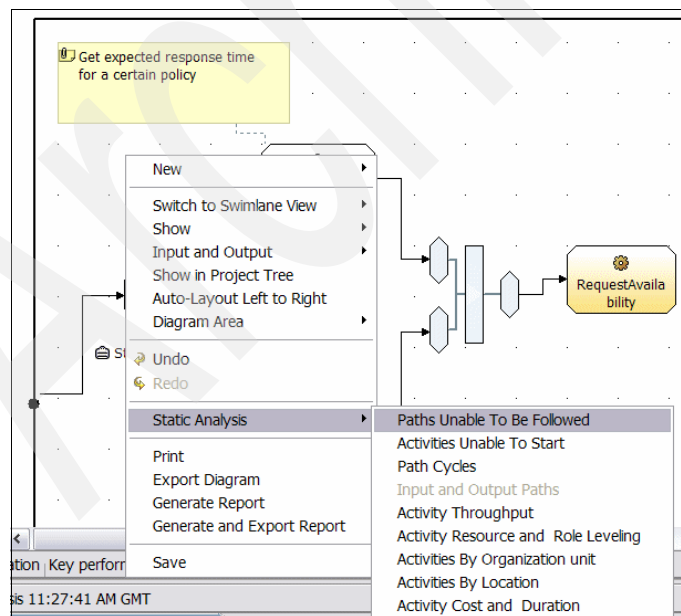
Figure 4-25   Checking your process for errors

The second part of the process

In the second part of the process, you decide to either.

► Manually select the assessor who will handle the assessment
► Have the process automatically select an assessor

In our case, the decision to manually select an assessor is only made if no assessors have been identified. Following the choice, control passes to an automated process to request the assessment.

The second part of the process is shown in Figure 4-26 on page 102, with WebSphere Business Integration Modeler switching to **Intermediate** mode to allow more details such as specifying the decision modes.

> **Tip:** We recommend you build the initial connections in basic mode and then switch to intermediate node to complete specifying the input criteria. In basic mode, the data connection terminals are created automatically, while in intermediate mode you need to add input and output terminals using the attribute panel.
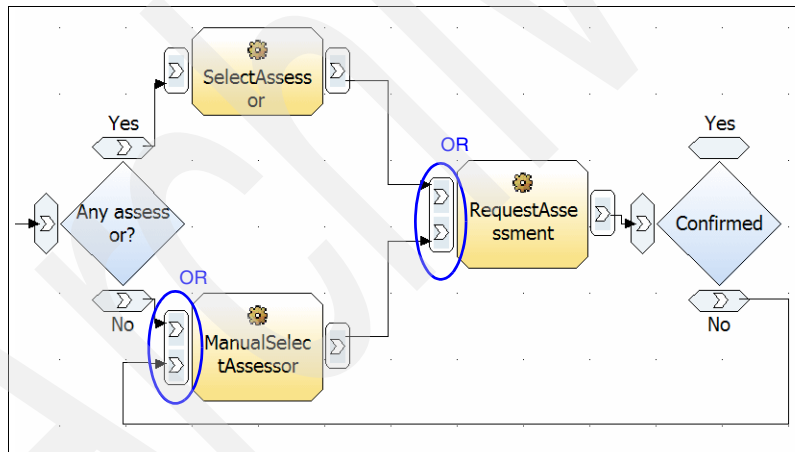
Figure 4-26   RequestExternalReports process part II of III

1. We use a simple **Decision** with the name of **Any assessor?** to direct the flow to either **SelectAssessor** or **ManualSelectAssesor** after the previous task of **AssessorReceiveAndResponse**. If there is at least one assessor in the list, process flows to **SelectAssessor**, otherwise to **ManualSelectAssessor**.

2. We also add the **Decision** with the name **Confirmed** to return the flow to the **ManualSelectAssessor** activity if the selected assessor fails to acknowledge receiving the assessment request.

3. We specify **OR** logic for the two inputs of **RequestAssessment** and **ManualSelectAssessor** respectively because both of tasks can continue as long as one of the inputs is received.

   a. To set **Input logic,** choose **Modeling** → **User Profile** → **Intermediate** from the action bar. Select **ManualSelectAssessor** → **Attributes View** → **Input logic** → **Input criteria** → **Add** → **Input** (from **available inputs**) and move it to **selected inputs** → **OK**

   b. Repeat to the procedure to add **Input:2**. The Input logic tab of the attributes panel should now look similar to Figure 4-27.
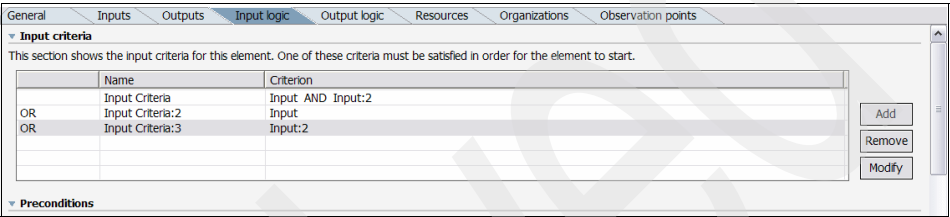
Figure 4-27   Input logic panel for ManualSelectAssessor

   c. Remove the first criteria and tidy up the panel until it looks similar to Figure 4-28.
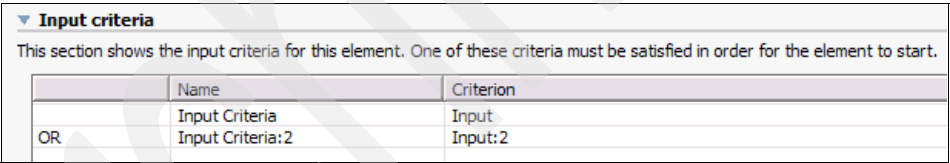
Figure 4-28   Input criteria for ManualSelectAssessor task

Tip: A **Merge** element can implement the same function.

The last part of the process is shown in Figure 4-29 in basic mode. We also connect **StoreReport** to the process output and Stop node.
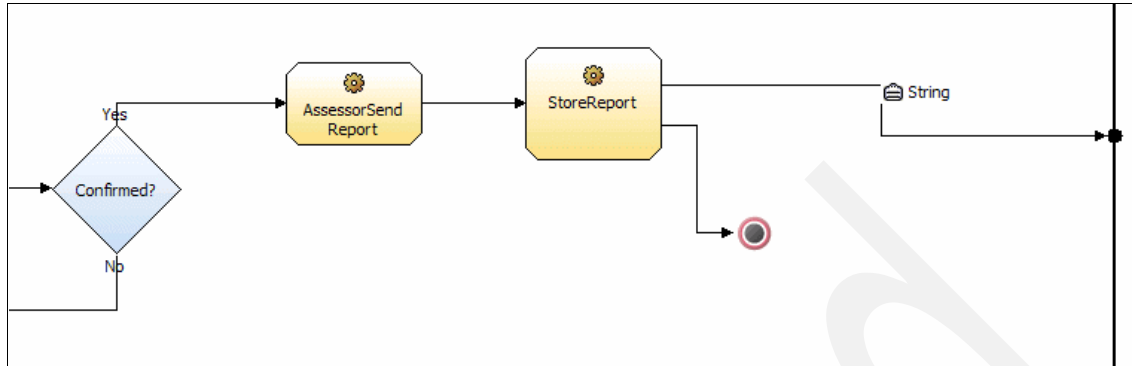
Figure 4-29   RequestExternalReports process part III of III

> **Tip:** At this point, we can run a simple simulation to validate the process logic.
> Some common errors can be picked up if the simulation stops prematurely,
> such as, Merge might be missed, that can cause the process to stop. To
> simulate, right-click the **RequestExternalReports** process in project tree and
> select **Simulate.**

Option 2 Import the process from Visio

To import a Visio diagram, you need to have saved the diagram from Visio in
XML, a .vdx file.

1. Click **File → Import → WebSphere Business Integration Modeler
   Import → Next** and select Microsoft Visio (vdx) from the **Types → Next** and
   select **.\SG24-6636\Modeler\Other\RequestExternalReports.vdx** from the
   additional material supplied with this redbook.

2. Click **Next** and select **Page-1**. Click **Add.**

   Your import wizard should look similar to Figure 4-31 on page 105. When
   creating the Visio chart, we selected chart items from the Visio Business
   Process diagrams. Most of these are already mapped to WebSphere
   Business Integration Modeler shapes. You can use the import wizard to
   associate any unmapped shapes.

3. Click **→ Next → Finish** to complete the import. Ignore the warning message
   and look for the imported process in the **Processes** folder. The result is
   illustrated in Figure 4-30 on page 105.
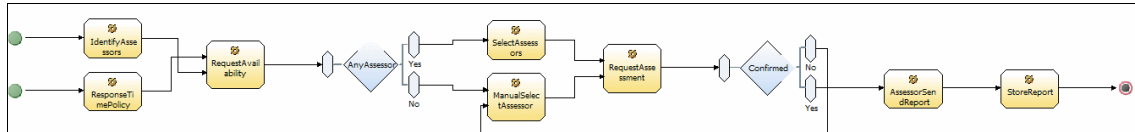
Figure 4-30   RequestExternalReports process imported from Visio

Note that the decision points are multiple choices rather than binary decisions.
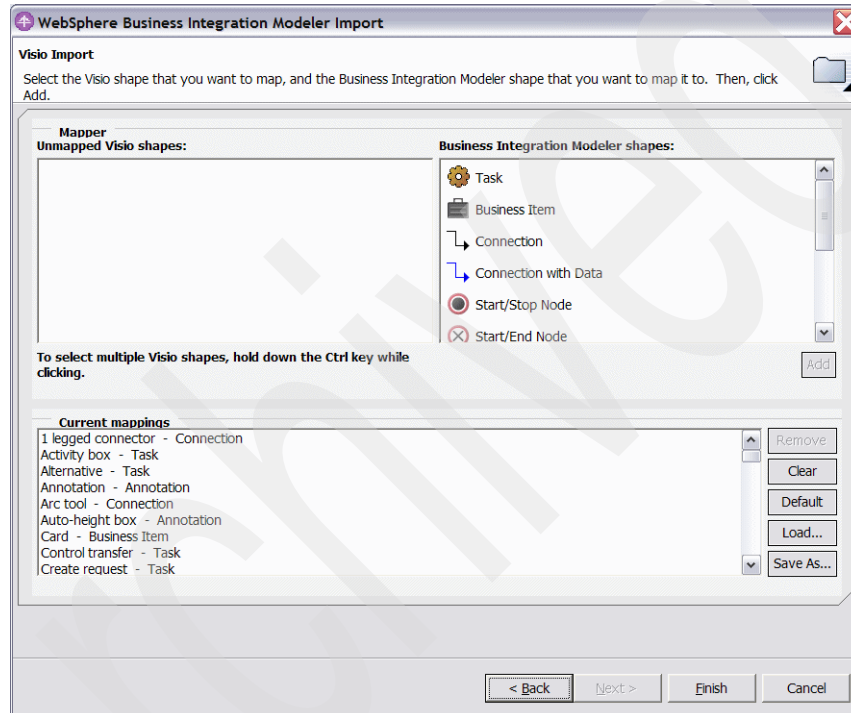
Figure 4-31   Mapping Visio shapes to WebSphere Business Integration Modeler

4. Move the flow into the TOBE process catalogue.

The flow will need some modification before it is refined. This is left as an exercise for you. There turn out to be very few modifications to make:

1. Delete the two start nodes, and the two connections into request availability

2. Drop a fork and join onto the canvas

3. Wire up the RequestExternalReports process to the input data connector of the fork, and the activity StoreReport.

4. Wire up the fork and the join.

5. You may want to switch the multiple choice decisions for binary decisions to make the flow clearer

Check the input logic of ManualSelectAssessor and RequestAssessment. They should both be set to OR the inputs.

> **Tip:** Backup your work at this time. The undo function is limited, so you should checkpoint your work regularly. There are many ways to do this. For example, you can save your work as a WebSphere Business Integration Modeler project. If you need to recover, you can start with a fresh workspace and reload the project.

Modify ClaimInvestigation to call RequestExternalAssessor

To perform this operation, follow these steps:

1. Open **ClaimInvestigation_TOBE** in the project navigator, and delete the **RequestExternalReports local task.**

2. Drag the **RequestExternalAssessor** process from the project navigator onto to **ClaimInvestigation_TOBE process**

3. Add a new map to the ClaimInvestigation_TOBE process.

   a. Right-click the process → **New** → **Map.**

   b. Open the Attributes view of the map and select the **Inputs** tab. Double-click **Associated data** → **Complex type** → **InvestigateClaim** → **OK.**

   c. Select the **Outputs** tab. Double-click **Associated data** → **Basic type** → **String** → **OK.**

   d. Wire up the elements as in Figure 4-32.

Figure 4-32   Adding RequestExternalReports subprocess

Define roles and organization units

To accomplish the newly defined tasks, additional roles and organization units are defined. These are the ones embraced with blue squares shown in Figure 4-33 on page 108. Note that *roles* include automated services as well as people. This will become important when we share the business process with the Rational Software Architect or Rational Software Modeler.

From where do these roles and organizations come? How do we know what automated services are needed at this stage in the process definition?

The business analyst has been working with the solution architect in the workshops. As discussed in 3.2.2, "Responsibilities and contract-based development" on page 52, the solution architecture is being developed in parallel with the business process so that decisions can be made about which activities to automate. At present, the tooling is not sufficiently integrated to enable very interactive development of the business process and the solution architecture together. In the workshops, we used traditional presentation tools to construct the initial business process and solution architecture proposal. This was sufficient to know what processes could be automated in the claim assessor solution and what roles needed to be added to the process model. See the diagram in "Figure 3-16 on page 73."

We explain the new roles briefly below. Add these roles to your WebSphere Business Integration Modeler workspace.

► *Claim handler* handles day to day claims for LGI. Modify the standard hourly rate to $15 per hour in the Claim handler role.

► *Assessor* is the role who does the claim assessment and sends a report back. The cost is $20 per hour.

► *Assessor Management, Business Rules Engine*, and *Document Handler* roles represent systems which automatically handle work. We fix the cost as $10000 per year.

► *Claim handler Desktop* role is the system that Claim handler uses to do work. The cost is $10 000 per year.

> **Tip:** A role enhances a resource, adding a set of functions that the resource must meet or perform. One resource can play more than one role.

Create this new organization unit:

► **External** Assessors represents the organization unit for claim assessors.

> **Tip:** It is better to assign costs to roles than to the resources that are allocated to roles, unless you want to cost a resource specifically. There are different cost capabilities to consider too: We can define a cost per unit of time to a bulk resource like a computer when it is playing a role, but not to the resource definition by itself.
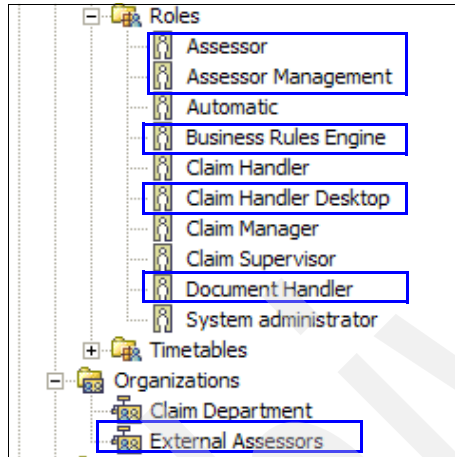
Figure 4-33   Newly defined roles and organization unit

Define resources for roles

To define resources for each role, follow these steps.

1. Start by adding two new resource catalogues:

   a. Select **Resources** in the Project Tree → **New** → **Resource Catalog**. Name it `Machines` and click **Finish.**

   b. Repeat the procedure to create `Timetables.`

2. Create a new resource definition for the claim assessors called External Resources.

   – Select **Resources** in the Project Tree → **New** → **Resource Definition**. Name it `ExternalResources.` Make it an **individual** rather than a bulk resource and click **Finish.**

3. Now create two computers:

   a. **Select Machines** → **New** → **Resource**. Select **Machine** as the associated resource definition and call the resource `Computer 001.` See Figure 4-34 on page 109.
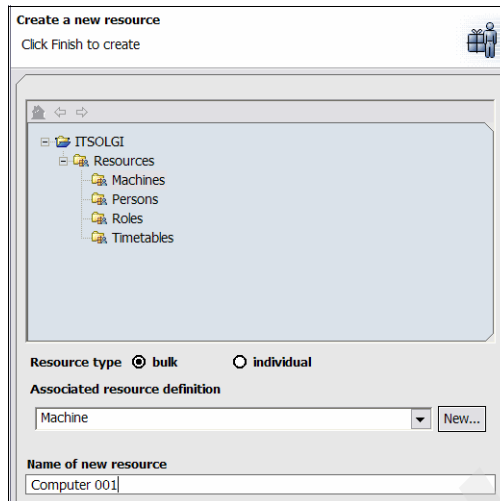
Figure 4-34   Creating a new computer resource - step 1

    b. Add the roles this computer is going to play by clicking the **Qualifications** tab and adding the roles in Figure 4-35.
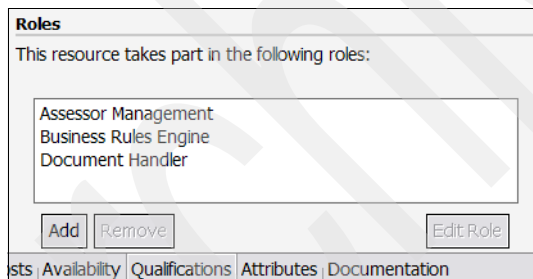
Figure 4-35   Roles of Computer 001

4. The quickest way to create Computer 002 is to copy and paste Computer 001 and then modify the roles as shown in Figure 4-36.
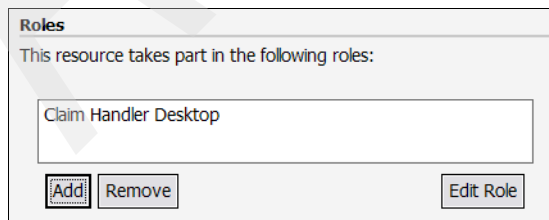
Figure 4-36   Role of Computer 002

5. Add the Claim handlers. Here, we show how to create the first Claim handler.

   a. Select **Project Tree** → **Persons** → **New** → **Resource** → **individual** → **Staff** and name the resource `Claim handler 001` → **Finish** (Figure 4-37).
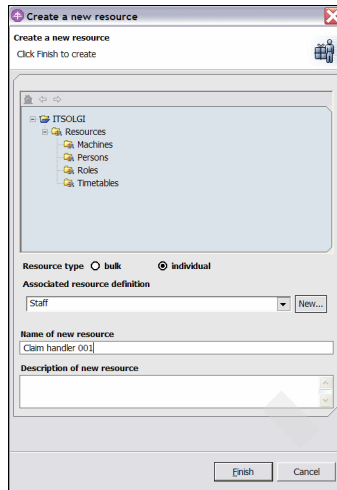
Figure 4-37   Create a new Claim handler

   b. Select the **Qualifications** tab → **Add** → **\Resources\Roles\Claim handler** → **OK.** (Figure 4-38).
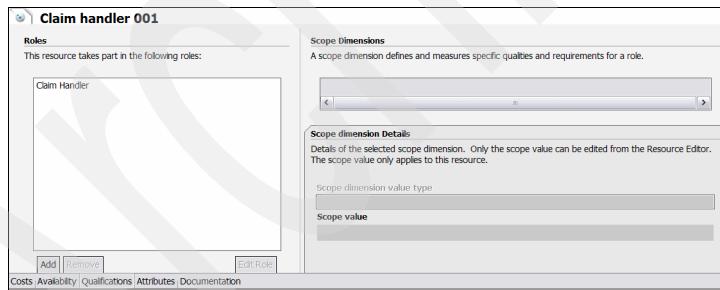
Figure 4-38   Set the Claim handler's role to Claim handler

6. Use the copy and paste on the pop-up menu to create 19 more handlers. You need to reselect **copy** before pasting each time if you are using an earlier version. This is unnecessary and is fixed in WebSphere Business Integration Modeler 5.1.1.

7. We assume that there are plenty of Assessors and they belong to external resources.

a. Select **Project Tree** → **Persons** → **New** → **Resource** → **individual** → **External Resources** and name the resource `Assessor` → **Finish** (Figure 4-37 on page 110).

b. We do not allocate a cost, but set the **Role** to **\Resources\Roles\Assessor.**

> **Important:** If a role does not have any associated individual or bulk resource and the role is assigned to a task, simulation will fail.

Define timetables

The timetable we want is constructed by combining individual timetables specifying the hours of work, lunch breaks and weekends. First we construct the three component timetables:

1. Define a timetable named `NormalWorkTime` as shown in Figure 4-39.

   a. **Project Tree** → **Timetables** → **New** → **Timetable** and name the table `NormalWorkTime` → **Finish** (Figure 4-39)

   b. Open the timetable and select **Time interval** → **Remove** → **Add** and type `Working hours` → **OK**. Set `9:00 AM` as the start time and the **duration** to `8 hours`. The default today's date is fine.

*Figure 4-39   NormalWorkTime time table, **Recurring time intervals** tab contents*

2. Similarly, define another two timetables named `LunchTime` as shown in Figure 4-40 on page 112 and `Weekend` shown in Figure 4-41 on page 112.

*Figure 4-40   LunchTime time table, **Recurring time intervals** tab contents*

*Figure 4-41   Weekend time table, **Recurring time intervals** tab contents*

3. Make sure all the timetables are saved and return to **NormalWorkTime**. Select the **Exemption** periods tab and add the Lunchtime and Weekend timetables as exemption periods.

   Therefore, the normal work time is Monday to Friday, 9:00am to 5:00pm with a half-hour lunch time. The final look of the NormalWorkTime is shown in Figure 4-42 on page 113, with working durations in blue color and nonworking durations in red color.

> **Tip:** Hover the mouse pointer in the Attributes view window and press the mouse buttons to zoom in or out on the timetable. Point at periods on the time table to inspect the name of the interval and its start and end times. Notice that by changing the name of the time interval from the default Time interval to Working hours, Lunch, and Weekend we can see more clearly how the overall timetable is constructed in this view.
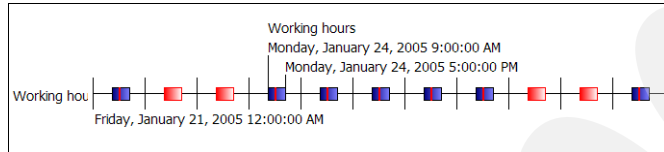
Figure 4-42   Attributes view of NormalWorkTime time table

We assign NormalWorkTime to the Claim handler and Assessor role. The timetable will not be displayed for the individual Claim handlers or assessors.

Assign resources to tasks

After the roles are defined, we assign them to tasks in the RequestExternalReports process.

1. Select the **ResponseTimeBasedOnPolicy** task in Process Editor by clicking it in the **Attributes** view, click the **Resources** tab → expand **Role requirements** if it is not expanded yet → **Add** button → specify its values from the data in Table 4-2. An example is shown in Figure 4-43 on page 114.

2. Modify the **Name** of the role requirement to something more meaningful. The name will be useful if the solution architect uses activity diagrams in Rational Software Architect. Activities are grouped into columns using the names of their role requirements. As a starting point, use the same name for the role requirement entry as the name of the role.

> **Tip:** You can drag resources from the project tree and drop them onto a task. You will need to open each task and check that each role requirement is complete.
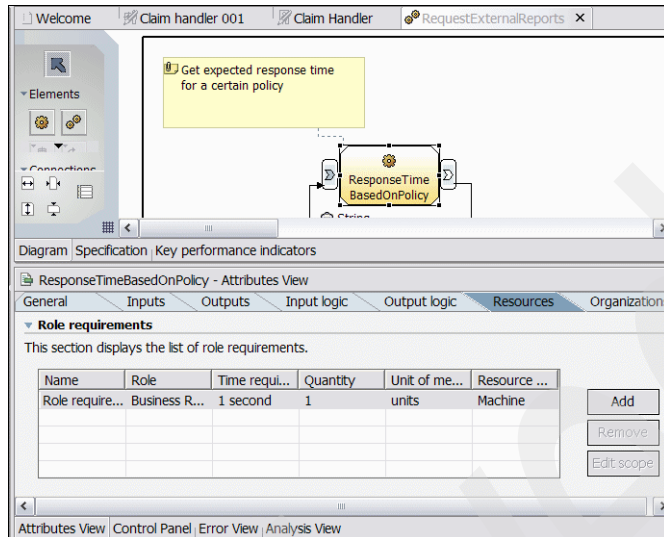
Figure 4-43   Specifying the role resource for the ResponseTimeBasedonPolicy task

3. Also define the organization unit for the tasks using the data in Table 4-2 on page 115. An example is show in Figure 4-44.

*Figure 4-44   An example of a task's **Resources** and **Organizations***

Tip: The field of **Time required** for **Resources** is used to calculate the cost of certain resource in simulation. It is different from **Time required to finish task,** which you can define for a task while doing simulation.

Table 4-2   Settings for tasks' resources and organization units

Task	Roles	Time required	Quantify	Resource Definition	Organization
ResponseTimeBasedOnPolicy	BusinessRules Engine	1 second	1	Machine	Claim Department
IdentifyAssesors	AssessorManagement	1 second	1	Machine	Claim Department
RequestAvailability	Assessor	2 minutes	1	External Resource	External Assessors
SelectAssessor	Business Rules Engine	1 second	1	Machine	Claim Department
ManualSelectAssessor	Claim handler Claim handler Desktop	2 minutes 2 minutes	1 1	Staff Machine	Claim Department
RequestAssessment	Assessor	1 second	1	External Resource	External Assessors
AssessorSendReport	Assessor	1 hour	1	External Resource	External Assessors
StoreReport	Document Handler	1 second	1	Machine	Claim Department

You can also assign individual and bulk resources to tasks. All the resources that are assigned will be allocated when the task is run. For example, we assign the role of Claim handler to a task and the quantity is 2. We also assign the individual resource of Claim handler 001 to the same task, then there will be three individual Claim handlers to do this task. Two of them come from the role requirements and are allocated dynamically, the other one come from the individual resource requirements and is fixed to Claim handler 001.

By defining the criteria for individual resource requirements, you have the resource allocated to a task dynamically. In this case, the field for Individual resource column has to be set to Person or Staff as shown in previous Figure 4-17 on page 94.

View the whole process in Swimlane

By clicking **Launch Swimlane Viewer** button in the palette as shown in Figure 4-45 on page 116, we get the options of viewing the process diagram in swimlane format sorted by Role, Resource, Organization unit and Location.
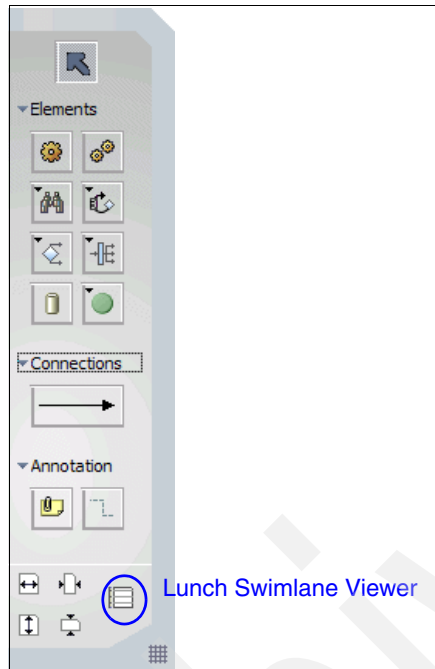
Figure 4-45   WebSphere Business Integration Modeler palette

Figure 4-46 on page 117 shows the swimlane view of the process sorted by role.
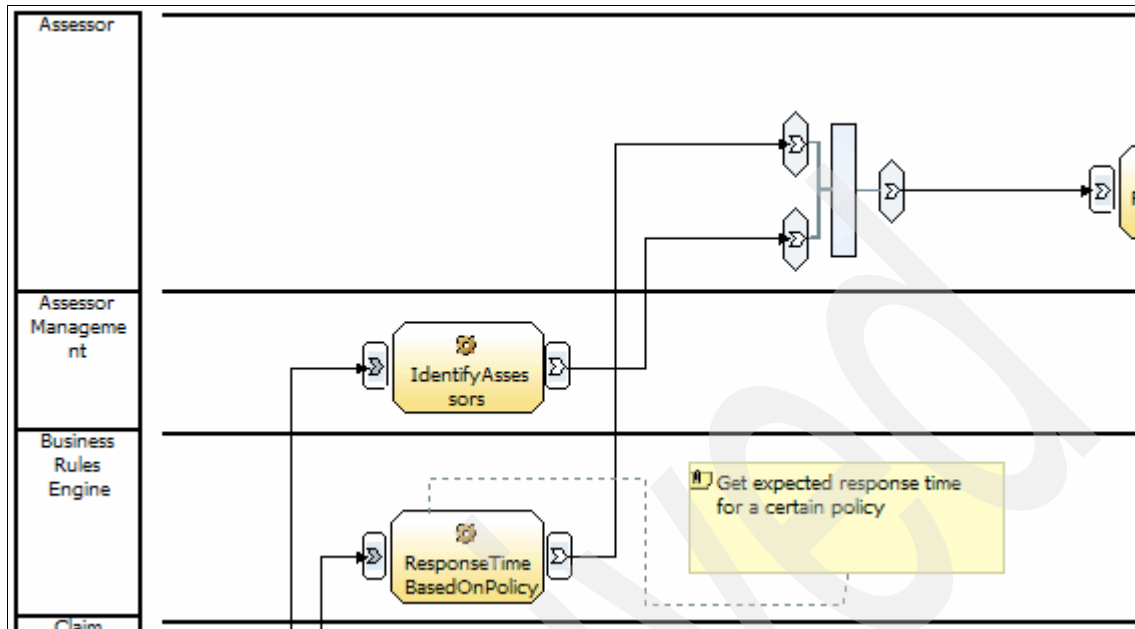
Figure 4-46   Swimlane view of the process sorted by role

4.3.6 Features attractive to business analyst

Analyzing a process model is one of the major steps in the cycle of developing a process model. WebSphere Business Integration Modeler provides a variety of analysis functions that allow you to extract specific types of important business information from your models.

There are two kinds of analysis, Static and Reports:

► Static analysis is performed based on modeling information.

   For example we use **Qualified Resources for** Roles to view how many resources for certain roles. We can access this by right-clicking anywhere in Project Tree → **Static Analysis** → **Resource Analysis** → **Qualified Resources for Role**. In the window, click **Select all** if you want to see all the resources, then click **Finish**. The result is shown in Figure 4-47 on page 118.

Role Name	Resource Name
+ System administrator	
− Automatic	
	Computer 003
− Claim Manager	
	Claim Manager
− Claim Supervisor	
	Claim Supervisor
− Claim Handler	
	Claim Handler 001
	Claim Handler 002
	Claim Handler 003
	Claim Handler 004

Figure 4-47   Some Qualified Resources for Roles analysis results

- We perform dynamic analysis based on the results of process simulation to get the most from it. There are four types of such analyses. They are:

- **Aggregated Analysis**: determines information about activities and resources used in all process instances generated during a simulation.

- **Process Analysis**: performs a process instances summary analysis to list the

  - **Process cases**
  - **Process instances**

  generated in a process simulation run, and to show the probability of occurrence for each process case.

- **Process comparison Analysis**: compares the weighted average analysis results for two simulated processes that use the same input parameters.

- Another feature is **Queries** provided. Queries return information about model elements of one specified type.

- You can use queries to confirm the content of your models, and as the basis for creating reports.

- There are 25 predefined queries in the **Queries** category in the Project Tree. You may also define new queries.

▶ **Reports** is a formatted presentation of information relating to a model or to the results of analyzing a process simulation.

- WebSphere Business Integration Modeler provides in total 95 types of report template in the categories of Basic profile, Intermediate Profile, Advanced profile and Collateral reports.

- You can generate reports after a query, a static, or dynamic analysis.

– You can view a report online in WebSphere Business Integration Modeler, print it, or export it to a variety of file formats.

4.4 Simulate the process

After creating the business process model, we can simulate it in WebSphere Business Integration Modeler to view the performance and make adjustments accordingly.

If you want to start with a complete model at this point, create a new project or a fresh workspace and import .\SG24-6636\Modeler\Projects\PreBPEL.zip. The results of simulation are in .\SG24-6636\Modeler\Projects\Simulation Results.zip. See Appendix A, "Additional material" on page 501.

> **Tip:** If you change anything in the process model you need to create a new simulation to pick up changes to the process or any new resources you have defined, for example.

4.4.1 Create a simulation snapshot

We can now run a simulation for the process **RequestExternalReports**.

1. Right-click the process model in Project Tree, choose **Simulate**. A simulation snapshot of the process is created as shown in Figure 4-48 on page 119.
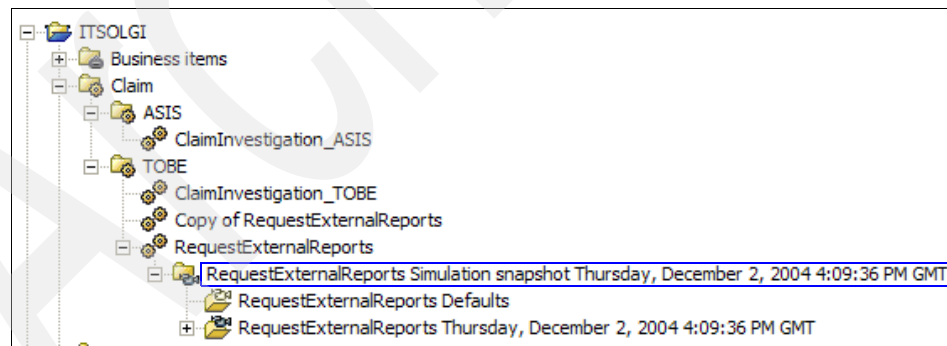
Figure 4-48   Simulating a process

The snapshot contains a:

– Copy of the business process
– Copy of all model elements for the project at that particular point in time

– Set of local preferences for simulation attributes as shown in Figure 4-49 on page 120
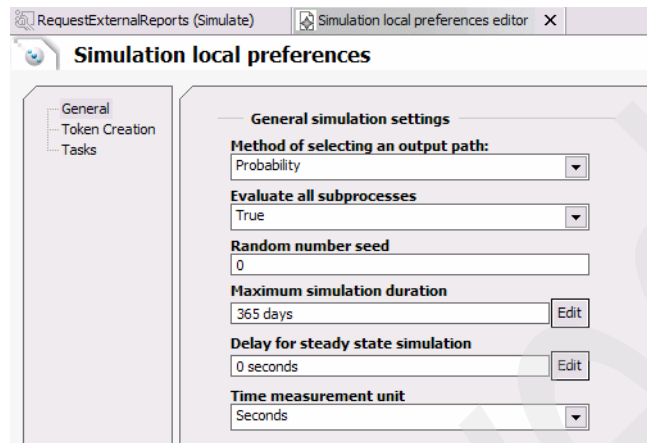
Figure 4-49   Simulation local preferences for process snapshot

> **Note:** Many values in the local simulation preferences inherit from the Global simulation preferences. The simulation preferences for a certain process and task also inherit from local simulation preferences. But the values defined for a certain process or task are the ones taken for simulation.
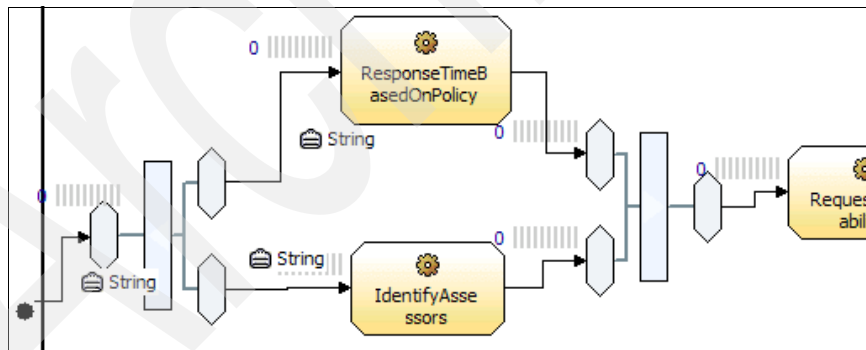
Figure 4-50   Process snapshot in Simulation Editor

2. By clicking the blank area of the process snapshot in **Simulation Editor** as shown in Figure 4-50, the simulation preference for the whole process is opened, which is shown in Figure 4-51 on page 121.

Figure 4-51   Simulation preference for the whole process

3. In addition, we can specify a task's simulation attributes by selecting it. Figure 4-52 displays the IdentifyAssessors simulation attributes. They are complimentary to the ones defined in process simulation preferences.

Figure 4-52   Simulation preference for a task

4. Check the values in the other tabs of the process simulation preferences. In particular, modify the Resources tab so that it only includes the resources we are going to use in the simulation, see Figure 4-53 on page 122.
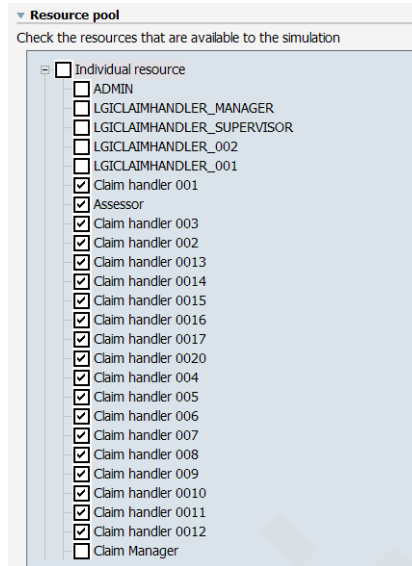
Figure 4-53   Select resources for simulation

4.4.2 Define values for simulation

There are certain fields that we can set for simulation. Here we discuss just a few of them.

> **Tip:** Click the mouse in the input box of a simulation parameter field and press F1. You get a short explanation of the field.

Set the number of simulations to run

We can set the number of simulations to run by the creation of tokens. A *token* represents a unit of work that is received by a process and transferred between different activities in the process flow. By specifying token creation settings, you define the quantity and rate of inputs that the process handles in a simulation run. You can also create tokens for each task. You would not normally do this as the work flows from the input to the business process, but you might want to analyse part of the flow.

1. Click **Inputs** tab in the **Attributes** view of the process simulation preference shown in Figure 4-51 on page 121 Select the row showing the input data. Your screen should look similar to Figure 4-54 on page 123.

Figure 4-54   Token creation settings

- We can choose Time trigger if the tokens arrive in a certain interval regularly or choose Random time trigger if the tokens arrive randomly.

- If **Random time trigger**, we can choose a certain distribution.

- If we select **Time trigger**, we can specify the start time for token to be created and **Recurring time interval for bundle creation**.

- By specifying a suitable **Recurring time interval for bundle creation**, the time period that we simulate is determined. For example, we define:

  - Number of tokens per bundle is 2
  - Total number of tokens is 20
  - Recurring time interval for bundle creation is 3 minutes
  - Then the simulated period is: (20/2)*3=30 minutes

Tip: It is always a good practice to run a simulation for the whole process by setting the total number of tokens to 1 before you change any other settings. In this way, you get a chance to check the correctness of the process logic flow while waiting for the shortest time for the simulation to finish.

2. To predict the claims volume, we use industry figures which show an average of 1.5 claims per 100 motor policies per year. For the merged company, this gives us an expected claim volume of:

   - 6,000,000 * 1.5 /100 = 90,000 claims per year

   - Assume 250 working days, 90,000/250 = 360 claims per day

   - Assume 70% of claims require assessment, 360*70% = 252 assessments per day

   - We can then obtain the recurring time interval for bundle creation: 8*60/252 = 1 minute and 54 seconds, where the number of tokens per bundle is 1

3. Set one hour's worth the claims based on the figures in Step 2 to keep the simulation of a manageable size. This date took about five minutes to simulate on a 2Ghz Intel® Pentium® IBM Mseries Thinkpad. See Figure 4-55.

Figure 4-55   Token creation settings for 1 hours simulation

Set durations for each task

For each task in the process to simulate there are two fields to complete in the simulation snapshot[3], see Figure 4-56 on page 125,

► Resource wait time

   This is the maximum time to wait for a resource - set this to a long period of time (say a year) unless you DO want to terminate a simulation if a resource is unavailable.

► Processing time

   This is the time required for the resource to do this task. The sum of processing time and resource wait time is sometimes known as *elapsed* time in project scheduling.

   –   Processing time is different to the time set on the role requirement. The processing time might exceed the time the role is required for the task. For

---

[3] Since WebSphere Business Integration Modeler 5.1.1.2 these values can be set in the model as well as the simulation snapshot. This is convenient for doing multiple simulation snapshots.

instance, the role might initiate the task, but not be required for the total processing time.

For example, suppose that the a Claim handler can handle multiple claims at the same time by claiming more than one claim at a time from their workitem list and working on them all at the same time. Also suppose that the business analyst has decided that an insurance claim must by examined by a Claim handler within a day of it arriving on their workitem queue.

- The resource wait time would be modelled as one day.
- The Claim handler role requirement time might be modelled as ten minutes - the amount of time the Claim handler actually spends dealing with a specific claim
- The processing time might be set at one hour, the amount of time between the Claim handler claiming the workitem, and finally deciding upon the outcome and completing the manual activity.
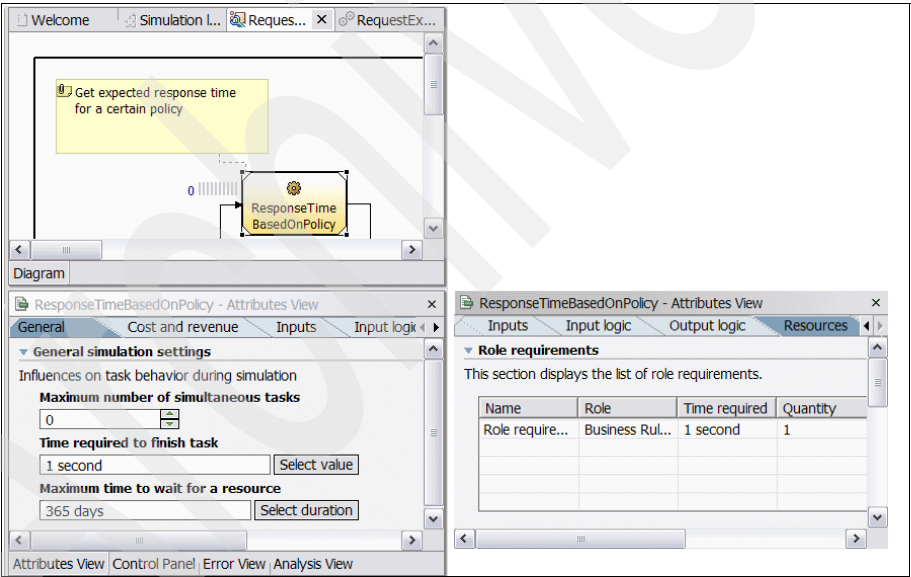
Figure 4-56   Resource Requirements

In some cases, the processing time and role requirement time are set to the same value. This is when the role spends all the time doing the task.

However in some cases, these two fields are set to different values. For example, in the RequestAvailability task, our system sends out the request to each of the possible assessors and expects them to reply indicating if they are available or

not. No one may pick the request for hours. However after the request is picked up, the assessor only needs minutes to check their schedule and return their availability. In this case we set **Time required to finish task** to 4 hours, and **Time required** to 2 minutes.

Table 4-3 lists the time durations set for tasks. Enter these figures into the simulation snapshot as in Figure 4-56 on page 125.

Table 4-3   Task durations set for simulation

Task	Processing time	Time role required	Resource
ResponseTimeBased OnPolicy	1 second	1 second	Business Rules Engine
IdentifyAssesors	1 second	1 second	AssessorManagement
RequestAvailability	4 hours	0[a]	Assessor
SelectAssessor	1 second	1 second	Business Rules Engine
ManualSelect Assessor	2 minutes 2 minutes	2 minutes 2 minutes	Claim handler Claim handler Desktop
RequestAssessment	1 hour	0[b]	Assessor
AssessorSendReport	2 days	0[c]	Assessor
StoreReport	1 second	1 second	Document Handler

a. By setting this to 0, we do not consider assessor's time and cost for Request-Availability in the simulation.
b. By setting this to 0, we do not consider assessor's time and cost for Re-questAssessment in the simulation.
c. By setting this to 0, we do not consider assessor's time and cost for Asses-sorSendReport in the simulation.

Set output for Decision

For the Decision elements, we set the probability for each output as shown in Table 4-4.

Table 4-4   Probabilities to Decision outputs

Decision	Probability to Yes	Probability to No
Any assessor?	90%	10%
Confirmed?	95%	5%

The settings in Table 4-4 on page 126 imply the following scenario:

► There is a 90% chance we get at least one assessor available. There is a 10% chance there is no assessor available, therefore the Claim handler needs to assign one.

► There is a 95% chance the assessor who was available before can still do the assessment. While there is a 5% chance the assessor cannot do the assessment anymore, because of exceptional circumstances, such as the assessor has an emergency.

We then specify **Method of selecting an output path** to **Based on probabilities to single path** as shown in Figure 4-57.

Figure 4-57   Screen shot for the Output logic setting of a Decision element

4.4.3 Run a simulation

To start a simulation, switch to **Control Panel** and press **Start**, as shown in Figure 4-58 on page 127.

Tip: If you cannot find the control panel, click **Window** → **Show View** → **Control Panel** in the main action bar.

We can always choose to pause, stop, or step-by-step a simulation by clicking appropriate button in the Control panel.

Figure 4-58   Simulation control panel

During the simulation run, we can see animations and the number of tokens transiting from one element to the next. Figure 4-59 shows a screen shot of such animation, where 13 tokens are being processed by RequestAvailability task. To complete the simulation quickly, disable the animation.
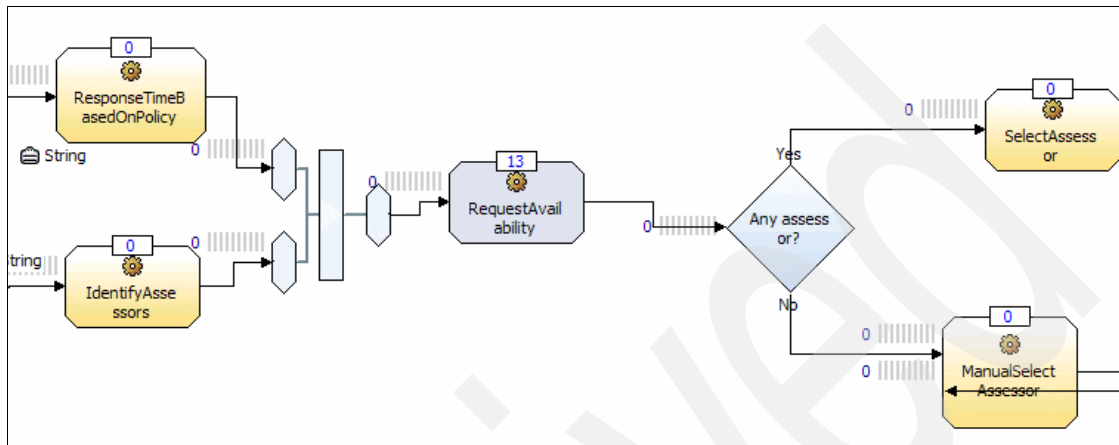
Figure 4-59   Simulation animation

After the simulation finishes, the result is stored in the Project Tree as shown in Figure 4-60. We can distinguish different simulation results from the time when it finishes.
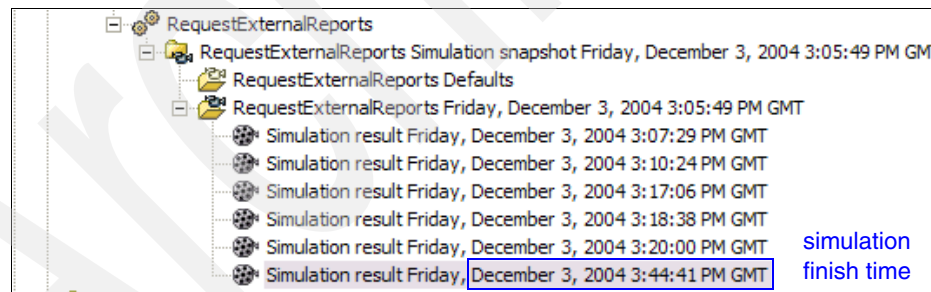
Figure 4-60   Simulation results stored in Project Tree

Tip: If a simulation stops and you receive a message saying that there were not enough resources available to complete the simulation, the message refers to simulation resources, not to a system problem. You need more resources available for the process. You probably have configured something wrongly. Check the following:

1. Maximum wait times are too small and not allowing the simulation to complete.

2. The Assessor resource is a member of external resources.

3. The Activities performed by assessors have the organization set to External Resources, not person or staff.

4.4.4 Simulate the whole claim investigation process

We require simulations of both the ASIS and TOBE processes to compare them.

► Simulate ClaimInvestigation_ASIS

– Set the time to manually process RequestExternalReports to 30 minutes in the resources tab of the InvestigateClaim local activity before creating a simulation snapshot

► Simulate ClaimInvestigation_TOBE

– This time there are 360 claims per day and 70% require evaluation.

There are two ways to perform a simulation of the whole ClaimInvestigation_TOBE process which contains subprocesses.

Note: You cannot evaluate a particular subprocess. However, you can choose an option that allows you to specify whether to evaluate all subprocesses, including subprocesses that have incomplete content. You can select No for Evaluate all subprocesses in the simulation attributes for a process, and you can set a default value for this attribute either as a local or as a global preference. This setting means the simulation will skip the subprocesses.

1. Provide values for all the subprocesses based on previous simulations and only simulate the new process.

Note: The project .\SG24-6636\Modeler\Projects\Claim preBPEL.zip or .\SG24-6636\Modeler\projects\Parms Set are good starting points for the TOBE simulation. The latter has the RequestExternalReports process simulation parameters set so we now only have to set the ClaimInvestigation_TOBE snapshot parameters.

Open the snapshot in Simulation editor, and set **Evaluate all** subprocess to **No** in the **General** tab of **Attributes** view.

a. Click the **RequestExternalReports** subprocess in Simulation editor, set **Time required to finish this task** in **General** tab to the **weighted average Process cycle time** of RequestExternalReports process, which we can obtain from dynamic analysis of previous simulation results. For example, two days, five hours, and seven minutes.

b. Click the RequestExternalReports subprocess in Simulation editor, set **Cost per execution of the task** in **Cost and revenue** tab to the **weighted average total cost** of RequestExternalReports process, ($0.105, for example).

2. Simulate the processes and all its subprocesses.

Leave the **Evaluate all subprocess** button as **Yes** in the **General** tab of **Attributes** view.

a. Right-click the canvas area of the ClaimInvestigation process → **Expand All.** See Figure 4-61.

b. Select the activities in the **RequestExternalReports** process and set their simulation parameters as before.
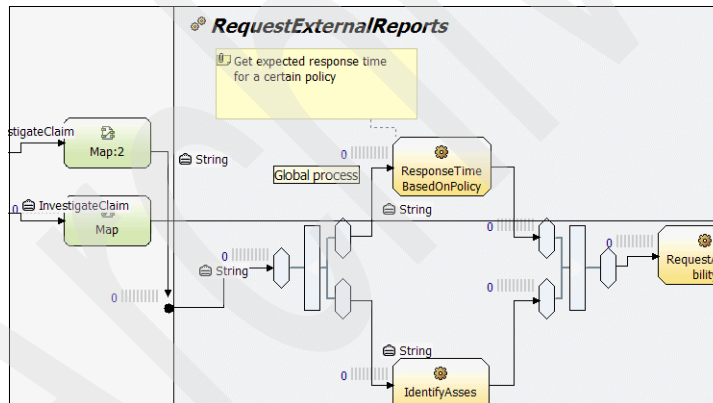
Figure 4-61   Expanding subprocesses in the simulation editor

4.4.5 Analyze the results

After a simulation is finished, a list of simulated process instances are listed in the Process tab of the control panel as shown in Figure 4-62 on page 131. For each instance, we can get the information of the process start time and end time, total cost, total revenue, and total profit.

Figure 4-62   Immediate results for process instances after simulation

In the Tasks tab, we obtain the cost for each task in each process instance. Figure 4-63 shows the tasks' cost for **ClaimInvestigation_ASIS** process instance.

Figure 4-63   Immediate results for tasks after simulation

The Connections tab shows the tokens transferred by each connection.

Dynamic analysis on simulation results

Based on the stored simulation results, we are able to perform dynamic analysis. Various types of dynamic analysis can be accessed by right-clicking the simulation results in **Project Tree** → **Dynamic Analysis**. Below we list several analyses performed.

▶ **Activity Resource Allocation** shows a summary of the resources allocated for each activity. As shown in Figure 4-64 on page 132, the resource allocation for the activities in RequestExternalReports are listed.

Figure 4-64 Analysis: Activity Resource Allocation for RequestExternalReports

► **Resource Utilization Analysis** lists all the resources that are allocated during the simulation. Table 4-5 shows one of the resources, Claim handler 001, as an example.

Table 4-5   An example shown in Resource utilization analysis

	Claim handler 001
Allocated from	December 6, 2004 3:14:00 PM GMT
Allocated to	December 6, 2004 3:16:00 PM GMT
Allocating process instance	RequestExternalReports 39
Allocating activity	ManualSelectAssessor
Allocating activity start time	December 6, 2004 3:14:00 PM GMT
Quantity allocated	1
Allocation duration	2 minutes
Shortage duration	0 seconds
Allocation cost	$0.50

► **Process cost analysis** is performed against RequestExternalReports process.

As the results in Figure 4-6 on page 133 show, there are four possible cases to proceed the RequestExternalReports process. The reason for this is the process has two decision points. Each case has the probability to happen. The weighted average cost for the whole process is $0.080866.[4]

Table 4-6   Process cost analysis for RequestExternalReports process

Case Name	Probability	Revenue	Execution	Idle	Allocated resources cost	Total
Case 1	85.50%	$0.00	$0.00	$0.00	$0.0013	$0.001
Case 2	9.50%	$0.00	$0.00	$0.00	$0.54	$0.539
Case 3	0.48%	$0.00	$0.00	$0.00	$1.078	$1.077
Case 4	4.28%	$0.00	$0.00	$0.00	$0.54	$0.539
Weighted average		$0.00	$0.00	$0.00	$0.081	$0.081

► **Activity cost analysis** is performed for ClaimInvestigation_ASIS process. We are interested specifically in the task of RequestExternalReports. The results are shown in Table 4-7.

Comparing the total average cost of this task with the equivalent figure of total weighted average cost for RequestExternalReports process, we get the conclusion that doing it manually costs LGI claims department 93 times more than outsourcing and automating it and involving a staff only when it is necessary. These figures could then be used to assist in negotiating a price with potential outsourcing suppliers.

Table 4-7   Activity cost analysis for RequestExternalReports in InvestigateClaims_ASIS

Average revenue	Average Execution cost	Average idle cost	Average allocated resources cost	Average total cost	Average profit
$0.00	$0.00	$0.00	$7.50	$7.50	($7.50)

► **Process cost comparison analysis** is performed for two simulation results of ClaimInvestigation_ASIS and ClaimInvestigation_TOBE.

   – After the simulation is run, right-click the stored simulation result in Project tree → **Dynamic analysis** → **Process comparison analysis** → in the appeared **Simulation results** window as shown in Figure 4-65 on page 134 → choose one to compare → **OK**.

---

[4] There could be more than four cases if any instance loops through the **Confirmed**? test more than once.
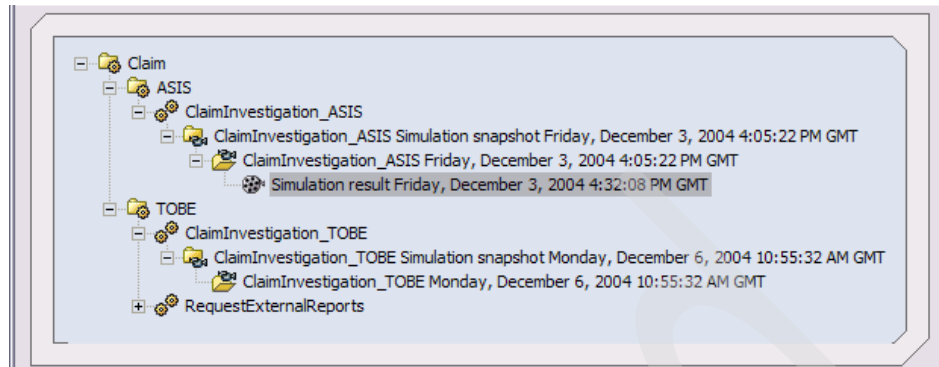
Figure 4-65   Simulation result selection window for Process comparison analysis

The results are shown in Table 4-8.

Table 4-8   Process cost analysis results

Process	Revenue	Execution Cost	Idle Cost	Allocated Resources cost	Total Cost	Profit
ClaimInvestigation_TOBE	$0.00	$0.00	$0.00	$0.250634	$0.250634	($0.250634)
ClaimInvestigation_ASIS	$0.00	$0.00	$0.00	$7.750634	$7.750634	($7.750634)
Difference	$0.00	$0.00	$0.00	($7.50)	($3.740487)	($7.50)
Change	0%	0%	0%	-2,992.409%	-2,992.409%	-2,992.409%

4.5 Developing the process implementation

At this point in the process of developing the External Claim Assessor solution, the business analyst has completed defining the new process. The analyst gets approval of the process from their executive sponsor and software architect, then freezes a copy of the project, perhaps using CVS.

> **Important:** The business analyst has finished specifying the claim assessor process. The main project responsibility now passes to the software architect to design the solution. We return to the WebSphere Business Integration Modeler when the IT specialists are ready to begin their implementation of the solution.

4.5.1 Export processes

Because the whole claims process is already running in a WebSphere MQ Workflow environment, we decided to minimize changes to the running claims process by only making a small modification to it. Rather than call RequestExternalReports as a manual local activity, we now call RequestExternalReports as a subprocess. The claims process continues to run in WebSphere MQ Workflow and the new RequestExternalReports subprocess will run in WebSphere Business Integration Server Foundation.

WebSphere Business Integration Modeler has the capability to export a process both as FDL to run in WebSphere MQ Workflow and as Business Process Execution Language that runs in a number of process engines, including WebSphere Business Integration Server Foundation.

Because the changes to the claims process were so minimal, the WebSphere MQ workflow specialist decided not to export the modified claims process from WebSphere MQ Workflow as FDL, but make the necessary changes directly using the WebSphere MQ Workflow buildtime. See 11.3, "Create the ClaimInvestigation_TOBE Workflow" on page 458. See Figure 4-66 on page 136.

The specialist did need to define the ClaimInvestigation data structure in FDL in order to pass ClaimInvestigation from WebSphere MQ Workflow to WebSphere Business Integration Modeler. It would have been convenient to export the new data structures from the UML in Rational Software Architect as FDL, but Rational Software Architect doesn't support FDL. WebSphere Business Integration Modeler does, and so the WebSphere MQ Workflow specialist decided to use the Modeler to construct the FDL version of the data structure. This is explained in Chapter 11, "Modify the Claim Investigation process" on page 453.

Why import the original WebSphere MQ Workflow process into WebSphere Business Integration Modeler, modify it, then not use the result in WebSphere MQ Workflow?

Why import it?

1. The business analyst's objective in importing the process is primarily to document the as-is process, understand new process requirements, and specify the to-be process. The split between process engines is an implementation issue. The analyst needs to work with the complete process.

2. To be able to simulate both the as-is and to-be flows.

Why not export to WebSphere MQ Workflow?

1. The new FDL is substantially different from the original.

   Importing and exporting of FDL is not reversible. Using the new FDL will incur more development effort and, more importantly, more testing and potential instability in the production environment.

2. Deployment information is not imported or exported with the FDL.

3. The modifications to the process, changing the RequestExternalAssessor from a local task to a subprocess, have to be reversed before exporting the FDL. FDL does not support the concept of invoking a subprocess in a different process engine.

Figure 4-66   When to use the .fdl WebSphere Business Integration Modeler

4.5.2 Export as FDL process

Because we are not exporting the claim investigation process from WebSphere Business Integration Modeler as part of the scenario, we have included the next section just to show how to do it.

Validate process model

Before we can export the ClaimInvestigation_TOBE business process to FDL, we need to modify it for the target runtime environment, if that is necessary.

1. Copy ClaimInvestigation_TOBE to Claims category.

2. Rename ClaimInvestigation_TOBE as ClaimInvestigation.

3. Because RequestExternalReports is a subprocess run in an environment other than WebSphere MQ Workflow, we need to treat this step as an automated task in ClaimInvestigation. Open ClaimInvestigation in Process

editor → **Delete** the RequestExternalReports subprocess → **Add** a local task called RequestExternalReports. **Reconnect** the elements as in Figure 4-67.
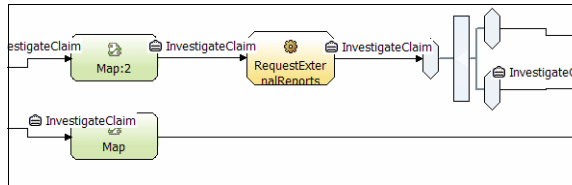
Figure 4-67   Rewire ClaimInvestigation

4. To check if the process model is valid in FDL notation, switch to **FDL** technology mode and click the blank area of the process in Process editor.

> **Attention:** You must click in the process editor of the open process to see the errors specific to that process. Clicking a process in the project tree will not alter the error view to a different process unless you open the process.

Certain constraints that are specific for FDL take effect in this mode. For details of the constrains to FDL technology mode, refer to WebSphere Business Integration Modeler information center.

```
http://publib.boulder.ibm.com/infocenter/wbihelp/index.jsp?topic=/com.ibm.b
tools.help.modeler.doc/doc/reference/techmodes/fdl.html.
```

5. If there are any FDL-specific validation errors or warnings for the process, in the Project tree, we can see the process name with error or warning symbols attached. Figure 4-68 shows the same processes with different symbols in different technology mode.
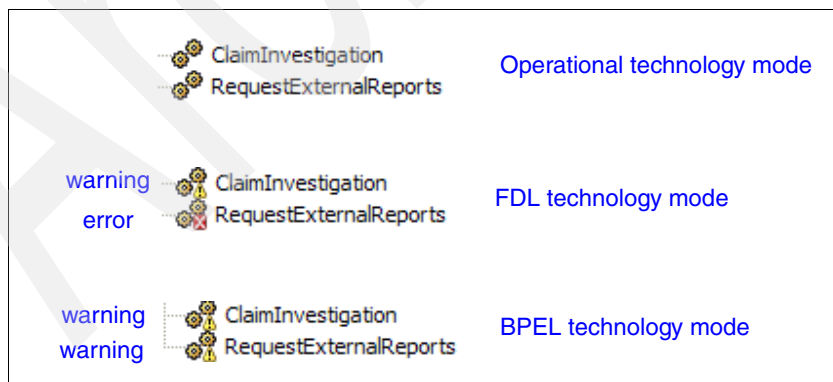
Figure 4-68   Appearance of the same project tree in different technology modes

We can see the details for every error and warming in **Error** view. Figure 4-69 shows a screen shot for the warnings shown in Error view of ClaimInvestigation process. For each error or warning, certain information is listed:
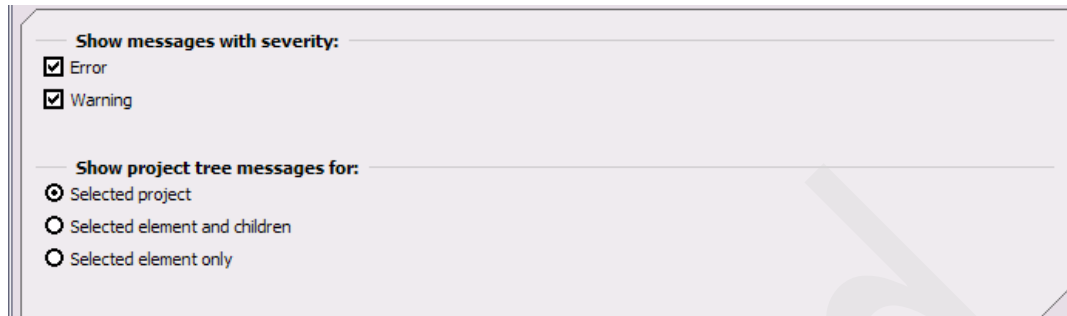
–   Description: a brief description of the error or warning.

–   Element name: the element to which the error or warning belongs.

–   Element type: the type of the element.

–   Parent name: to which parent the element belongs.

–   Patent location: where the parent is located.

–   Error code: the error code is in WebSphere Business Integration Modeler Information Center, where a detailed explanation of the error is provided.

–   Profile: in which profile you can fix the problem.

We can sort the list by clicking the column title.

	Description	Element name	Element type	Parent name	Parent type	Parent loca...	Error code	Profile (in which it can be fixed)
⚠	A resource d...	Role requirement:1	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203013W	Basic
⚠	Required qu...	Role requirement:1	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203016W	Basic
⚠	Required qu...	Role requirement:1	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203016W	Basic
⚠	Stop node "S...	Stop Node	Stop	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL200025W	Basic
⚠	A resource d...	SetClaimStat	Task	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203013W	Basic
⚠	Required qu...	ResourceRequirem...	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203016W	Basic
⚠	Time require...	Role requirement:1	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203017W	Basic
⚠	Time require...	ResourceRequirem...	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203017W	Basic
⚠	Unsupported...	Role requirement:1	Role requirement	ClaimInvestig...	Global process	/ITSOLGI/Li...	ZFL203001W	Basic

Figure 4-69   Error view for FDL export

6. By using a filter on the Error view, you can choose to view only errors that relate to a specific set of model elements or that have a specific severity level. To set a filter, click **Filter options dialog**, as shown in Figure 4-69.

7. Figure 4-70 on page 139 shows the window opened for you to set the filter options for Error view. See what happens when you change the project messages filter to **Selected element and children** and click the process in the project tree.

8. Now change it back to **Selected project**.

Figure 4-70   Set filter for Error View

9. We must correct any errors in the process before we are able to export it . There are none. We can ignore the warnings and go ahead to export the process.

Export process

To export the process, follow these steps:

1. Right-click the process name in **Project Tree** → **Export** → **WebSphere MQ Workflow (.fdl)** option as shown in Figure 4-71 on page 140 → **Next**. Pick the target directory that the process is exported to and make sure that ClaimInvestigation process is selected. Click **Finish**.

If there is no error, the export is successful. You can find the exported file with the name of the project and an *.fdl file extension. In our case, the exported file name is ITSOLGI.fdl.

If there is any error for the export, follow the error message for appropriate actions to fix it.
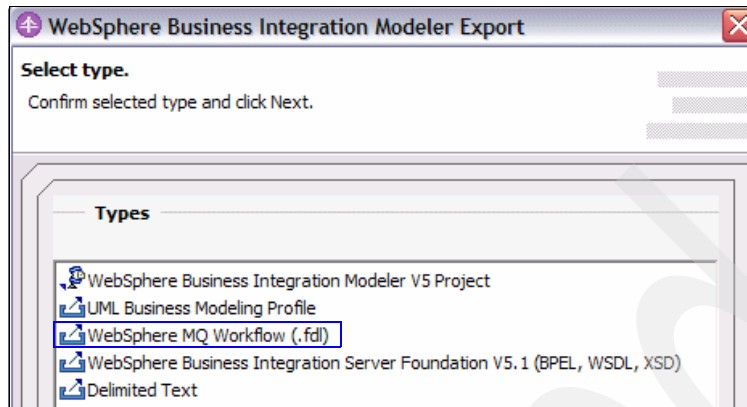
Figure 4-71   WebSphere Business Integration Modeler export wizard

You can now import the file into WebSphere MQ Workflow Buildtime.

Elements mappings

In this section we show how to map the elements from WebSphere Business Integration Modeler to WebSphere MQ Workflow.

Figure 4-72 shows the first part of the process mappings for WebSphere Business Integration Modeler and shows the process mappings for WebSphere MQ Workflow. The digits with red color indicate the one-to-one mappings.
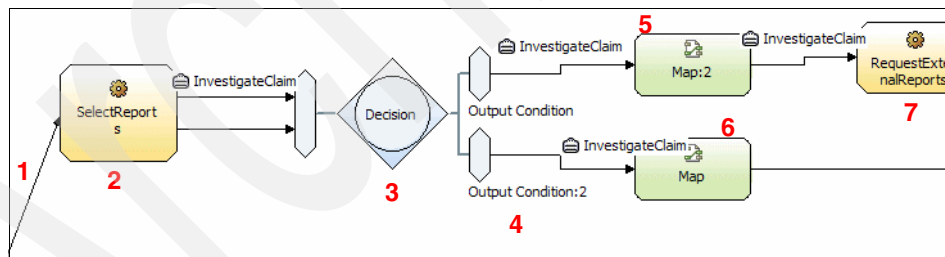
Figure 4-72   Mappings for ClaimInvestigation: part 1(a) of 3 of the WBI Modeler

► Input of the process (1) is mapped to the Data source node (1).

► SelectReports (2) local task is mapped to the SelectReports (2) program activity. A program with a randomly generated name is assigned to it. The role requirement is mapped to Member of roles in the Staff2 tab of the SelectReports' properties.

► Decision (3) is mapped to the Decision (3) program activity, to which the FMCINTERNALNOOP program is assigned.

- The Output conditions (4) are mapped to the Transition conditions (4) from the Decision to Map and Map2 activities respectively.

- Map:2 (5) is mapped to the Map (5) program activity. A program with a randomly generated name is assigned to it. The Contents in the Description of Map:2 is mapped to the Description field in the General tab of Map's properties.

- Map (6) is mapped to the Map (6) program activity. A program with a randomly generated name is assigned to it.

- RequestExternalReports (7) local task is mapped to the RequestExternalReports (7) program activity. Again a program with a randomly generated name is assigned to it.
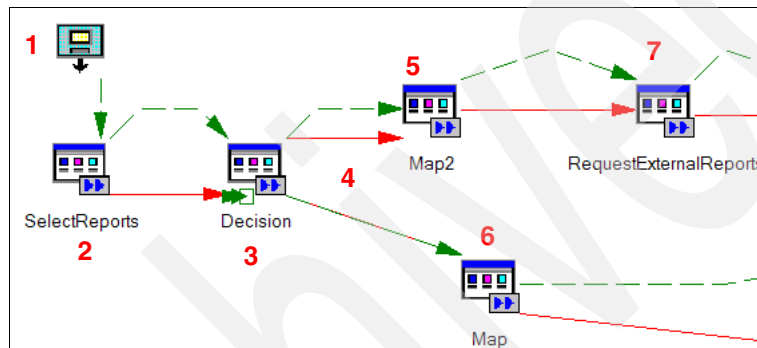
Figure 4-73   Mappings for ClaimInvestigation: part 1b of 3 of the WMQ Workflow

Figure 4-74 shows the second part of the process mappings in WebSphere Business Integration Modeler and Figure 4-75 for WebSphere MQ Workflow. The digits with red color indicate the one-to-one mappings.
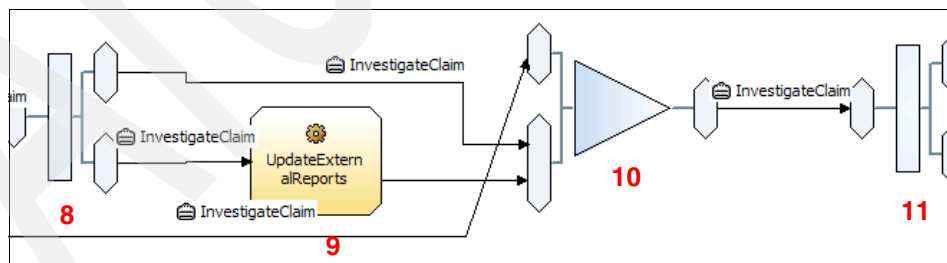
Figure 4-74   Mappings for ClaimInvestigation: part 2(a) of 3 of the WBI Modeler

- Fork (8) is mapped to the Fork (8) program activity, to which the FMCINTERNALNOOP program is assigned.

- UpdateExternalReports (9) local task is mapped to UpdateExternalReports (9) program activity. A program with a randomly generated name is assigned to it. Note that there is no data connector to the next element.
- Merge (10) is mapped to the Merge2 (10) program activity, to which the FMCINTERNALNOOP program is assigned. Merge2 takes two data inputs and three control connectors. The condition to start it is when all incoming connectors are true.
- Fork (11) is mapped to the Fork2 (11) program activity, to which the FMCINTERNALNOOP program is assigned.
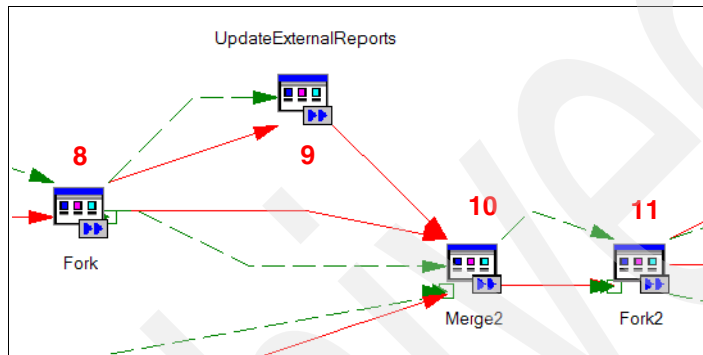
Figure 4-75   Mappings for ClaimInvestigation: part 2b of 3 of the WMQ Workflow

Figure 4-76 shows the last part of the process mappings for WebSphere Business Integration Modeler and Figure 4-77 for WebSphere MQ Workflow. The digits with red color indicate the one-to-one mappings.
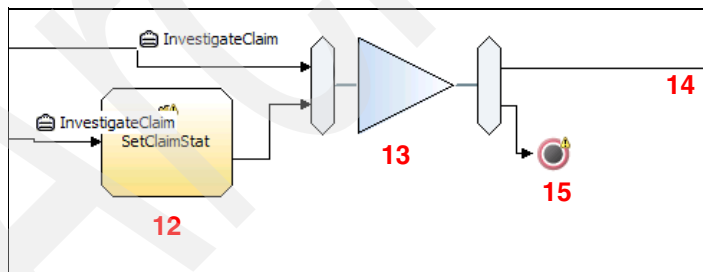
Figure 4-76   Mappings for ClaimInvestigation: part 3a of 3 of the WBI Modeler

- SetClaimStat (12) is mapped to the SetClaimStat (12) program activity. A program with a randomly generated name is assigned to it.
- The Merge (13) local task is mapped to the Merge (13) program activity, to which the FMCINTERNALNOOP program is assigned. Merge takes one data

input and two control connectors. The condition to start it is **All incoming connectors true**.

► The Process output (14) activity is mapped to the Data sink node(11).

► The Stop node (15) is not imported as it is not supported by WebSphere MQ Workflow.
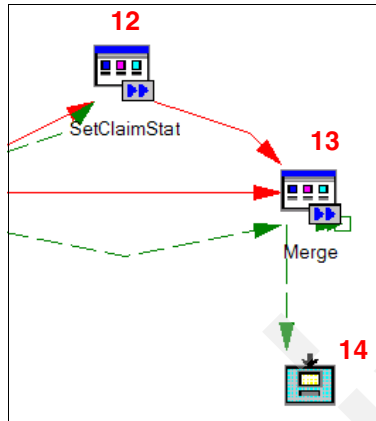
Figure 4-77   Mappings for ClaimInvestigation: part 3b of 3 of the MQ Workflow

No deployment-related information has been generated by the export from WebSphere Business Integration Modeler.

4.5.3 Export RequestExternalReports as a BPEL4WS process

The new RequestExternalReports business process will be executed in WebSphere Business Integration Server Foundation. The IT specialist is responsible for exporting the process as BPEL4WS. This task is not one performed by the business analyst.

The IT specialist responsible for exporting the RequestExternalReports process as BPEL should read chapter 5 of *"BPEL4WS Business processes with WebSphere Business Integration: Understanding, Modeling, Migrating"*, SG24-6381. This chapter explains the mappings between WebSphere Business Integration Modeler and BPEL4WS elements. Chapter 6 of the same book helps you to understand the exported artifacts from WebSphere Business Integration Modeler to BPEL4WS.

Validate process model

We check if the RequestExternalReports process is valid in BPEL notation before the export. There turn out to be numerous warnings again which can be ignored, and eight errors that need to be fixed.

In WebSphere Business Integration Modeler, switch to BPEL technology mode and open RequestExternalReports in the process editor. In this technology mode, we see the BPEL-specific errors and warnings. When the process model lacks some information that is mandatory for BPEL4WS, or the elements include some objects or settings which are not supported in BPEL4WS, WebSphere Business Integration Modeler lists all the problems with the BPEL. Similar to the **FDL** technology mode, we must correct the errors before exporting the model, but we can ignore the warnings.

There are certain best practices for BPEL mode validated with WebSphere Business Integration Modeler Advanced Edition v5.1.1. Some of them are also applied to FDL mode.

1. Ensure that each input criterion is associated with one and at most one output criterion[5]. In other words, given an input criterion, there is one specific output criterion that can be triggered. This corresponds to a WSDL operation, which can have at most one output message defined. If you must associate an input criterion with multiple output criteria, try to introduce a decision node to model the exclusive branching logic.

   There are four errors in RequestExternalReports process that are associated with this.

   > **Tip:** Make sure that RequestExternalReports is open with all changes saved. Click an error message and the relevant element in the process model is highlighted to show where the error lies.

   These errors are due to the Input criteria of local tasks ManualSelectAssessor and RequestAssessment not being associated with any output criteria. To correct them, switch to **Advanced** user profile → select **Advanced Output Logic** in Attributes view → click the **Output criteria** to associate with → click the boxes next to the **Input criteria**. Both boxes should be selected Figure 4-78 on page 145 shows the correct settings.

2. Repeat this for both ManualSelectAssessor and RequestAssessment and save the process. By refiltering the error messages, we should be down to five errors.

---

[5] This also applies to FDL mode.

Figure 4-78   Advanced output logic settings

3. Ensure that each process has only a single input criterion and a single output criterion. You can check this by going to the specification tab of the Process Editor. For example, to check if you have a single input criterion → **Input specification** → **Input Logic**. Make sure you have only one input criterion listed in the Input settings table list as shown in Figure 4-79.

Figure 4-79   Input criteria setting for process

4. For tasks and services, if you want WebSphere Business Integration Modeler to generate a port type with multiple operations, define multiple input criteria for the task or service. Each input criterion must be associated with one and at most one output criterion.

5. Ensure that the process model does not contain any downstream tasks connected back to upstream tasks, which is permitted in WebSphere Business Integration Modeler but not in BPEL. This also applies to FDL mode. Use a while loop instead.

There is one error that belongs to this type in RequestExternalReports, which is due to the output **No** of decision node **Confirmed?** is connected to the input of **ManualSelectAssessors** as shown in Figure 4-80.
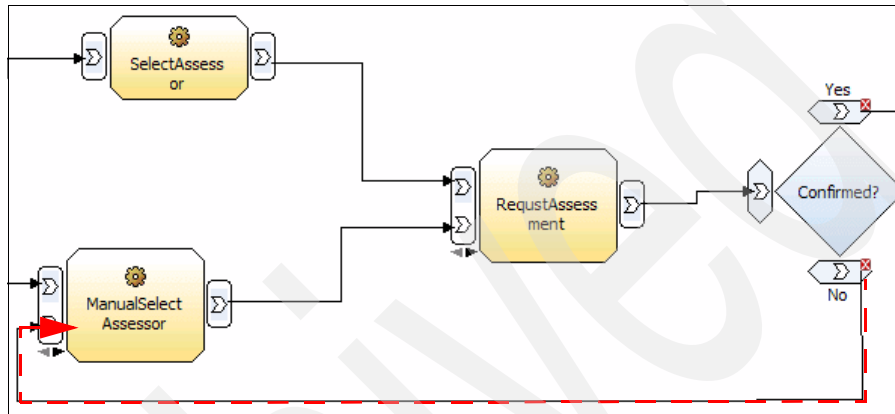
Figure 4-80   Downstream task connecting back to upstream task

To resolve this, we add a local repository to the diagram, and connect the output **No** of decision node **Confirmed?** to the input of this local repository as shown in Figure 4-81. A dummy data type of String is added.
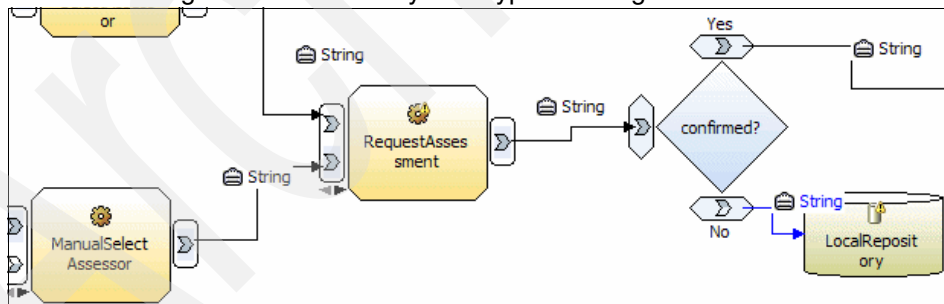
Figure 4-81   Modified part of RequestExternalReports process

We change the Input:2 of ManualSelectAssessors to take data from Repository. (Save the process before changing ManualSelectAssessors) The complete settings for it is shown in Figure 4-82.

Figure 4-82 Input settings for ManualSelectAssessor local task

6. Ensure that the minimum and maximum number of items needed have the same values for the connected object input and output between two nodes. This also applies to FDL mode. For example if:

   – An object output of Task1 is connected to an object input of Task2.

   – The object output of Task1 has the minimum number of items needed set to 3 and the maximum number of items needed set to 5.

   – The minimum number of items needed on the object input of Task2 should be set to 3 and the maximum number of items needed should be set to 5. Otherwise, incorrect BPEL will be generated.

7. For a decision node, you should define a condition for each of the decision output branches. This also applies to FDL mode.

   There are four errors related to this type, which are due to the two decision nodes's transition conditions not being set. To correct this, we add a dummy data item of type String to the decisions and define the conditions using Expression Builder. The steps are:

   a. Make sure you are in advanced mode and the RequestExternalReports process has been saved.

b. Select **Any Assessor**? → **Attributes View** → **Outputs** → **Select an output** → **Associated data** → **String.**

c. Select **Output Branches** → **Yes** → **Details** → **Contents** → **Output:2** (or whatever the name is) → **Edit Expression** (see Figure 4-83).

Figure 4-83   Setting the conditions to output branches of decision nodes

d. In the Expression Editor window that opens, select **First term** → **Modeling artifact** → Navigate to and select the **Any Assessor Input** → **Operator** → **Is equal to** → **Second term** → **Text** → **Second term details** → **Yes** → **OK.** SeeFigure 4-84 on page 149.
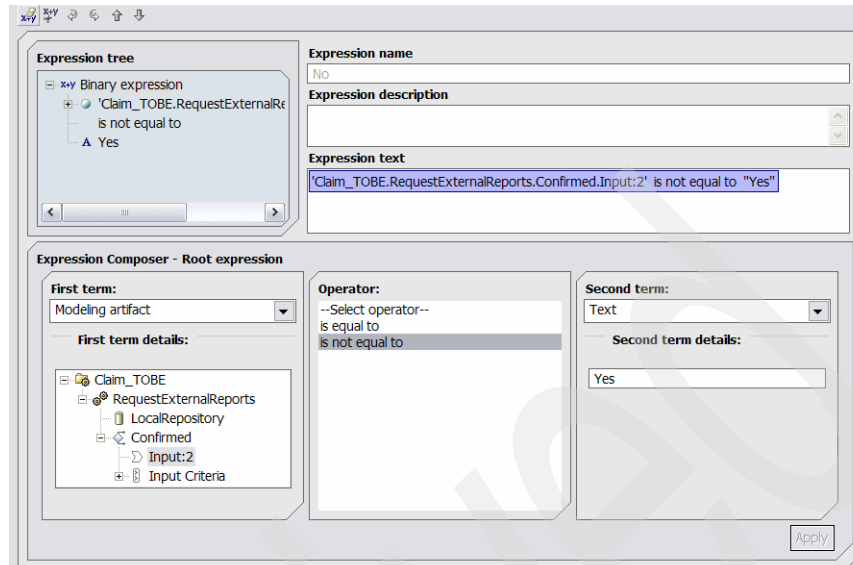
Figure 4-84   Using Expression Builder to build condition

8. You might have some other errors to correct. Perhaps some of the input and output connectors have no associated data? Look for ungreyed input or output chevrons in the input and output criteria. Convert them to string and reconnect the local elements. See Figure 4-85.

Figure 4-85   Further errors needing correction before exporting BPEL

9. When you have associated **String** with all the connections you might now see the error `DBL110014E – mismatched minimum and maximum number of items`. Modify the input to RequestAvailability to have a minimum and maximum number of items needed to 2.

10. When you have saved all the changes, perform a static analysis again to make sure all the connections are sensible.

Some WebSphere Business Integration Modeler model elements cannot be transformed to BPEL, either because there is no equivalent construct in BPEL or because the semantics of similar constructs in BPEL are different. They are Notification broadcaster, Notification receiver, Observer, Timer, For loop, Do-while loop and Global repository. We cannot select these elements in BPEL mode.

Export process

After correcting all the errors, save the process. If there is no error, we can export the process into BPEL even if there are some warnings remained.

1. Right-click RequestExternalReports process in the project tree, and select **Export**.

2. In the export wizard, select **WebSphere Business Integration Server Foundation V5.1 (BPEL, WSDL, XSD)** as shown in Figure 4-86 and click **Next**.
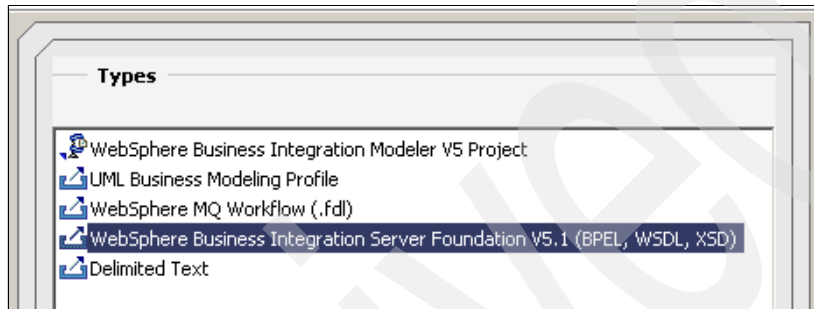
Figure 4-86   WebSphere Business Integration Modeler export wizard 1

3. In the next window, specify the target directory by clicking **Browse**. Select **Export Specific objects** and expand the project tree. Select **RequestExternalReports** process → **OK.**
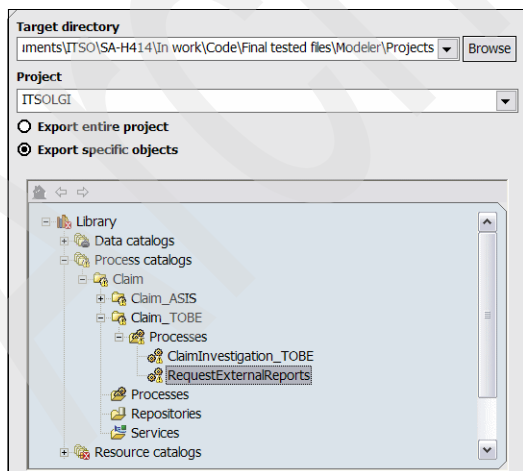
Figure 4-87   WebSphere Business Integration Modeler export wizard 2

4. Click **Next** to specify the process execution mode. Select **Long-running (receive/reply)** → **Finish**.

5. If there is any error for the export, follow the error message for appropriate actions to fix it.

You can now import the files into WebSphere Studio Application Development Integration Edition for development. See Chapter 10, "Build the Request External Reports process" on page 377.

> **Note:** A copy of the exported BPEL is in the .\SG24-6636\Modeler\BPEL Export directory. There is also a zipped project containing the process after the BPEL fixes for you to import and examine in .\SG24-6636\Modeler\Projects\PostBPEL.zip

4.6 Summary

In this chapter we showed you how to use WebSphere Business Integration Modeler to model a business process, simulate it, and export FDL and BPEL to start the implementation. The next step is to hand over the project to IT architect to design the system and decide the solution architecture using Rational tools.

5

System Architecture

This chapter describes the system architecture that provides the platform for the External Claim Assessor solution.

The architecture and design of a solution can be thought of as being developed through a series of progressive elaborations, rather than in discrete steps. For your readability, we split the architectural work into two chapters. This chapter describes the overall system architecture. It forms the platform for the solution architecture described in Chapter 6.

We demonstrate how to use the Patterns for e-business to guide us through the steps to select a reference architecture. In the first step, Collating Requirements, the technique shows how the process model developed by the Business Analyst can be incorporated into the UML model being developed by the IT architect as part of the process of understanding business requirements.

Chapter 6, "Solution Architecture" on page 187 looks at the solution design and how to develop the solution architecture.

153

5.1 Selecting the architectural patterns

We are following the steps in Figure 3-10 on page 60 to select the appropriate architectural pattern for the external claim assessor solution. This is quite a lengthy process. To help you navigate the process, here is a summary of the steps:

► Step 0: Collate requirements.

In the these preliminary steps, the solution architect collates the requirements that have been captured in the workshops sessions with the business analyst. Using the integration of WebSphere Business Integration Modeler with Rational Software Architect as much as possible, the solution architect starts to create the elements of the solution, such as use cases, roles and components, that will be used in the architectural model.

► Step 1: Select a Business Integration Pattern.

In the first step we select the kind of business integration patterns we will use. This has a major affect on the solution architecture. Are we going to want a data or process focused integration? Is collaboration between users a requirement? Do we need to include partner applications in the solution?

► Step 2: Select the application pattern.

In the next step we consider the business integration patterns we have selected (we end up selecting two) and for each select an application pattern. To do this we construct a collaboration diagram to describe the as-is system, and using the new components and use cases, construct the to-be collaboration diagram with the components split between the two integration patterns we chose. As there is overlap between the integration patterns there is flexibility about which components to include in each application pattern.

► Step 3: Select and merge the runtime patterns.

The e-business patterns provides both service-oriented and generic styles of the runtime patterns for each of the application patterns. We now map our application components onto the pattern components. Again there are choices to be made, steered by factors such as the IT strategy and constraints. In the end, we end up with a hybrid solution.

► Step 4: Apply the product mappings.

Finally, we apply the product mappings to the runtime pattern. This becomes our reference architecture which we capture by creating a deployment diagram in Rational Software Architect. The reference architecture has some major uses:

– It is a starting point for creating the physical environments on which we are going to build, test, and run the solution.

- It is a contract between the solution architect and the Infrastructure architect which defines how the solution will be deployed.
- It details how the solution collaboration is distributed across different devices and therefore which IT specialists and tools are required to build the solution. In deployment diagrams, the term *device* refers to a type of node that represents a physical computation resource in a system, such as an application server.
- It forms the skeleton for the solution architect to start the next phase of the solution refinement: to define the interactions between the components and their precise interfaces so that the IT specialists can start implementing their components.

Now we can return to doing the work. The preliminary step, say call it Step 0, is to collect together requirements and examine the pieces that need to be integrated together to build the external claim assessor solution.

5.2 Step 0: Collating requirements

The solution architect worked with the business analyst in the workshops to gather requirements and define the as-is and to-be claims investigation processes. (see Section 3.2.2, "Responsibilities and contract-based development" on page 52). The outputs from the workshop provide the requirements for the system architect.

Although we did not use Rational Requisite Pro to capture the requirements and integrate them with the architectural model in Rational Software Architect, it is something to consider doing. It is especially useful if tracking requirements is one of the goals of the IT director. The sort of answers that Requisite Pro can provide, if it is used with other change management tools such as ClearQuest are:

► Which requirements are being implemented in which solution?

We can identify who raised a requirement and perhaps involve them, or give them a report on what we are doing about their requirement.

► What is the impact of dropping a capability from a solution?

   a. On other solutions that may depend on it
   b. On the design and implementation of a solution
   c. What parts of the implementation we can now drop?

► If the IT Infrastructure manager decides to modify the upgrade plan, what is the impact on the solutions that are under development?

a. What were the assumptions behind the upgrade plan upon which we decided?

b. Are these assumptions still pertinent?

c. How will they affect the new plan?

We collected the requirements together as a presentation which was reviewed with the executive sponsors and other team members. The following sections summarize the requirements and show how the solution architect can use the integration of WebSphere Business Integration Modeler and Rational Software Architect to build the information that is needed to develop a reference architecture for the design of the solution.

5.2.1 Business goals

See Section 1.2, "Business goals" for a full description of LGI's business goals. Here is a brief summary:

▶ Reduce administration costs by minimizing manual activities involved in managing claims assessment.

  – Automate management of claims assessment.

▶ Increase customer satisfaction by reducing administrative delays on claims queries.

  – Allow monitoring and management of the entire claims process including activities performed by external claim assessors.

▶ Maximize return on development investment.
  – Minimize impact on existing systems
  – Ease of development matching existing skills
  – Fast identification and resolution of business processing delays

5.2.2 Business use cases

There were a number of business use cases identified in the requirements analysis of the claims process. We can use Rational Software Architect to automatically generate a visualization of the uses case derived from the business process model created by the analyst using WebSphere Business Integration Modeler.

This is illustrated in Figure 5-1 on page 157.
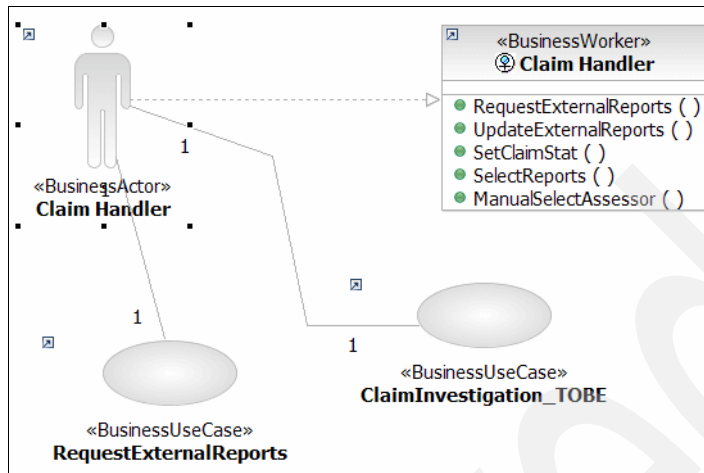
Figure 5-1   External claim assessor use cases

Before looking at how to use Rational Software Architect to construct this diagram, we look at how Rational Software Architect constructs use cases from the business process model.

Mapping the business process model to use cases

The business process model is mapped to a variety of UML2 artifacts. At an abstract level, each activity is represented as a collaboration. There are two activities in the External Claim Assessor solution, which we can show as two collaborations using Rational Software Architect in Figure 5-2 and Figure 5-3 on page 158:
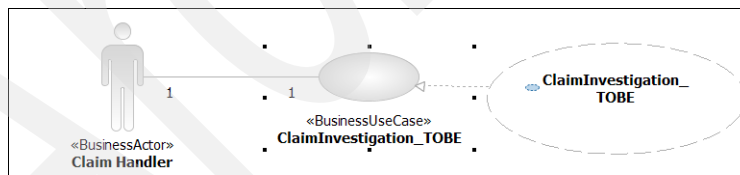
Figure 5-2   ClaimInvestigation_TOBE collaboration

The Claim handler was the only role resource the Business Analyst used in the high level ClaimInvestigation_TOBE process. There were no automatic activities.
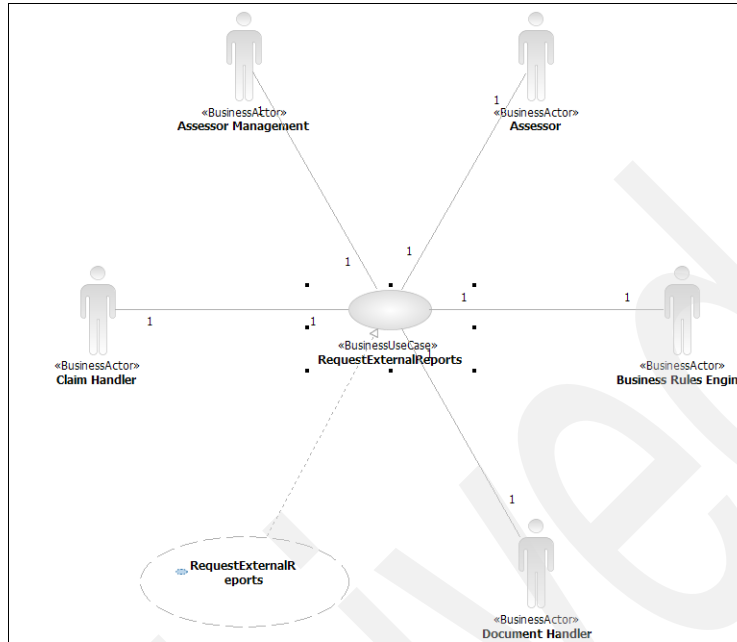
Figure 5-3   RequestExternalReports collaboration

On the other hand, the business analyst identified a number roles played by automatic activities in the RequestExternalReports process in addition to the manual role played by the Claim handler. The analyst regarded the Assessor as an automatic activity.

From these collaborations, there are therefore only two use cases involving a manual player: the Claim handler performing the ClaimInvestigation_TOBE process, and the Claim handler working as part of the RequestExternalReports activity.

Visualizing the use cases in Rational Software Architect

There are several ways to visualize the Claim handler in Rational Software Architect and produce diagrams such as Figure 5-1 on page 157. We look at two ways only here.

To reference to WebSphere Business Integration Modeler model from Rational Software Architect, follow these steps[1]:

1. Import the WebSphere Business Integration Modeler project in to Rational Software Architect. Use either the process you have developed or one of the

---

[1] There is more information about the integration of WebSphere Business Integration Modeler with Rational Software Architect in Appendix B, "Integration considerations" on page 505.

processes supplied in the additional materials for this redbook. Because Rational Software Architect references the Modeler workspace, you first need to have a Modeler workspace available with the project you want to import into Rational Software Architect. Then, import the ITSOLGI project from the Modeler workspace. Suppose you have a workspace called ExternalAssessors, and within it a project called ITSOLGI:

File → Import → Existing WBI Modeler 5.1 project into Workspace → **Next → Browse** to WebSphere Business Integration Modeler workspace → **Select** `.\SG24-6636\Modeler\Workspaces\Pre Bpel\ITSOLGI` → **OK → Finish.**

2. Double-click `resources.xmi` to see the WebSphere Business Integration Modeler model.

> **Important:** Never modify and save the resources.xmi model. It is a read only file. If you save a version of it as a new UML2 model (.emx file) with extensions or modifications, you lose the ability to update dynamically from WebSphere Business Integration Modeler. These instructions enable you to build model extensions and diagrams based on your WebSphere Business Integration Modeler model.
>
> Check that you are not accidently adding elements to the resources.xmi file in the WebSphere Business Integration Modeler model. If the UML model editor tab resources.xmi is starred, you have inadvertently made a change to the model which you will not be able to save. To clean up, close the WebSphere Business Integration Modeler project in Rational Software Architect by **right clicking → Close**. Then reopen it and fix any problems.

– As the first option to produce a use case diagram, simply browse:

  i. In the Model Explorer (see Figure 5-5 on page 161) **Open** `resources.xmi` → **ITSOLGI → RootProcessModel → Claim → Claim_TOBE.**

  ii. Right-click the **Claim handler → Visualize → Explore in Browse diagram.**

– Secondly, create a use case diagram as part of the architect's model. :

  i. Create a new project to store the architect's solution architecture in **File → New Project → UML project → ITSOLGI Architecture**.

  ii. **Select** the **Blank Model template** and *deselect* **Create a default diagram in the new model → Finish.**

  iii. Right-click **Blank Model → Refactor → Rename** and type `Claim Investigation.`

   iv. Right-click **Claim Investigation** → **Add Diagram** → **Use Case Diagram** and name it `Claim Investigation.`

At this point your Model Explorer should look like Figure 5-5 on page 161.

3. Populate the use case diagram to show the relationships you want to document.

 a. Drag the Claim handler from the Claim_TOBE package onto the Claim Investigation package editor.

 b. Right-click the **Claim handler** and select **Filters** → **Show related elements** → **Show All Relationships [Default]** → **OK** (see Figure 5-4).
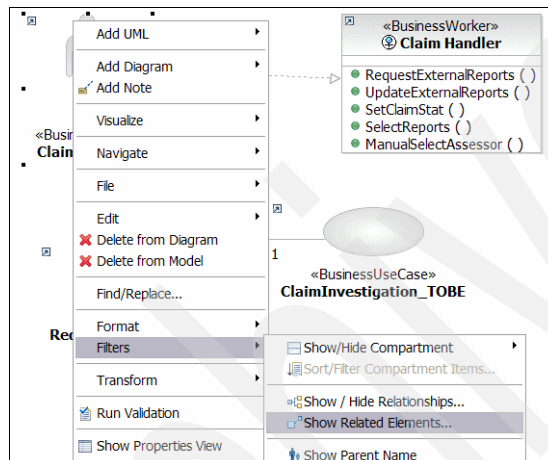
Figure 5-4 Showing the Claim handler's relationships

 c. Tidy up the diagram. A good practice is to have the Claim handler in the top left corner because it is natural to read use cases top to bottom, left to right starting with the role player.
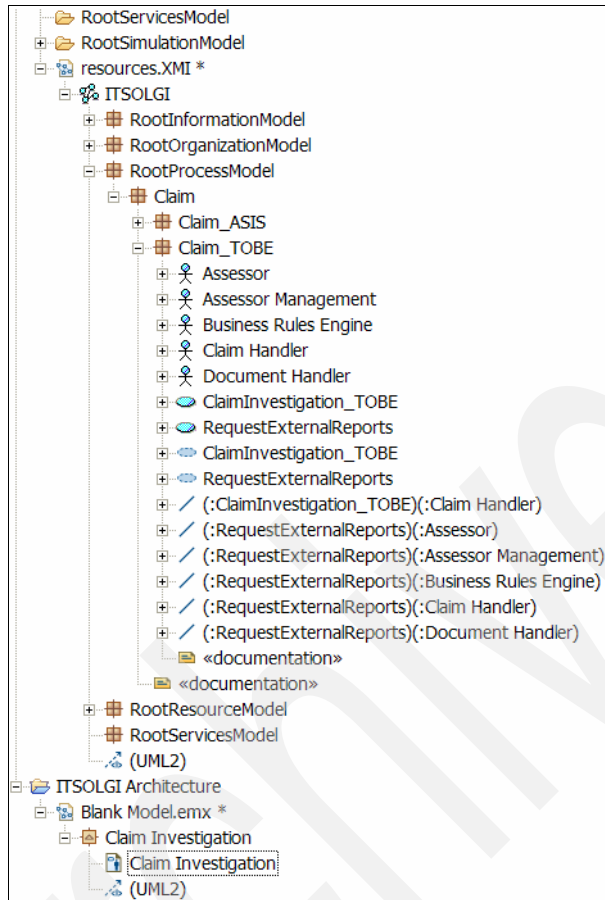
Figure 5-5   ITSOLGI Model Explorer

The use cases themselves are not stored in Rational Software Architect. You can use Rational Requisite Pro to manage the use case documents.

Use Case 1: ClaimInvestigation_TOBE

Role: Claim handler

1. Select policy and check policy coverage.
2. Enter initial customer estimate of damage cost and generate claim reserve.
3. Check previous claim history → Alert → Claims exceeding $30000.
4. **Send externally for detailed Assessment of damage** (see Use Case 2).
5. Check third-party and Assessor report.
6. Negotiate Payment with Customer.
7. Initiate Payment or Repair.

Use Case 2: RequestExternalReports

Role: Claim handler

1. Claim handler logs onto Business Process Manager.
2. Selects claim awaiting assessment.
3. Initiates automated assessment process.
4. Identifies suitable assessor by post code and vehicle type.
5. Send Requests for availability.
6. Await confirmation requests.
7. Select assessor.
8. Request Assessment.
9. Wait to Receive assessment report.
10. Return assessment report back to Claim handler.

5.2.3 Roles

See 1.4, "Roles" on page 10 for a full description of all the roles. Here is a brief summary:

► LGI roles

– Claim handler: Manages claims
– Claims Supervisor: Has discretion to deal with exceptional claims
– Claims Analyst: Monitors overall claims process for cost performance

► External roles

– Assessor: Assessing Claim

You can visualize how these roles are involved in the claims process model by using the Model Explorer. To draw a diagram showing the roles in more detail in Rational Software Architect, create a freeform diagram in the ITSOLGI Architecture project. Call it `Manual Roles` and drag the selected roles from the WebSphere Business Integration Modeler model (in **Model Explorer** → **ITSOLGI** → **RootResourceModel** → **Resources** → **Roles**). Use Ctrl-click to select multiple roles and left-click to drag the roles onto the diagram as in Figure 5-6 on page 163. This diagram not only shows some of the roles we defined, but because the Business Analyst identified what operations the roles performed in, we also have a list of operations in which each role is involved. The roles are conceptualized as BusinessWorkers by the WebSphere Business Integration Modeler integration. If you look at the properties of one these roles you can see it is a stereotype of an interface.
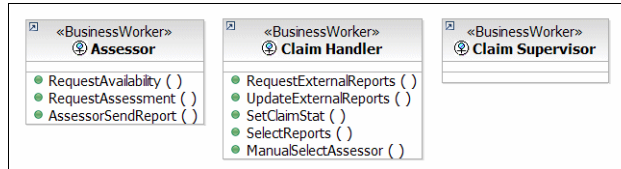
Figure 5-6   Roles and the operations in ClaimInvestigation_TOBE

Some of the roles in the original requirements statement were not modeled (such as Claim Analyst) and are missing. There are other roles that we used to model automatic activities which we have omitted here.

5.2.4 Components

The process model created by the business analyst defines the activities, roles and resources, including both manual and automatic activities. The solution architect is responsible for defining the services and interfaces to be used to automate the process. The architect can use the interfaces derived from the roles responsible for the automated activities in the process model to define the components that are required in the solution.

To create the new components in the model, follow these steps:

1. Open the **ITSO Architecture** project and create a new `Claim Investigation Component Diagram`.

2. Drag the **Assessor, Assessor Management, Business Rules Engine** and **Document Handler** onto the component diagram.

3. Now create six new components called `AssessorManagement`, `BusinessRulesEngine`, `AssessorSystem`, `ClaimsSystem`, `AssessorAutomation` and `ClaimsWorkflow`. You can do this on the diagram directly by right-click → **Add UML** → **Component**.

4. Wire up each interface to its corresponding component by creating an implementation relationship. **Hover** over a component and click the **small arrow** pointing at a square box. **Drag** it to its corresponding interface and **release**. Select the **Create New Implementation**. Wire up the AssessorAutomation and ClaimsWorkflow components as shown in Figure 5-7 on page 164. We can color the components to match the Pattern for e-business scheme and show the Assessor System is part of a different organization.
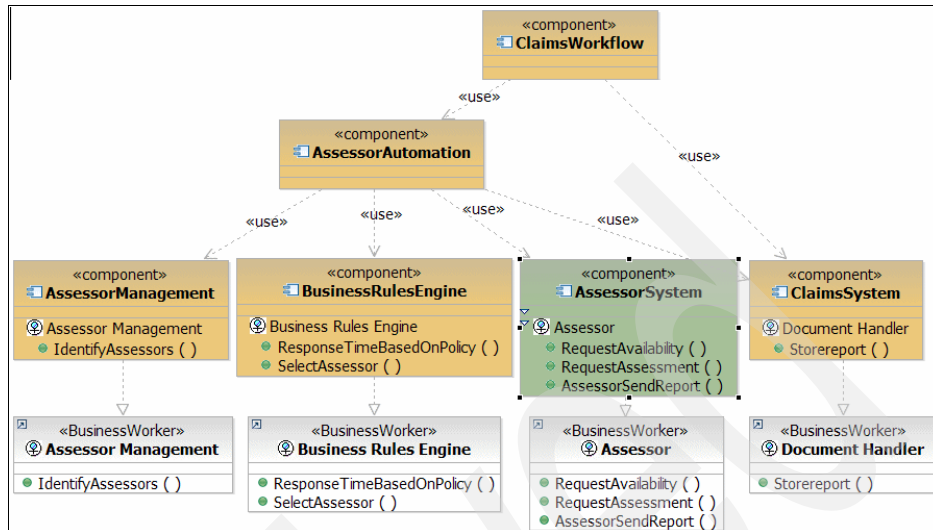
Figure 5-7   Claims Investigation component diagram

When we did the Patter for e-business selection, described later in this chapter, we discovered we needed to add another component to this diagram.

The AssessorSystem is not part of LGI and needs a proxy component we called the *proxyAssessorSystem* to connect to the AssessorSystem which will mediate between LGI and the assessors. We give this new component a different interface from that exposed by the Assessor System. The interfaces cannot be the same, so it is not a proxy in the sense often understood, It cannot be generated automatically from the real AssessorSystem interface. For example, the Request Availability activity asks for the availability of all the eligible assessors, while each assessor receives an individual request for their availability. Of course, the requests to the assessors go to different destinations over different transport protocols.

We add the proxyAssessorSystem and other details from the completed architecture to the diagrams. Although the component architecture was not constructed until we did the runtime mappings step of the e-business patterns method described in 5.5, "Step 3: Select and merge the runtime patterns" on page 176.

It is unlikely the architect would arrive at the final component model that gets implemented at this early requirements gathering stage. Nonetheless, understanding what components the solution is going to need is a useful way of testing whether we really have sufficiently understood the scope of the problem, and whether we need to probe for more requirements.

One of the virtues of Rational Software Architect is that it makes it easy to create and modify models that can be analyzed and criticized, until one arrives at a picture that all members of the team will agree upon.

To complete the component model we followed these steps:

1. The manual tasks that are the responsibility of the Claim handler have been added and divided between the claims workflow and automated assessor process engines.

2. Both the provided and required interfaces have been added to the components.

The completed component model diagram is illustrated in Figure 5-8.
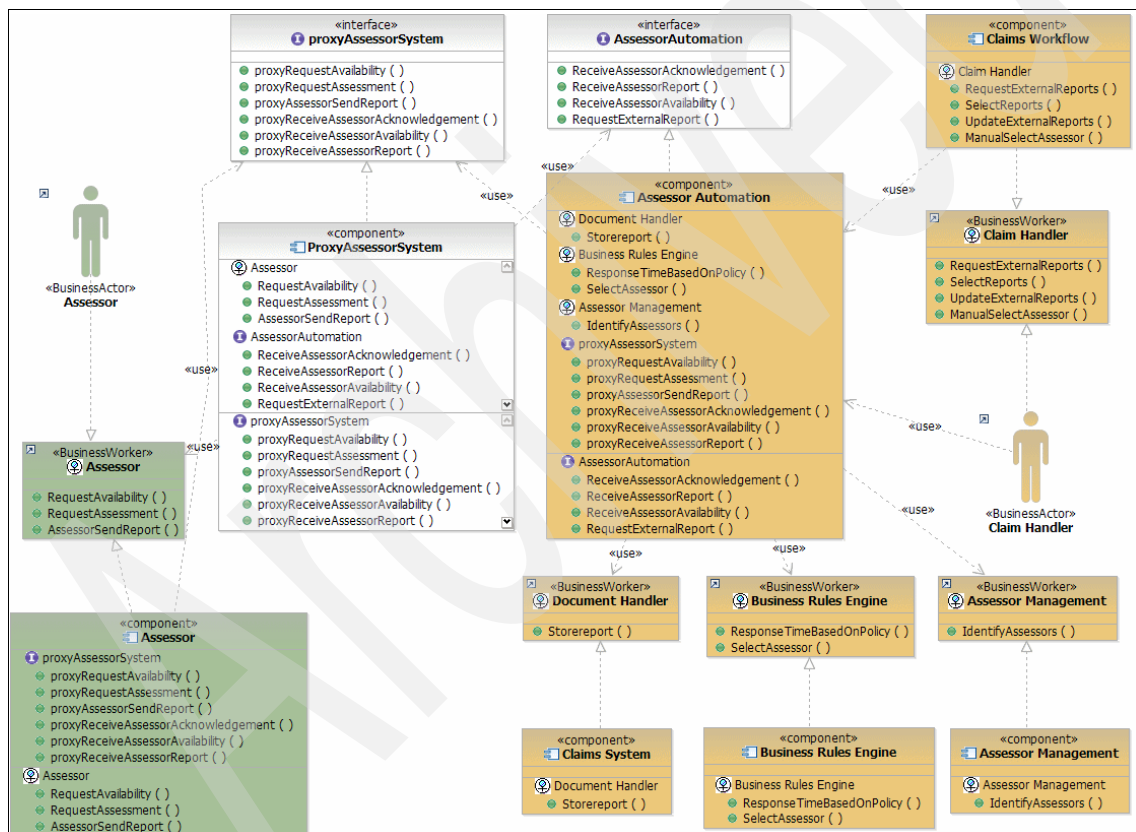
Figure 5-8   Completed component diagram for ClaimInvestigation_TOBE

The interactions and interfaces of the components will be refined when we look at interactions in Chapter 6, "Solution Architecture" on page 187. At this stage of the architectural refinement, the systems architect needs to briefly describe what

each component does in order to make the best decisions about what architectural patterns to employ.

Existing Components

The existing components are to be modified as little as possible in producing he new solution.

LGI Assessor Management System

This system provides details of each assessor and how they communicate with LGI.

It has the following operations:

► Identify Assessor

Given a vehicle type and post code of the location of the vehicle to be assessed, Identify assessor returns a list of potential assessors and how to get in contact with them

► Load/Register Assessor[2]

► Ranking Assessor[2]

► Record Selected Assessor[2]

Stores the selected assessor and returns the Claim id and Assessor details.

Claims Workflow

This the existing manual process. The sub-task we are interested in is the claim investigation. A Claim handler selects a claim to work on, gets the external reports and finishes by setting the claim status preparatory to judging the claim. The claim investigation has the following activities:

► Select Report[2]

The Claim handler selects a claim to work on from their worklist.

► RequestExternalReports

This is now the automated process which calls the automated assessor subprocess to perform a claim investigation and return the assessors report

► Update External Reports[2]

The Claim handler may gather some other external reports

► SetClaimStatus[2]

Finally sets the claim status as ready for the claim to be judged.

---

[2] These operations have not been included in this scenario.

Document Management System

The document management system holds all the reports that are generated in the claims and policy process. The only operation we are interested in is:

► Store the assessment report

New and changed components

The Enterprise service bus is built from existing WebSphere MQSeries and Web Services Gateway components to provide a service based transport to the components in the solution.

The Business Rules Engine (BRE) is a new component. We have not focused on the uses and appropriate technology for the BRE. The goal for this component is to improve customer satisfaction with the claims process by better managing the relationship with the external claim assessor.

Enterprise Service Bus

The Enterprise Service Bus provides a means to communicate safely and securely with assessors in a variety of ways to suit their connectivity capabilities. Communications are loosely coupled and can be initiated at either end.

► ProxyRequestAvailability

Given a list of assessors and claim details send a request to each assessor available to do the assessment asking for their availability. The request has to be sent using the communication method agreed with the assessor (e-mail, EDI, Web service, Browser page, and so on).

► ProxyRequest Assessment

Given the chosen assessor and the claim details request, an assessment report store an acknowledgement to the request.

► ProxyAssessorSendReport

Receive the assessors report and store it.

Business Rules Engine

This is a new component to allow customization of the assessment process depending on business goals. Its operations are:

► Choose a response time for an assessment, based on the type of policy a client holds.

► Apply business rules to select an assessor from a list of assessors

5.2.5 Organization and architectural constraints

See 1.3, "IT goals and constraints" on page 9 for a full description of IT goals and constraints. We briefly, summarize them here:

► IT Policies

  – Use open standards.
  – Maintain existing channels.
  – Build common infrastructure for LGI and DirectCar.
  – Reuse existing applications.
  – House the new infrastructure with LGI.

► Corporate Infrastructure constraints

  – New applications must use the corporate LDAP directory for staffing roles and definitions

► Be able to perform manual assessments to deal with special cases.

► Support different assessors' systems, including browser access, e-mail, EDI and Web services over http: and JMS.

5.2.6 Limitations

There are a number solution considerations we omitted because of time limitations.

► Data Modeling

  We have simplified the use of data in the solution. In practice data modeling, using an industry standard data model such as ACCORD from the insurance industry, is a very significant part of the business analyst's effort and creates new challenges for the whole development team.

► Security

  We omitted security considerations throughout the solution.

► Directory

  We did not implement a solution wide staffing directory.

► User Interface

  We did not focus on the user interface. However there was an integration issue: the integration of worklists from different process engines.

  WebSphere Business Integration Server Foundation do not share a worklists, and without further work, the Claim handler would need to use two windows to handle both worklists. Also, the Claim handler would need to work out the correspondence between the lists.

Both process engines have a worklist API, and one style of integration would be to write a worklist integration component to present a merged worklist to the user interface.

An alternative style which is more economical, but not as satisfactory as a full integration, is to use the built-in portal interfaces to both the Workflow and Choreographer clients, to display both worklists in the same window.

► We did not implement any business event monitoring to track the progress of insurance claims.

► We did not implement any system management for deployment or monitoring.

We have limited the style of connection to the external assessor to SOAP/Http:. In practice, the products used for the ESB gateway would need to cater for at least EDI, e-mail and Browser access.

5.3 Step 1: Select a Business Integration Pattern

Business/ Integration Patterns

Now that the architect has organized the requirements and understood the nature and scope of the solution, The first step in building the Patterns for e-business approach is to select a Business Integration pattern. The choice comes down to two patterns to consider:

1. Extended Enterprise

   The criteria for the Extended Enterprise pattern are,

   – *The business process needs to be integrated with existing business systems and information.*

   – *The business processes need to integrate with processes and information that exist at partner organizations.*

   For further information, see:

   `http://www-106.ibm.com/developerworks/patterns/b2bi/info.html`

2. Application Integration

   The criteria for the Application integration pattern are

   – *The business process needs to be integrated with existing business systems and information.*

   – *The business processes need to integrate with processes and information that exist at partner organizations.*

   – *The business activity has a need to aggregate, organize, and present information from various sources within and outside of the organization*

   For further information, see:

You can see there is overlap between the two patterns in the italicized bullets in Figure 5-9 and Figure 5-10 on page 171. Both patterns are appropriate. Should we continue with just one, the other, or both? There is some more guidance on the Patterns for e-business web site. Again using italics to highlight the overlaps.

- ► *Business Entities, which typically:*
  - – *Are programs, applications or databases that exist within an organization*
  - – *Access and connect to other Business entities across the network*
- ► *A Network which:*
  - – *Is based on TCP/IP and other Internet technologies*
  - – *Can be a dedicated Wide Area Network (WAN) connection*
- ► Business Rules that:
  - – Manage the integration between the Business entities
  - – Describe Trading Partner® Agreements
  - – Use Workflow rules to determine the sequence of steps and the data flow that need to be used to facilitate the integration. *
    These rules:
    - • Describe the sequence of steps that a message needs to go through before being transferred to the other business entity and
    - • Specify how and where the message should be delivered
  - – Use Transformation Rules to specify format and protocol transformations that need to be applied to messages that flow between the business entities
- ► A set of interactions that include the execution of a jointly-agreed business process

---

Figure 5-9   Selection of Extended Entity pattern elements

> ► *Business applications and data that need to communicate, interact and integrate with other business applications and data that exists within the organization or in business partner organizations*
> ► *A network, which*
>   – *Is based on TCP/IP and other Internet technologies*
>   – *Can be a dedicated LAN connection or WAN connection*
> ► Other business applications and data which can be:
>   – Custom developed systems (old and new)
>   – Enterprise Resource Planning systems and other Packaged applications such as SAP, BAAN and PeopleSoft
>   – Databases
> ► Application Integration services that include:
>   – Protocol adapters
>   – Message handlers
>   – Data transformation
>   – Decomposition/Re-composition
>   – Routing/Navigation
>   – State management
>   – Security
>   – Local business logic
>   – (Business) unit-of-work management

Figure 5-10   Selection of Application Integration pattern elements

Rather than pick one of these patterns at this stage, we continue with both and resolve any overlaps when we do the runtime mapping in 5.5, "Step 3: Select and merge the runtime patterns" on page 176.

5.4 Step 2: Select the application pattern

Application Patterns

We have two business integration patterns to refine, the Extended Enterprise and Application Integration patterns. The Extended Enterprise pattern will be most useful in designing the architecture for interacting with the assessors. The Application Integration pattern will be useful integrating the new external claim assessor solution with the existing claims workflow, assessor management, and claim systems.

5.4.1 Collaborations

At some point, it is useful to draw the collaborations between the solution components we have identified. This helps in the analysis of what application integration patterns to choose, and then how to map the solution components to the components in the runtime mapping. Choosing the application pattern,

deciding where different business integration patterns overlap, and mapping the solution components to the runtime components, calls for discussion and judgement. To promote discussion, the work is best done in a workshop with some of the participants we identified in Section 3.2.1, "Roles and responsibilities" on page 46. The understanding achieved between the participants is as important as the answer that emerges.

We use Patterns for e-business style for drawing the collaborations of the existing and new claim system use the to-be diagram.

Referring back to Figure 2-6 on page 21, the as-is assessment process is shown in Figure 5-11.
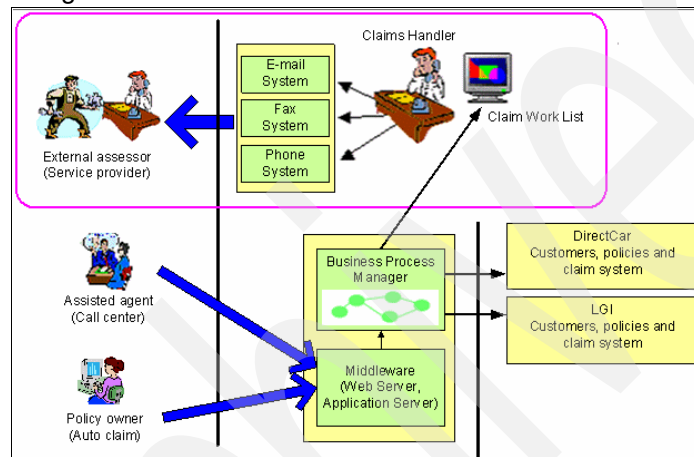
Figure 5-11   Manually requesting assessments reports from external assessors

In the [P4EB] style, leaving out the applications upstream of the claim system, the as-is system becomes Figure 5-12

:
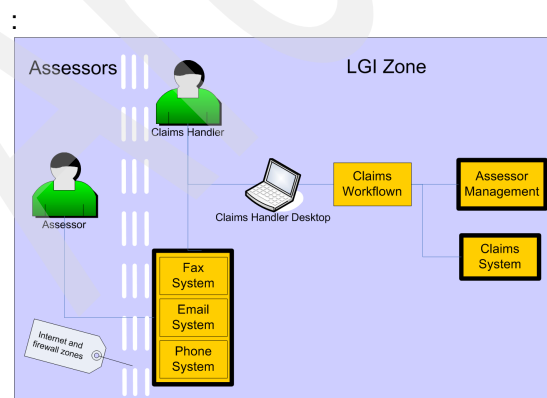
Figure 5-12   [P4EB] ClaimInvestigation_ASIS collaboration

The vertical dashed lines denote domains or governance areas. Applications are shown as square boxes, and those which cannot be altered are given heavy borders.

Figure 5-13 shows the Extended Enterprise and Application Integration patterns superimposed on the to-be system.
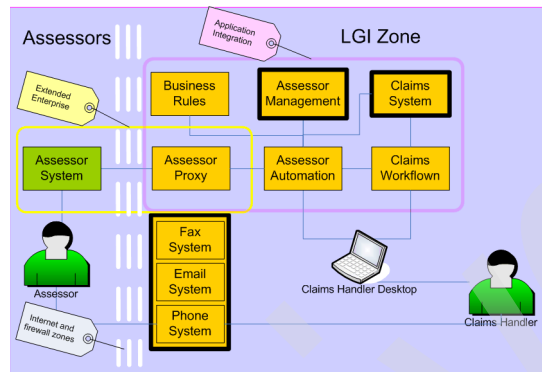
Figure 5-13   [P4EB] ClaimInvestigation_TOBE collaboration

We now need to choose two application patterns, one for the Extended Enterprise business integration pattern and the other for the Application Integration business integration pattern.

5.4.2 Application Patterns for the Extended Enterprise

The focus of the Extended Enterprise part of the solution is to communicate with the external assessors. Figure 5-14 on page 174 shows the business drivers from which we have chosen the Exposed Broker application pattern. Notice the outlines around that column.

Business drivers	Exposed Direct Connection= Message Conection	Exposed Direct Connection= Call Connection	Exposed Router variation	Exposed Broker	Exposed Serial Process	Exposed Serial Workflow variation
Improve organizational efficiency	✓	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓	✓
Support a structured exchange with business partners	✓	✓	✓	✓	✓	✓
Support real-time one-way "message" flows to partner processes	✓		✓	✓	✓	✓
Support real-time request/reply "message" flows to partner processes		✓	✓	✓	✓	✓
Support dynamic routing of "message" between partners to one of many target applications			✓	✓	✓	✓
Support dynamic distribution of "message" between partners to multiple target applications				✓	✓	✓
Support automated coordination of business process flow between partners					✓	✓
Support human interaction and intervention within the process flow between partners						✓

Figure 5-14   Selecting the exposed broker application pattern[3].

The Exposed broker application pattern is illustrated in Figure 5-15.
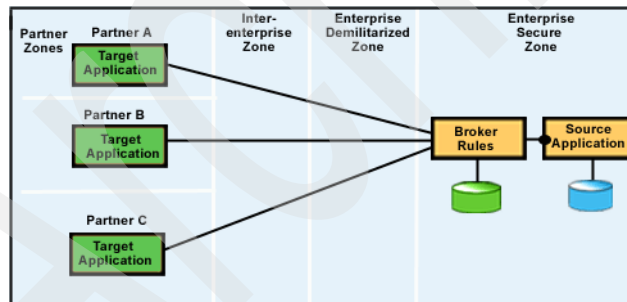
Figure 5-15   Exposed Broker application pattern

The key drivers behind the choice of this pattern are:

1. Process management will be performed in the application integration pattern
2. Dynamic distribution of messages to send messages to multiple assessors

---

[3] This chart is taken from
http://www-106.ibm.com/developerworks/patterns/b2bi/select-application-topology.html.
There is also a set of IT drivers which reinforces the decision to go for an exposed broker pattern.

3. Need to recompose replies solicited from multiple assessors back into a list of possible assessors to perform the accident assessment
4. One-way and two-way message flows

Quoting from the Patterns for e-business Web site, this pattern fits our needs very well:

> The primary business driver for selecting this application pattern allows one application to interact with one or more of multiple partner applications across organization boundaries. Using a hub-and-spoke architecture instead of a point-to-point architecture allows for the seamless integration of applications while minimizing the complexity. A request for information can be routed to one of many targets or simultaneously to multiple targets. The resulting request message can be decomposed into multiple request messages, and the reply messages then recomposed into a single reply message using appropriate recomposition rules.
>
> This externalization of routing, decomposition, and recomposition rules from individual source and target applications increases the maintainability and flexibility and reduces the enterprise wide integration complexity.
>
> The primary IT driver for selecting this application pattern allows loose coupling of clients and services with minimum modification to each. The solution should allow for multiple transmission protocols to be used and for transformation of protocols between client and service.

5.4.3 Application patterns for Application Integration

Looking at the business drivers for the application integration patterns in Figure 5-17 it is clear that to support human interaction we need to choose the Parallel Workflow variation on the Parallel process pattern.
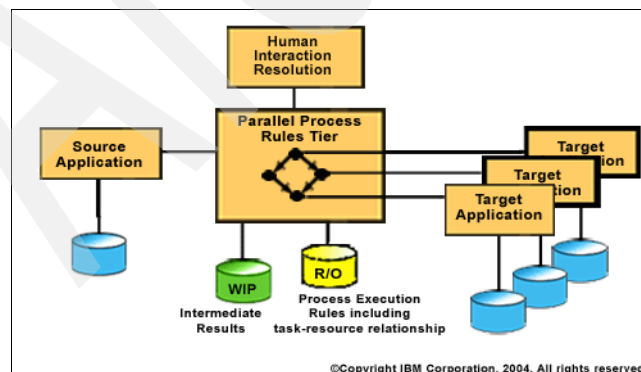
Figure 5-16   Parallel Workflow variation application integration pattern

Business drivers	Direct Connection= Message Connection	Direct Connection= Call Connection	Router variation	Broker	Serial Process	Serial Workflow variation	Parallel Process	Parallel Workflow variation
Improve organizational efficiency	✓	✓	✓	✓	✓	✓	✓	✓
Reduce the latency of business events	✓	✓	✓	✓	✓	✓	✓	✓
Support a structured exchange within the organization	✓	✓	✓	✓	✓	✓	✓	✓
Support real-time one-way "message" flows	✓		✓	✓	✓	✓	✓	✓
Support real-time request/reply "message" flows		✓	✓	✓	✓	✓	✓	✓
Support dynamic routing of "messages" to one of many target applications			✓	✓	✓	✓	✓	✓
Support dynamic distribution of "messages" to multiple target applications				✓	✓	✓	✓	✓
Support automated coordination of business process flow					✓	✓	✓	✓
Reduce cycle time through parallel execution of portions of a process flow							✓	✓
Support human interaction and intervention within the process flow						✓		✓

Figure 5-17   Selecting the Parallel workflow variation application pattern

The pattern is illustrated in Figure 5-16 on page 175.

The key criteria for choosing this pattern are:

1. Support human interaction.
2. Run parallel executions of portions of the process flow to speed execution.
3. Support long running processes with transactional recoverability.

> **Note:** Many of the other criteria overlap with the exposed broker pattern we have chosen to use to connect LGI to the assessors, so we can expect that there will be cases where we can implement some function in either one or the other of the components introduced by these design patterns.

5.5 Step 3: Select and merge the runtime patterns

Runtime Patterns

In this section we describe the runtime patterns for the two application patterns we have chosen, merge them, and map the application components onto the runtime components.

The IT strategy directs us to go down an open standards approach, and we have decided to adopt a Service Oriented Architect (SOA) runtime. Each of the

Exposed Broker and Parallel Workflow application patterns has an SOA runtime variation as illustrated in Figure 5-18 and Figure 5-19.
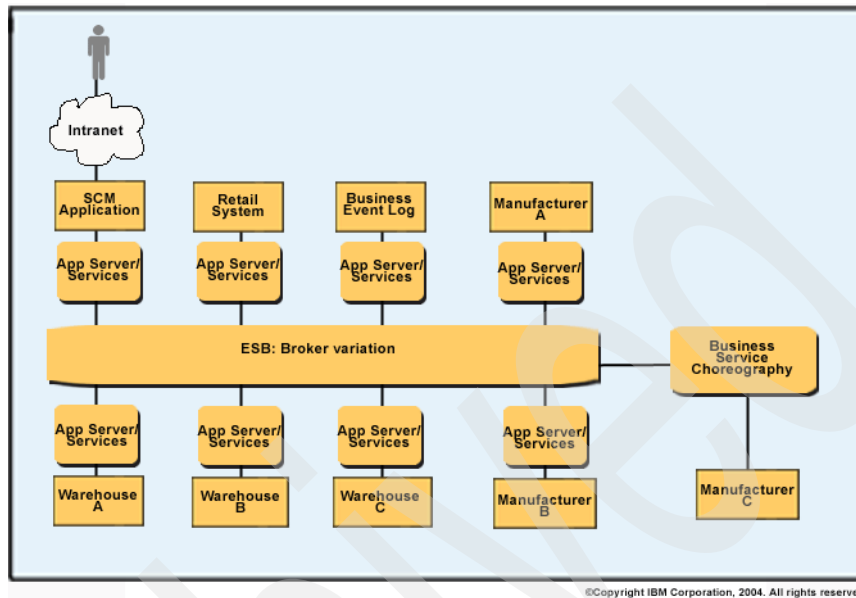
Figure 5-18   [SOA] Application Integration::Parallel Workflow variation::Runtime pattern
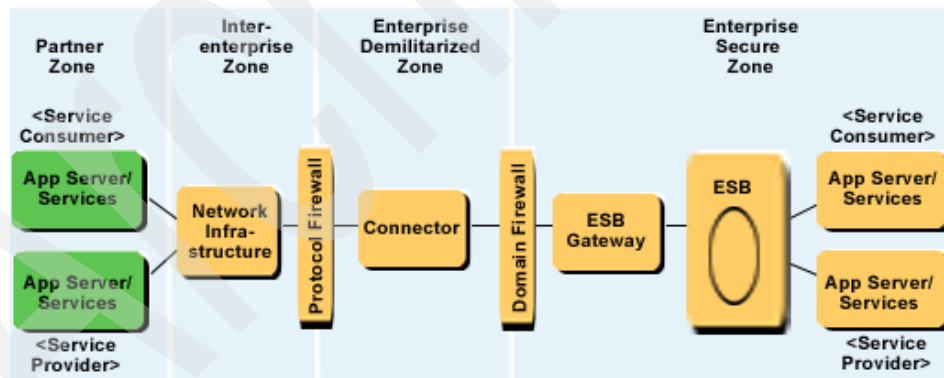
Figure 5-19   [SOA]Extended Enterprise::Exposed Broker::Runtime pattern

5.5.1 Proposal 1: Broker focussed integration pattern

Our first architectural proposal combines these two runtime patterns using the broker to connect all the components together. This is shown in Figure 5-20 on page 178 below. To avoid clutter, we are drawing only one assessor and Claim

handler in these diagrams. Clearly, there are many assessors and Claim handlers.

We have deployed the new assessor automation system on a new process engine, and minimized changes to the existing Claims Workflow engine. The services we require are all running in an application server environment. The services that manage the assessors (the Assessor Proxy component) are hosted in the ESB gateway and all the components are connected by a common service bus that connects the two integration patterns together.
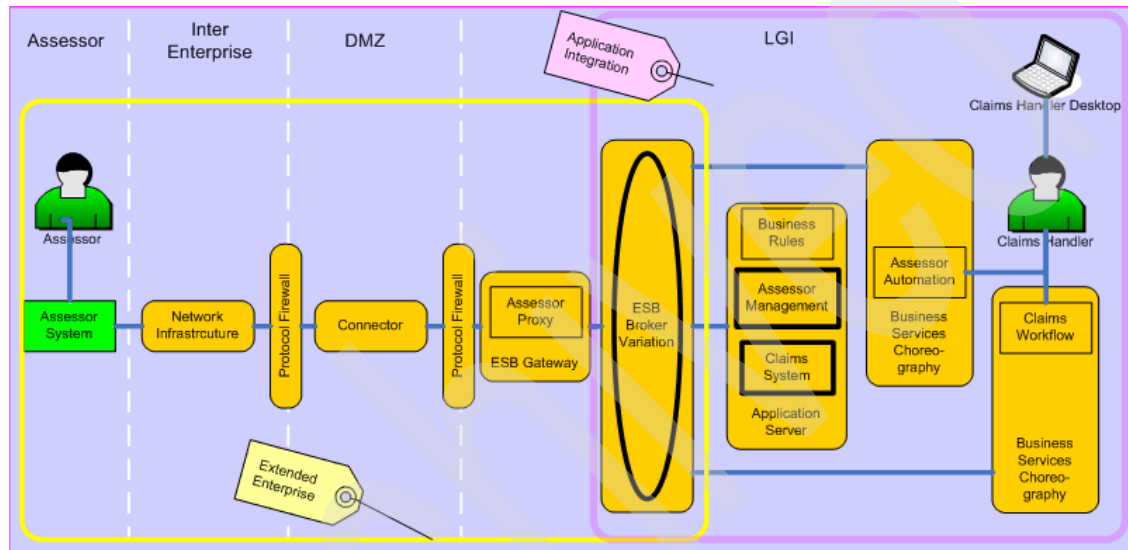
Figure 5-20   Broker focussed combination of application integration patterns

This architecture has a number of advantages when you consider the skills and infrastructure LGI already has:

► LGI are very familiar with the implementation of a message bus and have the skills to evolve their message bus into to a service bus.

► Having all the services connected through the ESB is appealing from the perspective of manageability, governance and reusability. The ESB is a cross enterprise resource. Connecting services directly to the applications that first require them may lead to duplication and lack of control.

► The technology used to implement the hub of the ESB (WebSphere Business Integration Message Broker) has proven very effective at overcoming integration problems revealed when connecting different components together. Issues like incompatible protocols, different levels of application data structure, and so on, can be overcome by writing message flows, or *mediations*, in the broker.

► The loosely coupled style of connection implemented by the ESB has proven valuable both in decoupling development teams.

This solution met some criticism from the team concerning the amount of additional effort required to implement routing the connections from the Assessor Automation process engine through the ESB to its services and then back to the Assessor Automation process engine. With LGI's ESB running primarily on WebSphere Business Integration Message Broker there is no common tool to make these connections. Whereas if a direct connection model is used between the Assessor Automation process engine and those of its services that are running as EJBs, then the WebSphere Studio Application Development Integration Edition tool will generate the service connections as well as implement the process engine.

5.5.2 Proposal 2: Process focused integration pattern

We changed the runtime pattern to that illustrated in Figure 5-21, which has direct point to point connections with the process engine

.
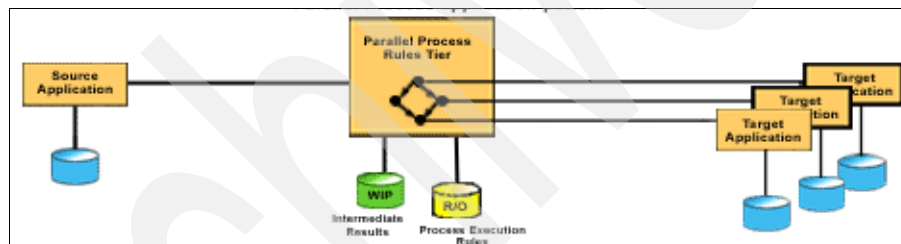
Figure 5-21   Parallel Process application pattern::Runtime pattern

Applying this pattern led to the second proposal shown in Figure 5-22 on page 180. It uses direct point-to-point connections between the Assessor Automation system and the services it requires. It can use Web services for these connections, but the services are addressed directly rather than through a bus.
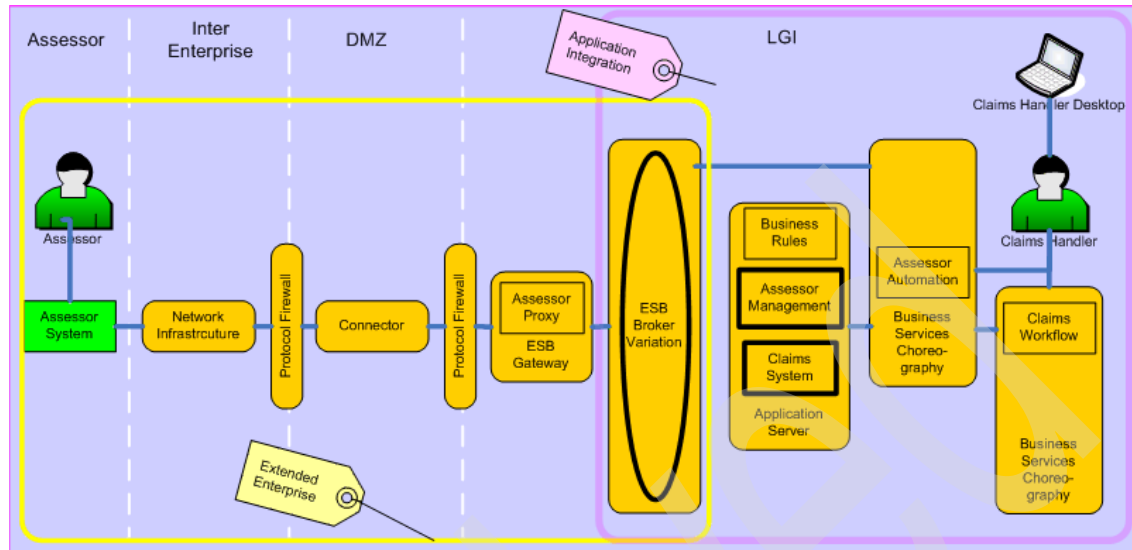
Figure 5-22   Process focussed combination of application integration patterns

There are pros and cons of both approaches. Which one we choose will be strongly influenced by practical considerations such as the current skills, current IT infrastructure architecture, and the amount of change involved for either approach. As discussed in our case, a major influence on the choice was the tooling to build the solution that lead us to choose the process focused solution. A further factor was there is a supportpac to connect the WebSphere MQ Workflow server that is running the claims workflow with the WebSphere Business Integration Server Foundation server that will be running the Assessor Automation system which should make that integration relatively easy.

5.6 Step 4: Apply product mappings

After reviewing Runtime patterns, we now map the logical nodes defined in the runtime pattern to specific products which implement the runtime solution design on a selected platform. The product mapping identifies the platform, software product name, and version numbers as well. Consider the following issues when deciding on a platform to host your e-business application:

5.6.1 Existing systems and platform investments

Figure 2-12 on page 25 shows the complete existing IT infrastructure and tools used to build the current claim system. The significant highlights are:

► The merged claims workflow runs on WebSphere MQ Workflow and DirectCar has a WebSphere Application Server Enterprise process engine.

► The ESB runs on WebSphere MQSeries and WebSphere Business Integration Message Broker with Web Services Gateway to support two existing Web services connections to external suppliers.

► The Web front end and some existing claims applications run as EJBs on WebSphere Application Server.

5.6.2 Available customer and developer skills

LGI have considerable WebSphere MQSeries and WebSphere Application Server skills. They have built point to point Web service applications and EJB applications. They have built business processes using WebSphere Business Integration Modeler 4.3.4 (the existing Holosofx product acquired by IBM) and run process on WebSphere MQ Workflow.

5.6.3 Customer choice

LGI wants to develop more applications around open standards such as Java 2 Enterprise Edition (J2EE), Web services and BPEL. In particular, they want to use this project to assess the tools and run time for a model-driven development approach using UML and BPEL. See 1.3, "IT goals and constraints" on page 9.

► The platform chosen should fit into the customer's environment and ensure quality of service, such as scalability and reliability, so that the solution can grow along with the e-business.

– The claims handling application does not demand such high performance as the policy application, which is responsible for winning new business. The business analyst has performed some simulations that can be used to estimate the size and numbers of servers that are required after some performance measurements have been done.

– LGI has established a reliable messaging backbone for their ESB, and want to use it for asynchronous flows to improve reliability. They want to use asynchronous interactions where responses are delayed and where connections are made outside their IT infrastructure to improve availability and to reduce the coupling of LGI processes on the availability of assessors applications.

5.6.4 Product Mappings

Figure 5-23 summarizes the product mapping we chose.
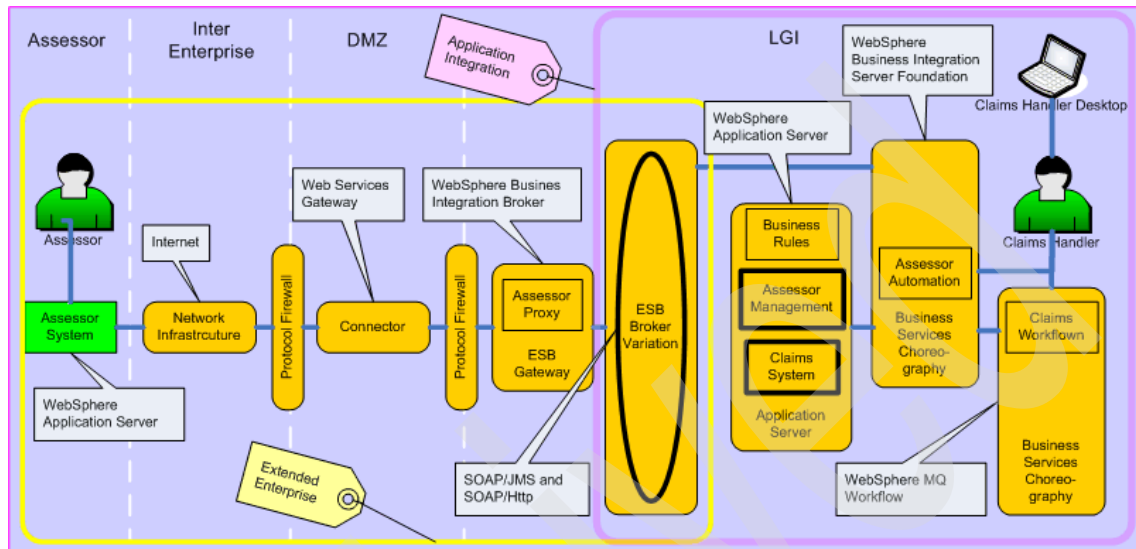
Figure 5-23   External Claim Assessor::Product Mapping

Assessor Systems

We will implement one Assessor system as an EJB running on WebSphere
Application Server. The assessors responsibilities are:

► Responding to a request for availability
► Responding to a request to perform an a claims assessment
► Sending a completed claims assessment

Business Rules Engine

The Business Rules Engine is an EJB running on WebSphere Application Server
and is responsible for:

► Providing the response time required for a claim
► Selecting the assessor to process a claim from a list of available assessors

Assessor Management

The Assessor Management system is an EJB running on WebSphere
Application Server. Its responsibility is to:

► Provide a list of assessors that could potentially perform an assessment
  based on vehicle type and location.

Claim System

The claim systems is an EJB running on WebSphere Application Server. It is responsible for:

► Storing a claim assessment

Assessor Automation

The Assessor Automation system is a BPEL engine running on WebSphere Business Integration Server Foundation. It is responsible for:

► Receiving a claims assessment report request from the claims workflow engine

► Managing the process of creating the report

► Returning the report to the claims workflow engine.

Claims Workflow

The Claims Workflow system is a FDL engine running on WebSphere MQ Workflow. It is responsible for:

► Presenting the Claim handler with:

  – A list of claims to be investigated

  – A list of claims of claims that have now received an assessment

► Sending a claim to the Automated Assessment process and waiting for the assessment to be completed

ESB

The ESB is a service bus supporting MQ, SOAP/JMS and SOAP/Http: It is responsible for:

► Isolating service requests from the transport protocol and physical addresses of endpoints

ESB Gateway

The ESB gateway are services used by the service bus. It is responsible for

► Implementing the appropriate communication style for each assessor (SOAP/Http Web services, EDI, Browser, and so forth)

► Distributing requests to multiple assessors

► Aggregating responses from multiple brokers

► Setting timeouts on response times

► Handling late replies

Web services Gateway

The Web Services Gateway is in the DMZ. It is responsible for:

► Proxying Web service addresses between the intranet and internet

► Responding to automated requests for supplying WSDL interfaces to Web service clients

► Securing communications across the internet

5.7 Reference architecture

Figure 5-24 shows the physical reference architecture we will use in the scenario. The deployment is modelled in Rational Software Architect using component model we have already defined.

1. In the Model Explorer select **ITSO Architecture** → **Claim Investigation** → **Right click** → **Add Diagram** → **New Deployment Diagram**.

2. Drag the components onto the diagram (omit the assessors and the Web Services Gateway for brevity) select them all and click **Name Compartment Style** to show only the component names and not all their attributes. See Figure 5-24.
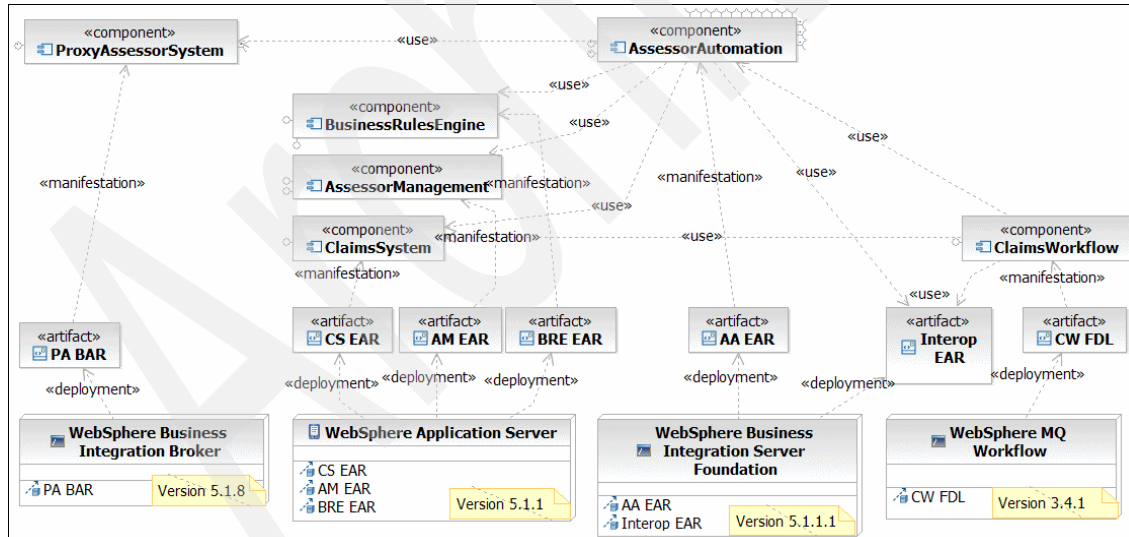
Figure 5-24   Claim Investigation deployment diagram

► Add **Artifacts** for all the Broker Archive Repository (BAR), Enterprise Archive Repository (EAR) and FDL flows that will be deployed. We will also add an artifact to represent the Supportpac W0AD that will be used to connect

WebSphere MQ Workflow to WebSphere Business Integration Server Foundation.

► Add **Devices** or **Execution Environments** to represent the middleware servers that will host these artifacts.

5.7.1 Omissions from the reference architecture

There are things we chose not to model at this stage. We did not go into any more details about the deployment, such as the physical machines or the network addresses, nor have we modelled the transport layer or the tools we use to create the artifacts. All these things could be added at this stage. Whether to do so or not depends on uses to which the model is going to be put and whether there are any architectural decisions to make. For example, we have chosen not to model the robust messaging transport or to model the SOAP/Http: layer. There is nothing nonstandard about these parts to consider, and we are not planning to built an exact deployment model to drive the configuration of deployment artifacts. In other situations, these might be important parts of the solution to model.

For our purposes, we need to focus on what artifacts are to be created for which runtimes. So we have refined the model sufficiently to identify which runtimes we will be using, which tools are required, and what are the collaborations we will be using for interactions and interfaces between components.

5.8 Summary

This chapter has looked at three steps in producing the system architecture:

1. Pull together the requirements for the IT architecture into Rational Software Architect. The requirements came from the Business Analysts process model, the original business and IT goals, and from the workshop which was used to create the business process model.

2. Analyze the requirements to build the reference architecture using the Patterns for e-business Integration Design Approach.

3. Returning to Rational Software Architect to capture the reference architecture as a deployment diagram.

The next step is to study the details of the process and build the solution architecture based on the reference architecture, the interactions, and the interfaces to the components.

6

Solution Architecture

In the previous chapter we followed the Patterns for e-business methodology to create a system or reference architecture for the software infrastructure based on the requirements of the business process. In this chapter, we analyze the business process's behavior to create a solution architecture.

The combination of the reference and solution architecture is necessary for the IT specialists to begin to implement the solution.

From the reference architecture we know:

► What components to implement
► What platform to use to implement them

The solution architecture will define

► The interfaces that need to be implemented
► The interactions and the technologies to implement them
► The qualities of service required from the components and interactions

Producing the final architectural specification is an iterative process. It might turn out be true that in analyzing the interactions we find the platforms chosen for the reference architecture do not support the qualities of service such as performance or robustness that we require. In such a case, the design team will need to look for alternative solutions which can even involve changes to the business process.

6.1 Interaction Model

The bases for the interaction model are twofold:

► The collaboration pattern we have selected in the reference architecture
► The business process

From these we can draw out the rough solution flow in Figure 6-1. It shows the interactions derived from the Claims Investigation business processes mapped onto the collaboration pattern from Figure 5-22 on page 180. The collaboration patterns has been simplified and turned to read from left to right . The picture is beginning to look like a UML sequence diagram.
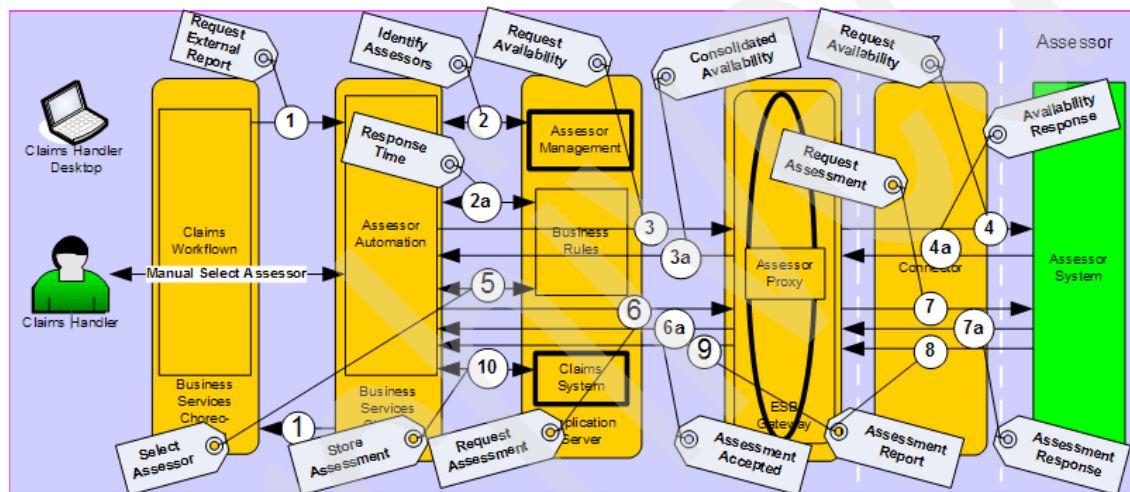
Figure 6-1   Solution flow

6.1.1 Interaction descriptions

The next stage in the refinement is to describe the interactions in more detail, specifying their qualities of service, particularly if they are synchronous or asynchronous and one-way or two-way interaction. These are listed in Table 6-1 on page 189. Also we want a verbal description of the message that is passed. Later, we will need to define the interfaces to the components so that we can specify the message contents for the flows precisely.

Table 6-1   Interaction Descriptions

Flow	Description
1	The new process gets started from the existing Claim Workflow by requesting an external assessor report. The interface for this request is: requestAssessor, an XML document that is delivered with WebSphere MQ queue. This is a synchronous call/return (see flow (1a) and expected to have a two-day response time. The underlying parallel processing engines can implement such long duration synchronous requests efficiently and reliably. The assessor automation process, which is implemented with Business Process Choreography gets this message and starts a new instance of the process
2	The first task of the assessor automation process is to invoke the Assessor Management system to get the list of available assessors. The interface is assessorManagment. This is a synchronous call/return and expected to have a sub-second response time.
2a	The second task is to get the response time based on the service level agreement with the customer from the Assessor Management System in parallel. The interface is requestResponseTimePT.
3	With this information the assessor automation process invokes the assessor proxy in step 3. The interface is assessorAvailability. The process waits for a response from the AssessorProxySystem to confirm it has successfully received the request.
4	The AssessorProxySystem takes the list of assessors from flow 3 and in a distributor service builds the request messages tailored to the format required for each assessor. The requests are routed through the gateway as one-way flows.
4a	The AssessorProxySystem waits for the time given by the RequestResponseTimePT from flow 2a and when the timeout occurs aggregates the responses it has received in the 4a flows.
3a	The assessor automation process waits until the AssessorProxySystem returns the list of available Assessors in step 3a. Flows 3a is expected a few hours after flow3.
5	The assessor automation process then gives the list of available assessors to the Business Rules Engine to return one assessor who should produce the report. The process uses the preferredAssessor interface of the Business Rules. This is a synchronous call/return with an expected subsecond response time.

Flow	Description
6	The process then requests the assessment report by invoking the assessor proxy with the interface allocateAssessmentRequest in step 6 with an asynchronous call which might take several hours. It can only be implemented as a synchronous call/return if it can be implemented both by the long running business process and the underlying enterprise service bus efficiently and reliably. In version 5 of the WebSphere platform this is hard to achieve, so we implement the interaction as one-way requests initiated at each end using SOAP/http.
7	The AssessorProxySystem passes the allocateAssessmentReport to the chosen assessor and waits for an acknowledgement
7a	The AssessorProxySystem receives the acknowledgement (accept or not) from the assessor via the interface deliverAssessmentReponse.
6a	The AssessorProxySystem delivers the acknowledgment (accept or not) of the Assessor to the assessor automation system in step 6a.
8	The assessor sends a one way flow to the assessor proxy via the interface deliverAssessment.
9	The assessor proxy transfers the report back to the automated assessor process in a one-way flow using the interface assessorReport
10	The automated assessor process invokes the claim system's document handler using the storeAssessorReport interface. The flow is a synchronous call/return with an expected sub-second response time.
1a	The assessor automation system finally returns to the claims workflow by using the requestAssessorResponse interface

From this information we can start to build a sequence in Rational Software Architect.

6.1.2 Sequence diagram

A sequence diagram shows the behavior of a collaboration. We focus on two things: the detailed behavior of the RequestExternalReports collaboration and the interface where it interacts with the ClaimInvestigation_TOBE collaboration.

The solution architect needs to create a new collaboration to show this behavior for a number of reasons:

1. The architect has added new component (proxyAssessorSystem) to the collaboration.

2. The collaboration in the WebSphere Business Integration Modeler cannot be altered by the Rational Software Architect.

3. We do not intend to alter the business process description agreed between the business analyst and the solution architect to incorporate changes that only affect the technical view of the solution. As long as the behavior of the business process remains the same, then the solution architect does not need to renegotiate the contract with the business analyst. When the architect completes the PIM, the analyst and architect will review it to ensure the changes and extensions the architect has made do not alter the behavior of the business process model.

Create the new collaboration, then drop the interfaces onto the interaction diagram.

1. In the **Model Explorer** select the **ITSO Architect** → **Claim Investigation** right-click → **Add UML** → **Collaboration** and call it `External Claim Assessor.`

2. Right-click **External Claim Assessor** → **Add Diagram** → **Sequence Diagram** and name the diagram `Sequence Diagram.`

   – **Refactor** the name of the interaction from **Interaction1** to something more descriptive, such as `Assessor Automation Interactions.`

3. Populate the sequence diagram with the following lifelines:

   a. ClaimsInvestigation_TOBE collaboration in the WebSphere Business Integration Modeler model

   b. AssessorAutomation and proxyAssessorSystem interfaces defined as part of the architect's component model in Figure 5-8 on page 165

   c. Drag the Assessor Management, Business Rules Engine, Document Handler and Assessor from the WebSphere Business Integration Modeler model RootResourceModel → Roles package.

   > **Tip:** Drop the components onto the diagram in the order you want to show them on the diagram. The order should follow a (generally) left to right flow of iterations in the model.
   >
   > After you have dropped a lifeline on the sequence diagram, it is difficult to move it. So it pays to get the order your want first time!

4. After you have populated the sequence diagram, you can show the collaboration visualization by right-clicking the **External Claim Assessor** collaboration → **Visualize** → **Show in Browse Diagram**.
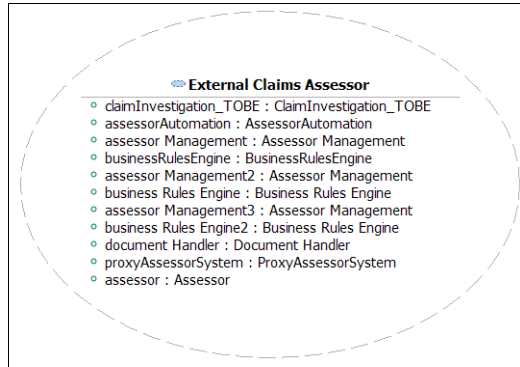
Figure 6-2   External Claim Assessor collaboration

The next step is to add the interactions to the sequence diagram by adding synchronous or asynchronous messages to the lifelines. If we have the roles and interfaces defined correctly then all the operations that are required to label the interactions are pickable. However, drawing a sequence diagram often shows some problems with the design and our design is no exception. Interactions had to be added to the following interfaces to cater for flowing asynchronous responses.

► Assessor Automation

   – ReturnAvailability
   – ReturnAssessmentConfirmation
   – ReceiveAssessmentReport

► proxyAssessorSystem

   – ProxyReturnAvailability
   – ProxyAssesmentConfirm

There were two reasons for this:

► In the case of flow 6 and 7, we wanted to have an acknowledgement from the assessor that they would perform the assessment, in addition to sending the assessment report (flow 8 and 9).

► There is only one combination of flows (flows 6, 7, 8 and 9) that cannot be modeled as a call/return pair (two replies, an acknowledgement and a data reply, are both requested). Strictly speaking, we do not really need all of the new interfaces if we could completely reply upon having asynchronous middleware and long running transactions such those as provided by WebSphere MQ, WebSphere MQ Workflow and WebSphere Business Integration Server Foundation. For example, the request for assessor availability is planned to take two hours. At the business level, this is a straightforward request/reply interaction and could be implemented efficiently

as such on the WebSphere middleware. But some transport layers, for example http:/SOAP, would not be suitable implementations of this call return pattern because they would need to block for the whole duration. It seems unreasonable to force the assessor system to implement its response within a single unit of work.

So the new interfaces were added to give the implementers more flexibility, and the infrastructure architect more choices of transport protocols.

When you finish, your sequence diagram will look similar to Figure 6-3 and Figure 6-4 on page 194. The annotations tie up with the solution flow diagram (Figure 6-1 on page 188). Note the use of synchronous and asynchronous flows.
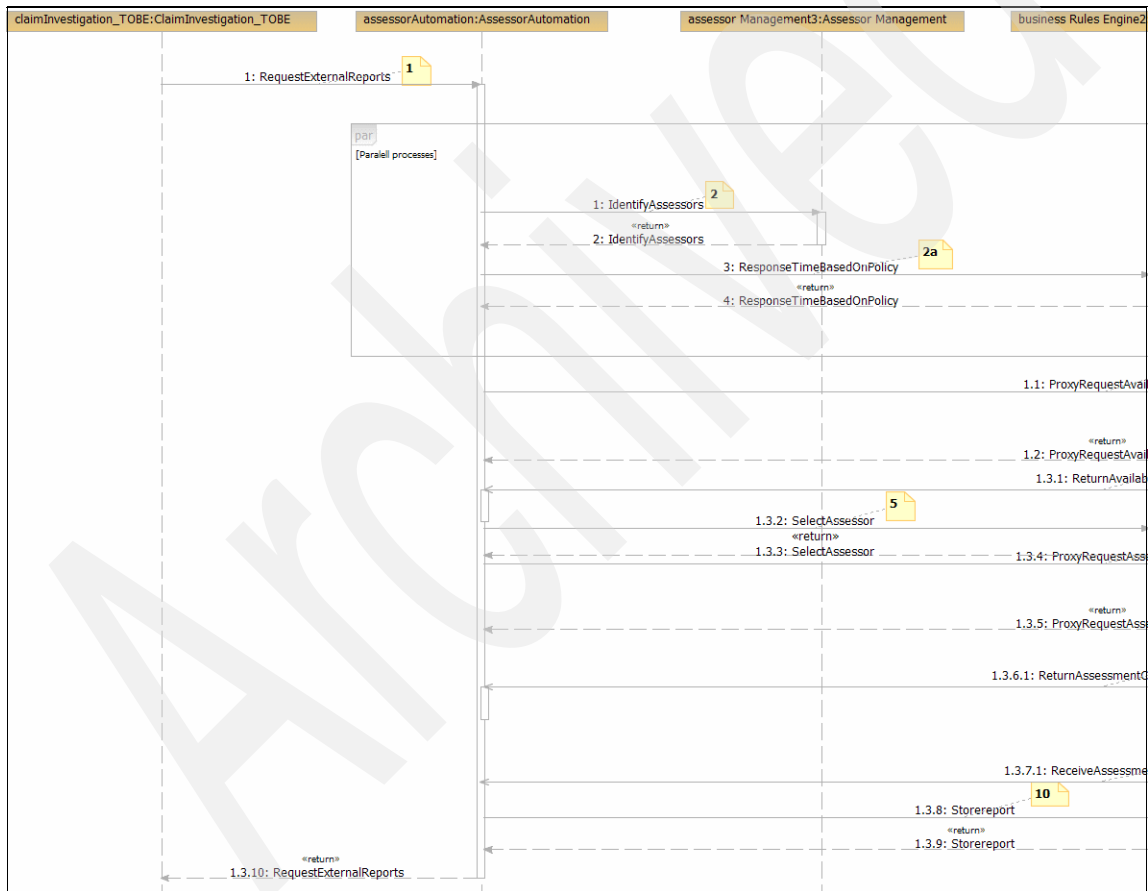
Figure 6-3 External Claim Assessor Sequence diagram: left side
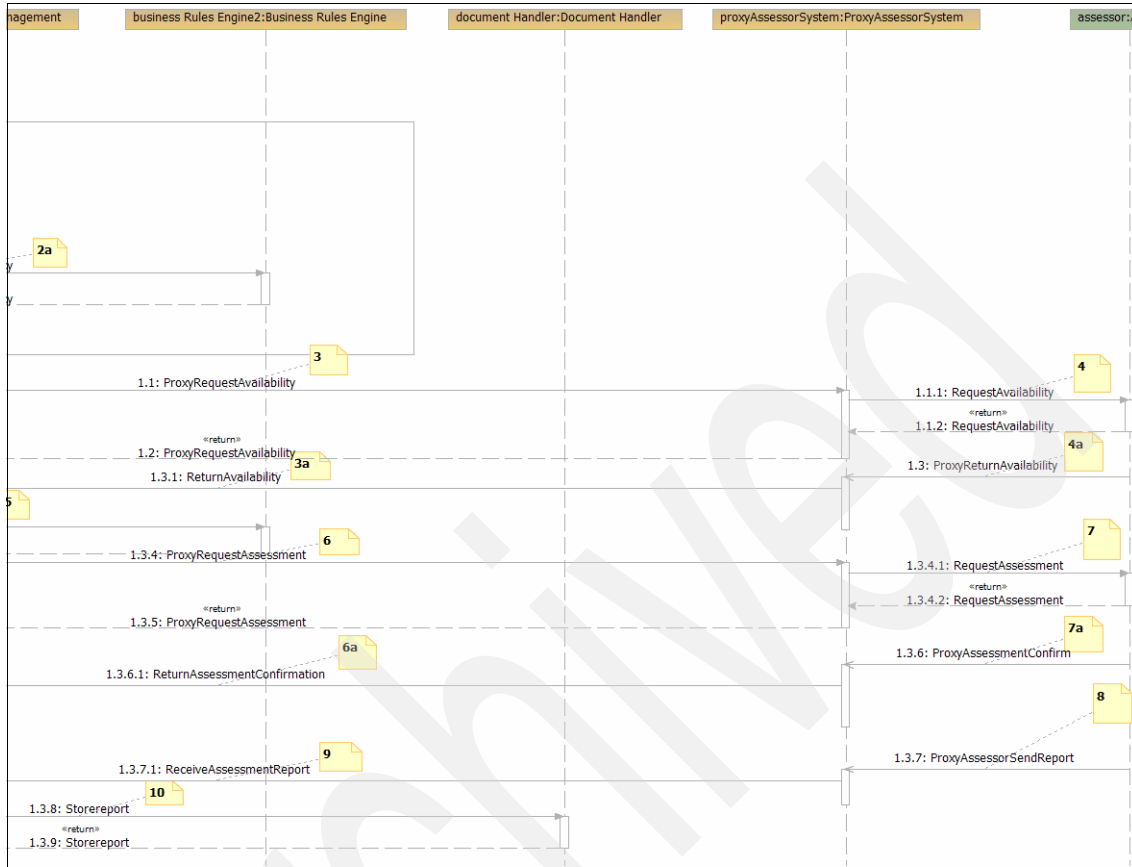
Figure 6-4   External Claim Assessor Sequence diagram: right side

6.2 Interfaces

We have now described the business process, the components, and the interactions in the solution. We have sketched out all the interfaces at the level of naming the operations that are required. The next step is to specify the interfaces in sufficient detail for the IT specialists to implement them.

Assumptions

We limited the scope of this redbook to looking at process integration and limited the amount of data used to only that which was essential to make the process work. If we had used a data model of the size and complexity of those in the insurance industry, it would result in some changes to the development process to accommodate the needs of the Data Architect, and the use of data modeling

tools. We would need to consider the interaction between the process and data modeling tools, and address issues concerning data transformation between the data model and the interfaces provided by the existing components.

We have also assumed that all the automated services (Assessor Management, Business Rules Engine, Claim System) and the Assessor process are already available as EJBs. We are focussed on building the business process and integrating the services together rather than creating the programs.

6.2.1 Choice of interface description language

We plan to use Web services to connect the components, therefore we will use Web Services Description Language (WSDL) to describe the interfaces in the architecture model. WSDL is the natural way to describe Web services.

Using WSDL as an interface definition language

As an interface description language WSDL has a number of strengths:

1. It is independent of the programming technology used to implement the interfaces.

2. A number of programming technologies support using WSDL to generate and run Web services, whether as Java Beans, Enterprise Java Beans, .NET services or as message flows in WebSphere Business Integration Message Broker.

3. It is supported by excellent tooling.

However because it is a new technology there are gaps in the tooling and there are questions to be answered before it is implemented fully in mode- driven development.

1. It has both a logical aspect, represented by PortTypes and Messages, and a physical aspect, represented by Services and Bindings, which cuts across the notion of an interface being an early step in the refinement of a model into an implementation.

   The simple solution is to be selective about which parts of the WSDL definition to complete at any stage in the design.

2. It has not been fully integrated into UML tools to the extent, say, that EJBs are integrated into Rational Software Architect. There is no simple way to input or output WSDL definitions to and from a UML model.

   We came up with four recipes to interchange WSDL and UML, depending on where the solution architect was obtaining the interface definitions. See Figure 6-5 on page 197.

3. Because various tools are missing WSDL importers, there are some difficulties in constructing a tool chain based on WSDL as the interface definition language. But there is no other stronger contender with better coverage of tools, except perhaps XML schema. We think WSDL is preferable to XML schema as an interface definition language because XML schema is principally a way to provide language- independent data typing. WSDL describes component interfaces made up of operations *and* typed messages.

A combination of using both WSDL and embedded .xsd (XML schemas) has some attractions because of wider tooling support for the combination of WSDL and schemas. We have tended to place type definitions in line in the WSDL files because the EJB tools we use generated data types directly in WSDL files. Had we been more data-focused, we might have used .xsd more, and used import statements in WSDL files. The choice between all WSDL, or a combination of WSDL and .xsd, is largely one of what works best for the tool chain. We discuss this again in Section 13.1.4, "Meta data" on page 492, and come to the opposite conclusion that using a mixture of .xsd and .WSDL would have worked better for us. This is a really hard decision to get right!

From a project management perspective, the notion that all the interfaces used in the project can be extensively documented in a *single* format is very valuable, so we decided that all the interfaces in the solution would be defined in WSDL, and collected together by the Solution Architect who is responsible for the interface definitions. Rational Software Architect provides an excellent WSDL editor. WebSphere Studio Application Development Integration Edition also provides an .xsd editor.

6.2.2 Creating WSDL interfaces

There are basically three ways to create a WSDL interface.

1. Top down using a WSDL editor

2. Bottom up, by generating WSDL from some other implementation or interface definition

   Rational Software Architect generates WSDL files from EJBs for example.

3. Meet in the middle, building some of the interface with the WSDL editor, and importing the rest from some other format, such as importing types from .xsd files

Figure 6-5 on page 197 shows a number of different transformations between WSDL and other interface definition languages that can be used to create an architectural description in UML from implementation artifacts or alternative to generate implementation artifacts from UML.
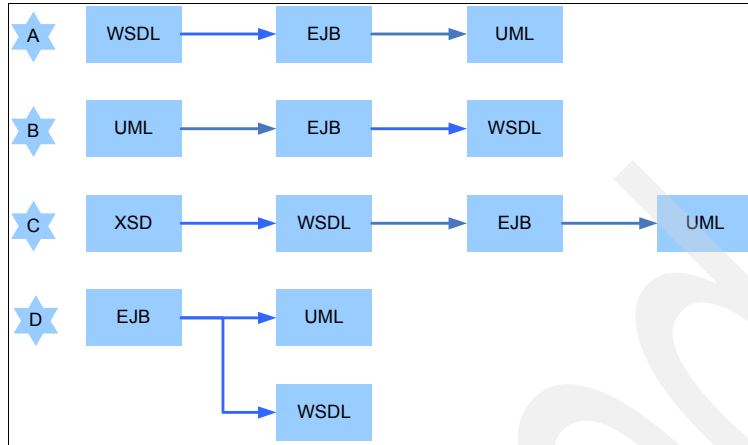
Figure 6-5   Integrating WSDL interfaces with the UML model

- ▶ **A**: This path was taken where we already had WSDLs defined. The details are explained in "A: WSDL to UML" on page 200.

- ▶ **B**: Path taken to create a new Interface. Use the EJB transformer in Rational Software Architect to generate an EJB and then generate a Web service. Creating an EJB from a UML model is explained. The details are explained in "B: UML to WSDL" on page 203.

- ▶ **C**: Path taken for interfaces defined for WebSphere Business Integration Message Broker. The interface types were created as schemas in WebSphere Studio Application Development Integration Edition and imported into the Broker to create message sets. This is explained in Chapter 9.4, "Implementation of the message sets" on page 289.

  The schemas were also imbedded into WSDL files in WebSphere Studio Application Development Integration Edition and an EJB was generated. The EJB was imported into Rational Software Architect and a UML interface description extracted. The detail for this is the same as "A" once the .xsd file has been imported.

  The EJB could also be used for unit testing clients of the Web service without needing to wait for the WebSphere Business Integration Message Broker implementation.

- ▶ **D**: Existing EJB interfaces (such as the Assessor Management System) were imported into Rational Software Architect and both UML and WSDL interfaces generated from the EJBs. This method is explained in "B: UML to WSDL" on page 203.

6.2.3 Sources of Interface Information for the scenario

The WSDL files and an .xsd file defining the interfaces have been created either from the EJBs that are going to be used to implement some of the services, or as part of task of defining the interfaces to the AutomatedAssessor process and the AssessorProxyService. The general rule is that the service owner of the Web service will cooperate with the solution architect in defining the service interface.

Table 6-2 lists the WSDL files so that you can find them in the additional materials directory (.\SG24-6636\) that comes with this redbook. The names include the flow number for easy cross referencing. All the WSDL files are in the .\SG24-6636\RSA\Project Interchange\wsdl subdirectory.

Table 6-2   WSDL and .ear file details

Flow	Component (Interface owner)	Interface details - WSDL file name - source generated from
1	Assessor Automation	ExternalClaimAssessorsInterface.wsdl. Which is derived from a folder in proxy(1).wsdl. Proxy(1).wsdl was built using the fdl2wsdl tool and it is described in Chapter 11, "Modify the Claim Investigation process" on page 453.
2	Assessor Management	AssessorManagement(2).wsdl .\SG24-6636\WAS\Flow2\Flow2_AssessorManagementService_SOURCE.ear
2a	Business Rules Engine	RequestResponseTimePT(2a).wsdl .\SG24-6636\WAS\Flow2A\Flow2A_ResponseTimeRules_SOURCE.ear
3	Proxy Assessor System	AssessorAvailability(3).wsdl .\SG24-6636\WBIMB\schemas\AssessorAvailability_3.xsd
4	Assessor	Availability(4).wsdl .\SG24-6636\WAS\Flow4and7\AssessorAvailabilityApplications_withSource.ear
4a	Proxy Assessor System	AssessorAvailabilityPT(4a).wsdl .\SG24-6636\WBIMB\schemas\AssessorAvailabilityPT.xsd
3a	Assessor Automation	AssessorAvailablityList(3a).wsdl
5	Business Rules Engine	PreferredAssessor(5).wsdl .\SG24-6636\WAS\Flow5\Flow5_AssessorRules_SOURCE.ear

Flow	Component (Interface owner)	Interface details - WSDL file name - source generated from
6	Proxy Assessor System	AllocateAssessmentReport(6).wsdl .\SG24-6636\WBIMB\schemas\AllocateAssessmentReque st_6.xsd
7	Assessor	DeliverAssessment(7).wsdl .\SG24-6636\WAS\Flow4and7\AssessorApps.ear
7a	Proxy Assessor System	DeliverAssessmentResponse(7a).wsdl .\SG24-6636\WBIMB\schemas\DeliverAssessmentRespon se_7a.xsd
6a	Assessor Automation	AllocateAssessorResponse(6a).wsdl
8	Proxy Assessor System	AssessorReport(8).wsdl .\SG24-6636\WBIMB\AssessorReport_8.xsd
9	Assessor Automation	AssessorReport(9).wsdl
10	Claim System	StoreAssessmentReport(10).wsdl .\SG24-6636\WAS\Flow10\Flow10_DocumentHandler_SO URCE.ear

6.2.4 Creating the interface definitions

Rather than go through every detail of creating interfaces and describing the fields, all the interfaces are provided in the additional materials together with the source we used to create the WSDL interface where it was not typed in from scratch. To create the interfaces, we used one of the procedures, A through D, in the previous section.

All the WSDL interface descriptions and the .ear files can be imported into Rational Software Architect and browsed using the Web service editor. The files can be stored the ITSOLGI Architecture project.

Many of the WSDL files were generated from the EJB implementations referred to in Table 6-2 on page 198 following a bottoms up approach. A bottoms up approach is common in Enterprise Integration projects, whereby Web services are used to wrapper existing interfaces, the EJB implementations in our case. Here, we demonstrate both a top down and a bottom up approach to incorporating the interfaces in the WSDL definitions in the UML model.

A: WSDL to UML
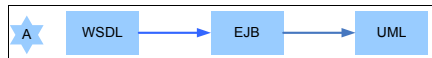
Figure 6-6   WSDL to UML

Approach A, starts with WSDL files . These can be created either in Rational Software Architect, or the tool of your choice and imported into Rational Software Architect.

Create new WSDL file in Rational Software Architect

To create a new WSDL file in Rational Software ARchitect, follow these steps:

1. **File** → **New** → **Other** → **WSDL** → **Next** → Select a project and name the file, say Assessor.wsdl, → **Next** → Select a target namespace for labelling the WSDL definitions uniquely. We use `http://itso.lgi.assessormgmt` as the root name and add `/Assessor` in this case to it. We then give it a prefix, say `ama`. Leave the **Create WSDL skeleton** checked, with **SOAP** and **document literal** selected → **Finish.** See Figure 6-7.
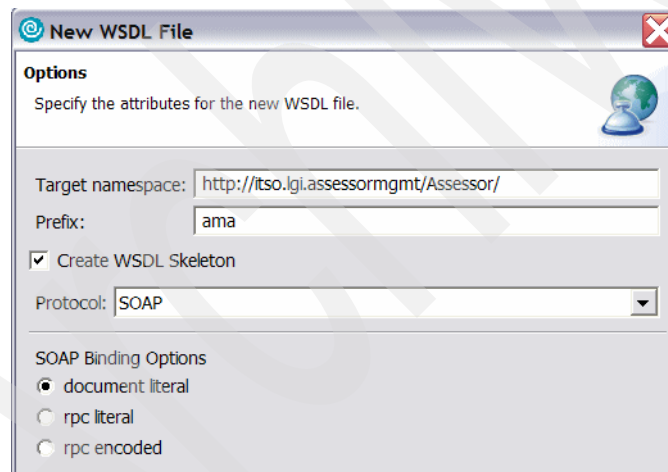
Figure 6-7   Creating a WSDL file - Options

2. You can now edit the WSDL file using either the graphics editor or the XML source page edit or a simple text editor by right clicking the Assessor.wsdl file and choosing one of the **Open with...** options.

> **Important:** In either the workspace or project preferences, add WS-I SSBP Require compliance.

An alternative approach, is to import the WSDL file.

Import a WSDL file directly and display it

To import a WSDL file, follow these steps:

1. We need to switch to the resources perspective because WSDL has not been integrated into UML and so there is no way to look at WSDL from a UML modeling perspective. Change to the **Resource** perspective. In the top right of the workspace, click the **Perspectives icon**, or **Window** → **Open Perspective** → **Other** → **Resource**.

2. Click **Model Explorer** → **ITSOLGI Architecture** → **Right click** → **New** → **Folder** → **WSDL** → **Finish.**

3. Drag and drop the WSDL files from the locations in Table 6-2 on page 198 into the WSDL folder you have just created.

4. Double-click one of the WSDL files such as **PreferredAssessor(5).wsdl**, to display the WSDL in graphical form or switch to the WSDL source file. See Figure 6-8 and Figure 6-9 on page 202.
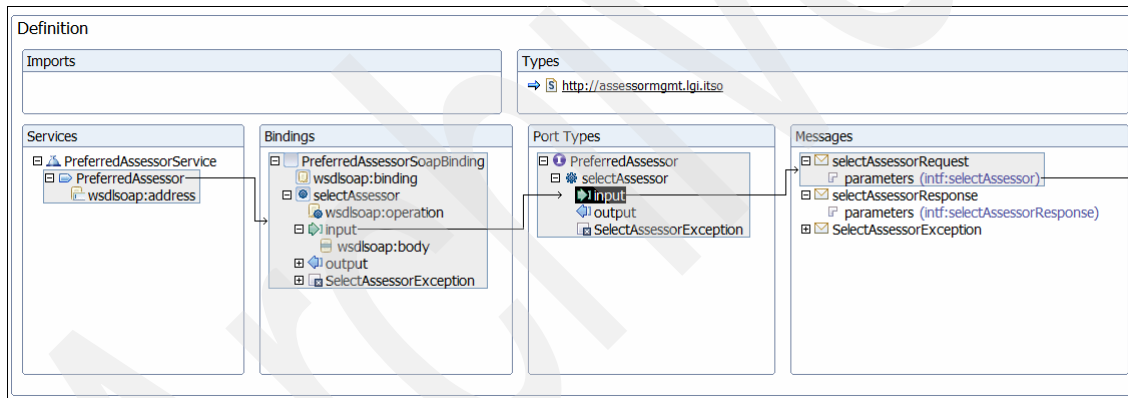
Figure 6-8   PreferredAssessor WSDL definition (1)
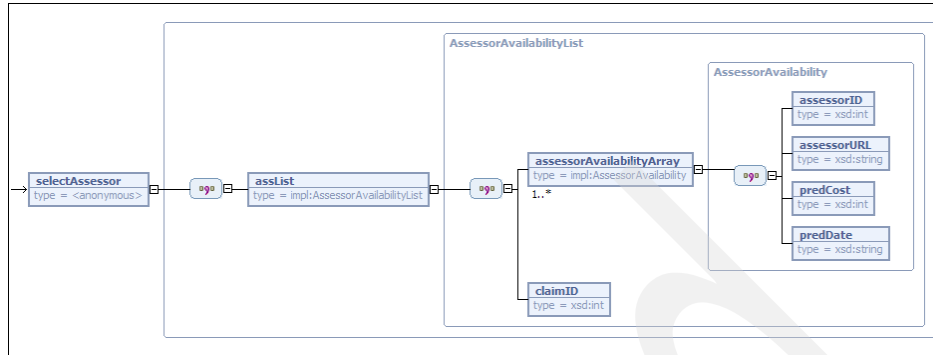
Figure 6-9   Preferred Assessor WSDL definition (2)

Generate an EJB skeleton for the Web service

Next, we generate an EJB from the Web service definition. We create a new Enterprise Application Project for the .ear files, rather than use the existing ITSOLGI Architecture project.

> **Tip:** You might not have all the J2EE and Web service perspectives available. In that case, check the Web services and J2EE developer in the workspace capabilities preferences.

1. Create a new Enterprise Application Project:

   a. Select **File** → **New** → **Project** → **J2EE** → **Enterprise Application Project** → **ITSOLGI Web services** → **J2EE Version 1.3** → **Target Server** → **WebSphere Application Server 5.1** → **Next** → **Finish** → Respond **Yes** to **Switch to the J2EE perspective.**

   > **Note:** We have implemented here on the WebSphere 5.1 platform. You will need to have installed the WebSphere 5.1 Test environment into Rational Software Architect to have this option in Rational Software Architect 6.0. We are selecting the J2EE version 1.3 as we are using the 5.1 test environment. J2EE 1.4 is first available when using the WebSphere Application Server 6.0 test environment.

   b. There might be an error associated with the applications.xml configuration file. Do not worry about this for now.

2. Create a new Web service for the PreferredAssessor:

   a. Select the **ITSOLGI Implementation** project → **Right click** → **New** → **Other** → **Web services** → **Web service** → **Skeleton EJB Web service** → **Deselect Start Web service in a Web Project** → **Select**

> **Overwrite files without warning** and **Create folders when necessary** → **Next** Browse to the WSDL file you imported → **PreferredAssessor(5).wsdl** → **Next.**

b. Edit the service deployment configuration to use **J2EE 1.3** on **WebSphere Application Server test environment 5.1** → **Next.**

c. Leave the default settings and click **Finish**.

Incorporate the interface in the UML model

Visualize the service (we are looking for the PreferredAssessor Interface which correspond with the PreferredAssessor PortType in the WSDL)

1. Return to the **Modeling** perspective

2. Open the **itso.lgi.assessormgmt** package in the **WebServiceProjectClient** java project.

3. Right-click **PreferredAssessor.java** → **Visualize** → **Explore in Browse diagram.**

   You should see something similar to Figure 6-10.

Figure 6-10   Preferred Assessor remote EJB interface

B: UML to WSDL
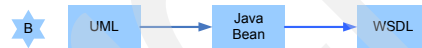
Figure 6-11   UML to WSDL

Approach B starts with a UML interface and then creates a WSDL file from it (Figure 6-11). To demonstrate how the approach works, we use a simple example of a service that multiplies two numbers together.

Create the UML interface

To create the interface, follow these steps:

1. Create the UML project.

   In Rational Software Architect, select **File** → **New** → **Other** → **UML Project** → Give it a name of Sums → **Next**. Accept the default selections (Blank Model) but change the default diagram to **Class Diagram** → **Next** → Don't select any projects and click **Finish.**

2. Create the UML Interface.

   a. From the right-hand palette, click **Interface** → drop it on the canvas and click the name **Interface1** rename it to Sums.

   b. Right-click **Sums** → **Add UML** → **Operation**, and rename it to Multiply.

   c. Expand the Interface Sums in the Model Explorer and right-click Multiply() → **Add UML** → **Parameter**. Call it m1.

   d. Repeat this to **Add Parameter** m2, and **Return result,** result.

   e. Type each of the parameters and the result:

      i. Select the parameter in the Model Explorer, open the **Advanced tab** in the properties editor, and find the **Type** property which will be blank.

      ii. Click in the second column of the **Type** property and set its type to **Integer.**

      iii. Repeat this procedure for **m2** and **result.**
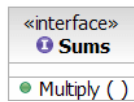
Figure 6-12   Sums Interface

Create the Java Interface

Create the UML to Java transformation (Figure 6-13), by following these steps:

1. From the main action bar, select **Modelling** → **Transform** → **Configure Transformations** → Select **UML to Java** → **New** → and call it SumsTransform.

2. Next to the Source: field click **...** → select the **Sums Interface** → In the Target area of the dialog click **Create New Target Container.**

3. In the Create a Java Project dialog box enter a **Project name:** SumsJava → **Next** → Click **SumsJava** on the **Order and Export tab** → **Finish.**

4. On the **Configure and Run Transformations** panel make sure **SumsJava** is selected in the Target UML element panel → **Apply.**

   See Figure 6-13 on page 205.
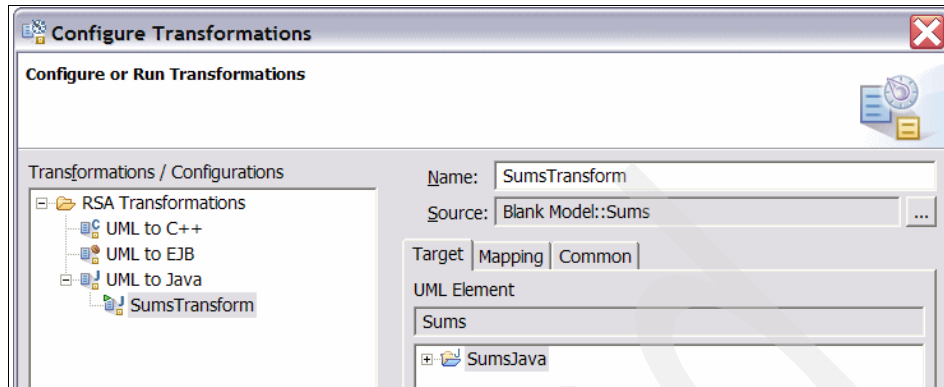
Figure 6-13   Configuration of the Sums UML to EJB Transformation

  f. The transformation can be run directly from the configuration editor, or selected from the main action bar (Figure 6-14).
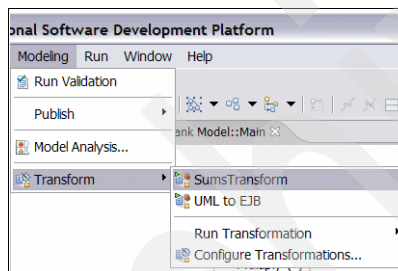
Figure 6-14   Running the SumsTransform from the action bar

This creates Sums.java in the default package of the SumsJava project. Open the default diagram in the Sums project and drag the Sums.java file into it. You can see a new Sums interfaced stereotyped to <<Java Interface>> (Figure 6-15).
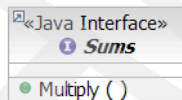
Figure 6-15   Sums Java Interface

Create the WSDL mapping

To create the mapping, follow these steps:

1. Create a Web Project.

  a. Select **File** → **New** → **Project** → **Dynamic Web Project** → call it `SumsWebProject` → **Next** → deselect any options → **Next** → **Finish.**

b. Right-click **JavaSource** in the **SumsWebProject** → **New** → **Package** and call it `mathematics`.

c. CTRL-Click **Sums.java** in the SumsJava folder and drag it to the **JavaSource** folder → **OK** to refactoring the elements.

2. Create a Java Web service.

a. Select the **Sums.java** file in the mathematics package → **New** → **Other** → **Web service** → **Next** → **Java Bean Web service** (accept the other defaults) → **Next**

b. mathematics.Sums should be in the Bean field (Figure 6-16) → **Next.** Otherwise, **Browse files...** → select **Sums.java** in the SumsWebProject folder → **Next.**

Figure 6-16   Selected the Mathematics.Sums Interface to turn into a Web service

c. On the next panel, the service project is SumsWebProject, and the EA project is **SumsWebProjectEAR** → **Next.** Leave the service endpoint interface and the Web service options unchanged. See Figure 6-17 on page 207) Click **Next** → **Finish.**
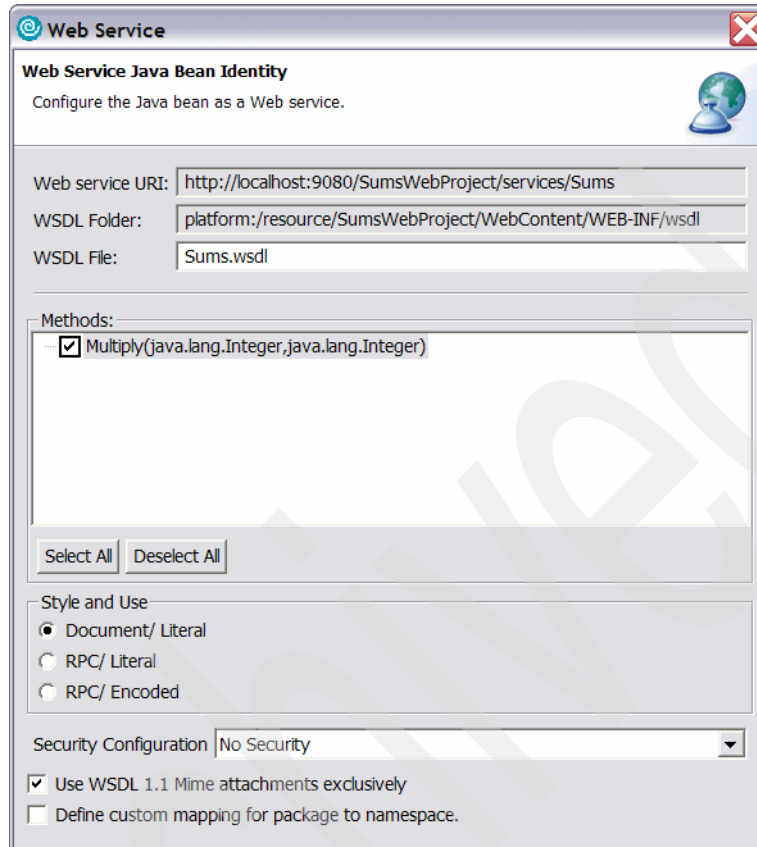
Figure 6-17 Web service options

3. Browse the resulting WSDL file, which you can find in the Project Explorer in the Web Perspective, and select **Dynamic Web Projects** → **SumsWebProject** → **WebContent** → **wsdl** → **mathematics** → **sums.wsdl.** See Figure 6-18 on page 208.
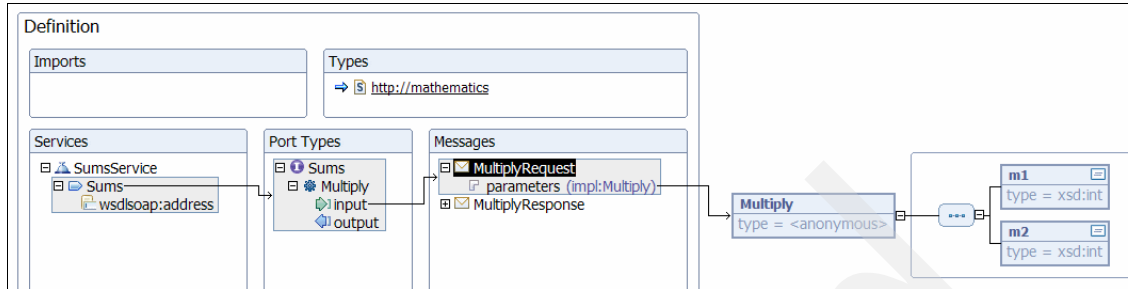
Figure 6-18   Sums Web service

Now that we have WSDL for this operation, we can also generate an EJB following the procedure for A.

D: EJB to UML and WSDL
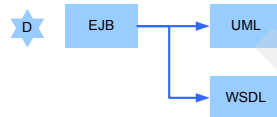
To demonstrate this, we import one of the .ear files from Table 6-2 on page 198 and generate the WSDL interface definition.

1. Import the .\SG24-6636\WAS\Flow2\Flow2_AssessorManagementService_SOURCE.ear file into the new EAR project.

    a. Select the **ITSOLGI Web services** folder, right-click **Import** and select **EAR file** → **Next.**

    b. Select **Flow2_AssessorManagementService_SOURCE.ear** → the existing **ITSOLGI Web services project (** Created in example A in "Generate an EJB skeleton for the Web service" on page 202) → deselect **Import Project.** Select **Overwrite existing resources without warning** → **Next** → **Next** → **Select Allow nested projects overwrites.**

    c. Make sure the **AssessorManagementServiceEJB.jar** and **AssessorManagementServiceEJBRouter.jar** are both selected → **Finish.**

2. Inspect the WSDL file for the AssessorManagement Service.

    a. Select **Model Explorer** → **Web services** → **Services** → **AssessorManagementService.**

    b. Open the **WSDL:AssessorManagementServiceEJB/ejbModule/META-INF/wsdl/AssessorManagement.wsdl file.** See Figure 6-19 and Figure 6-20 on page 209.
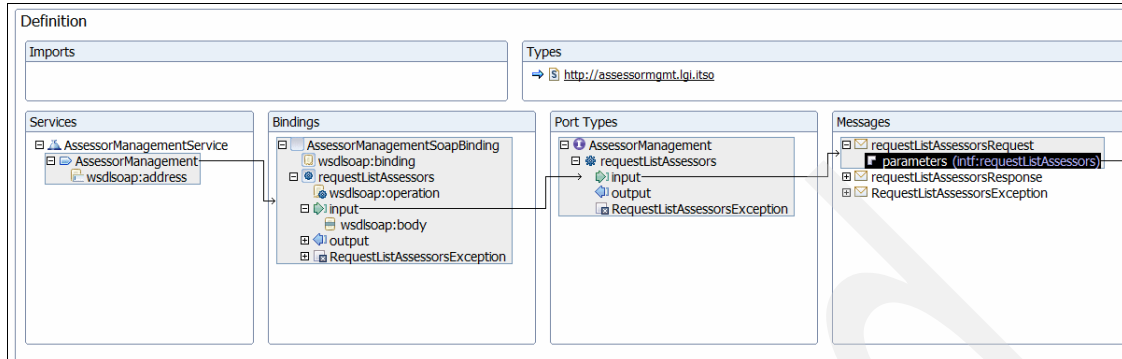
Figure 6-19   AssessorManagement.wsdl: part 1
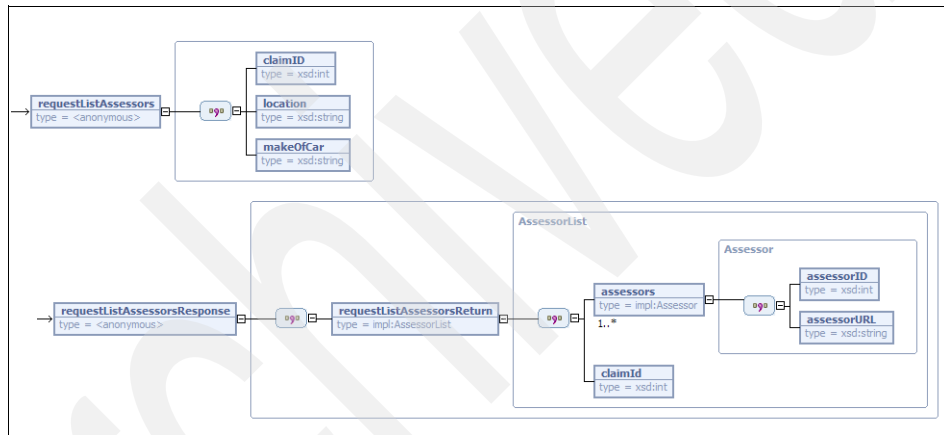
Figure 6-20   AssessorManagement.wsdl: part 2

3. To use the EJB interfaces in UML, drag the EJB remote interface into any UML diagram on which you are working. The next section illustrates incorporating the interfaces we have created from the WSDL definitions for the ExternalClaimsAsessor into a UML component diagram and a UML sequence diagram.

6.2.5 Incorporating the interfaces into the UML model

Whichever approach to defining the interfaces that we take now, we have a WSDL file and EJBs for each interface. We can use the EJBs to incorporate the

actual interfaces derived from the WSDL definitions into the UML model. We can demonstrate this for the AssessorManagement service.

1. Create a Component Diagram called `Assessor Management Component Diagram` for the **Assessor Management Component** in the ITSO Architect project. Populate it with the **AssessorAutomation Component**, the **Assessor Management Component**, and the **AssessorManagement Role** (**BusinessWorker**) from the WebSphere Business Integration Modeler model.

2. In the **Model Explorer** → **AssessorManagementServiceEJB** → **ejbModule** → **itso.lgi.assessormgmt** → **AssessorManagement.java** drag and drop the **AssessorManagement Java Interface** onto the Assessor Management Component Diagram.

3. Create a Realizes relationship from the **AssessorManagement Component** and the **AssessorManagement Java Interface** and a refines relationship between the **AssessorManagement Java Interface** and the **Assessor Management Business Worker**.

4. Make sure the AssessorAutomation Required Interfaces include AssessorAutomation by dragging the **Assessor Management Role** from the **AssessorManagement Component** onto the Required Interfaces part of the AssessorAutomation Component.

5. If the relationships do not already exist, create a **use** relationship between the AssessorAutomation Component and the AssessorManagement Component.

The component diagram should look similar to Figure 6-21.
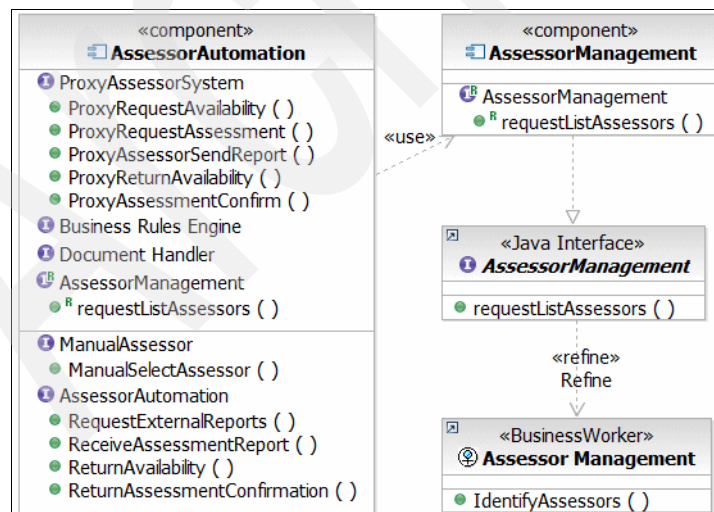
Figure 6-21   Refinement of the AssessorManagement system

6. We can now modify the sequence diagram (Figure 6-3 on page 193) by changing the lifeline from using the AssessorManagement Role to using the AssessorManagement Java Interface. We can now use the operation from the AssessorManagement Java Interface to specify the interaction between the AssessorAutomation system and the AssessorManagement system. The fragment around the AssessorManagement lifeline is illustrated in Figure 6-22.
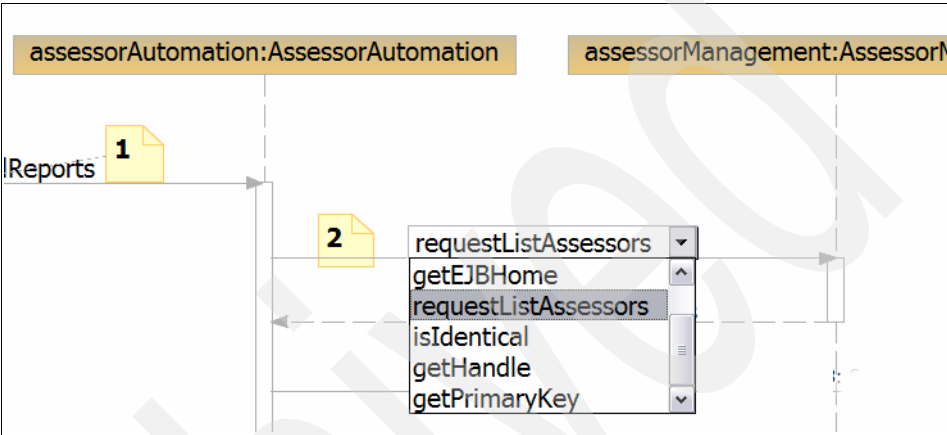
Figure 6-22   Fragment of the refined sequence diagram

6.2.6 Summary of the interfaces

All the details of the interfaces can be found in the additional material for this redbook, either by going to the source files listed in Table 6-2 on page 198, or by importing the ITSO Architecture project in the architect's workspace, found in the .\SG24-6636\RSA\Project Interchange\ITSO Architecture directory in the additional materials.

Table 6-3 highlights some important decisions the architect made about the interfaces.

Table 6-3   Interaction characteristics

Flow	Component (Interface owner)	Interaction characteristics
1	Assessor Automation	This uses JMS rather than a SOAP service. It flows XML messages over JMS. The choice of transport is a result of choosing to use a request/reply interaction between long running processes - Process choreographer only supports JMS/XML for this style of interaction.

Flow	Component (Interface owner)	Interaction characteristics
2	Assessor Management	This is a straightforward call/return interface. It will map to a partner link in BPEL and the transport options are SOAP/Http or SOAP/JMS. We have chosen SOAP/Http. SOAP/JMS would be the transport of choice here, by a small margin. The connection does not go outside LGI so there are no interoperability issues. SOAP/JMS offers better scalability and management over SOAP/Http at the cost of higher initial set up costs to define its queues, connection factories, destinations and the WebSphere MQ infrastructure. The interactions are short lived, so the differences in quality of service offered by SOAP/JMS and SOAP/Http are not a major issue here. However at the time we did the integration there were published restrictions on the use of SOAP/JMS and business Process Choreographer so we decided to standardize on the use of SOAP/Http for all our SOAP transport
2a	Business Rules Engine	As above
3	Proxy Assessor System	This is a more interesting interaction as the request results in a process fanning out requests to multiple assessors and waiting for their responses. It aggregates the responses it receives into a single reply (flow 3a), throwing out late responses. From the perspective of flow 3, however, this is simply a call/return interaction with a long duration between the call and the return. Because of the decision to allow SOAP/Http to be used for all the interactions, the interaction was split into two, with the actual data response (3a) in a separate interaction from the request. The request could be implemented as either a one-way or two-way SOAP message - we chose a two-way implementation so that AssessorAutomation can know that the request has been received by the proxyAssessorSystem.
4	Assessor	The requirements were for a number of different protocols to be supported for flow 4. We have only implemented SOAP/Http. As for flow 3, the data response is a separate flow because of the long duration between the request and the reply - and because a reply is optional (the assessor may choose not to do the work). We used a SOAP/Http request/reply interaction for flow 4.

Flow	Component (Interface owner)	Interaction characteristics
4a	Proxy Assessor System	As above.
3a	Assessor Automation	This is implemented as a one-way SOAP/Http datagram
5	Business Rules Engine	See flow 2a
6	Proxy Assessor System	See flow 3
7	Assessor	See flow 4
7a	Proxy Assessor System	See flow 4a. In response to flow 7, the chosen assessor responds with two flows, 7a and 8, both of which occur some time after flow 7.
6a	Assessor Automation	See flow 3a
8	Proxy Assessor System	See flow 7a
9	Assessor Automation	See flow 6a
10	Claim System	This is a simple synchronous request/reply flow like flow 5.

6.3 The architect's contracts

The architect has two deliverables, the PIM and the PSM. See Figure 3-7 on page 54.

PIM

In the PIM, the solutions architect should review the following materials with the business analyst to verify that the implementation will deliver the function required by the business process model.

► The summary of the requirements for the solution, possibly using a RUP template such as the vision document. See 5.2, "Step 0: Collating requirements" on page 155.

► The component architecture (Figure 5-8 on page 165)

► The runtime mapping (Figure 5-22 on page 180)

► The behavior of the solution shown by the flow diagram or the sequence diagram (Figure 6-1 on page 188 or Figure 6-3 on page 193 and Figure 6-4 on page 194.)

► The data model as captured in the flows exchanged between the components as described in the messages section of the WSDL (Figure 6-9 on page 202, for example)

PSM

In the PSM, the solution architect lays out the requirements on the system infrastructure for the solution to work, and maps the solution to various technologies for implementation by IT specialists. The solution architect needs to deliver the following materials to the infrastructure architect and IT specialists in the PSM in addition to the PIM.

► The product mapping (either Figure 5-23 on page 182 or Figure 5-24 on page 184)

► WSDL bindings, as well as port Types and messages as in Figure 6-8 on page 201. Completion of the service definition would be up to the infrastructure architect.

► The runtime mapping (Figure 5-22 on page 180)

► The behavior of the solution shown by the flow diagram or the sequence diagram (Figure 6-1 on page 188 or Figure 6-3 on page 193 and Figure 6-4 on page 194.)

Building the Java Beans, as we did in the "Generate an EJB skeleton for the Web service" on page 202 goes beyond what is absolutely necessary in the architect's PSM. It is useful to do to connect the WSDL interface definitions and the UML model, but that can be achieved by commentary on the interfaces.

The development team needs to decide whether the additional effort required by the architect to build the skeleton EJBs is justified. It probably will be for the services that will be built as EJBs, Java Beans or C++ objects, but not for services that are wrapping existing components, or are built using different

technologies, such as WebSphere Business Integration Message Broker. The skills to build the Beans correctly more likely belong to the IT specialist who will implement the beans, rather than to the architect.

6.4 Making materials available

After the architect has completed the WSDL interface definitions, they should be made available to the IT specialists to build the implementations.

1. **Open** the resources perspective in Rational Software Architect → **ITSOLGI Architecture** → **WSDL** → Select all the WSDL files → right-click → **Export**... → **Zip file** → **Next**. Select options **Compress** and **Create directory structure for files** and check all the required files are checked → choose a destination file name such as **ClaimInvestigation WSDL files** → **Finish**. See Figure 6-23.

Name ▲	Type
AllocateAsessorResponse(6a).wsdl	WSDL File
AllocateAssessmentRequest(6).wsdl	WSDL File
AssessorAvailability(3).wsdl	WSDL File
AssessorAvailabilityList(3a).wsdl	WSDL File
AssessorAvailabilityPT(4a).wsdl	WSDL File
AssessorManagement(2).wsdl	WSDL File
AssessorReport(8).wsdl	WSDL File
AssessorReport(9).wsdl	WSDL File
Availability(4).wsdl	WSDL File
DeliverAssessment(7).wsdl	WSDL File
DeliverAssessmentResponse(7a).wsdl	WSDL File
PreferredAssessor(5).wsdl	WSDL File
Proxy(1).wsdl	WSDL File
RequestExternalReports.wsdl	WSDL File
RequestResponseTimePT(2a).wsdl	WSDL File
StoreAssessorReport(10).wsdl	WSDL File

Figure 6-23   Exported WSDL files

6.5 Conclusion

In this chapter the solution architect has refined the system architecture in Chapter 5, "System Architecture" on page 153 by developing the behavior of the solution. The solution architecture is described by sequence diagrams, WSDL interface diagrams and component diagrams.

Sequence diagrams show which interfaces are used in which collaborations, and taken in conjunction with the business process model, the order of the interactions between the components.

WSDL diagrams show a great deal of detail from the physical transport and address of the deployed components to the definition of the interaction behavior and the message schemas used in the interactions. Of course, not all the detail needs to be provided at this stage.

The component diagrams have a two-fold use. They relate the original interface definitions to the refined interface specifications used in the implementation, and, for components that are implemented by java beans or C++ classes, they can be used to build and then generate the objects implementing the model.

The architect has collected together all the interface material and exported it as WSDL documents. In this chapter, we have shown how it is possible to mechanically transform to and from WSDL interfaces into other formats, making the set of WSDL files a convenient way for the architect to describe and control all the interfaces in the project.

Part 3

Implementation

In this part we describe how to build the solution as defined by the business analyst and solution architect. Different IT specialists are involved in building different parts of the solution with the solution architect dealing with issues arising with the interfaces and the overall behavior of the solution.

217

7

Install and configure runtimes

In this chapter we build the target infrastructure on which the solution is to be deployed.

219

7.1 System Infrastructure

Figure 5-24 on page 184 shows the deployment architecture provided by the solution architect. The Infrastructure architect is responsible for mapping the deployment architecture onto physical servers to meet cost, maintenance, security and performance requirements. The solution architect provides the lists of deployable artifacts and the execution environments to which they deploy in the PSM (product specific mapping). The infrastructure architect provides the deployment specifications to map the execution environments onto physical servers.

We are not focusing on deploying the solution to a production environment, the tasks performed by the infrastructure architect, or deployment tools to assist the architect. Our limited goal is to build a runtime environment to show the mechanisms to get the solution exported from the tools test environment onto the target runtimes.

In the architecture, there is flexibility to use one or more servers for each component, or to combine components onto a fewer number of servers. The server mapping we will build is shown in Figure 7-1 on page 222.

The configuration in Figure 7-1 on page 222 we are going to build will work on a single workstation, but can easily be reconfigured to work on multiple workstations. The single mobile computer configuration uses VMware to run WebSphere Business Integration Server Foundation on a different virtual host from all the other components. This separation means that most of the interactions flow across virtual network connections and are not local to the virtual hosts. It is therefore more like a production deployment. The solution has also been built on four IBM Netfinity® xSeries® processors.

7.1.1 Mobile computer configuration

The solution runs on an IBM T42p Thinkpad with 2GB of virtual memory, 60GB of disk space and a 1.99Ghz Pentium M processor. We used VMware Workstation Version 4.5.2 build-8848. The two VMware hosts were configured with 640MB of memory each, splitting the memory equally between the physical host and the two virtual machines. The virtual machines were run from an IBM Portable 40GB USB Hard Drive P/N 09N4257 or from a second hard drive mounted in the IBM Thinkpad Ultrabay. There was no discernible performance difference between these two options. Note that USB 1.1 does not have the bandwidth to run the solution.

For usability reasons, all the Eclipse tooling was run on the physical machine with deployments of .ear files, message flows and message sets across network

connections to the virtual machines. The WebSphere MQ Workflow buildtime was run in the virtual machine.

To reduce the middleware overhead on the configuration, We made a number of simplifications to the logical architecture to fit into the physical hardware.

1. A single assessor has been implemented as an EJB and deployed onto SAH414A along with the other application EJBs. This saves having another Virtual Machine for the assessor, or running a second application server on one of the existing virtual machines

2. We have not implemented a DMZ or the Web services Gateway, so flows go directly from the Broker to and from the assessors

3. We only configured two WebSphere MQ queue managers, one for each virtual machine. The two queue managers form a cluster. We could have used four queue managers (one each for the workflow, application server, broker and server foundation nodes) with three queue managers running on SAH414A. This would make it easier to redeploy the solution onto multiple servers beca4se the queue managers would be associated with the middleware nodes. The overhead on SAH414A would have been rather larger so we opted to have one queue manager per virtual machine. However by defining all the queues as cluster queues, it is relatively easy to rehost the solution by defining any new queue managers as members of the same cluster. The queue names known by the applications will not change.
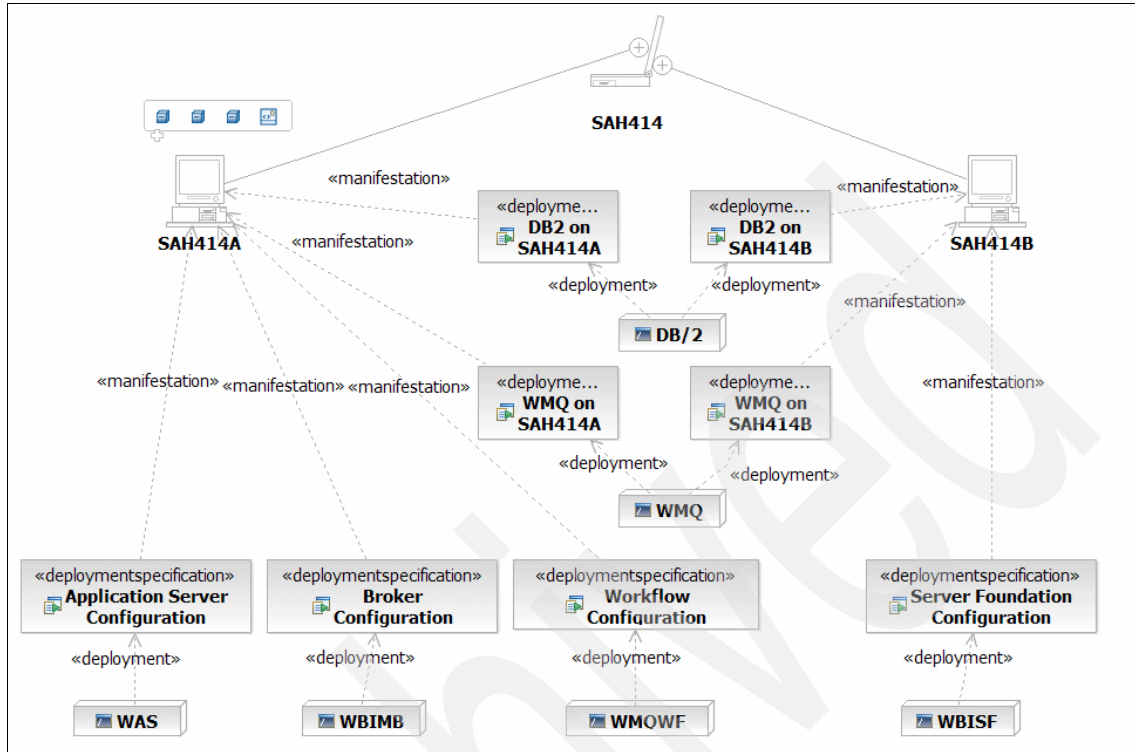
Figure 7-1   Deployment of RequestExternalReport onto a mobile computer configuration using VMware

7.1.2 Communication implementation

The system architecture requires all communication between services to use Web services. There is only one exception, and that is the connection between WebSphere MQ Workflow and WebSphere Business Integration Server Foundation which uses the XML over JMS solution provided in the WA0D supportpac.

The architecture also required a robust Web services implementation converting the existing WebSphere MQ messaging backbone at LGI to a service bus. However, because the SOAP/JMS implementation in across the WebSphere platform was not readily configurable to support this architecture, the solution architect redesigned the solution to work using SOAP/Http by replacing any long duration interactions with multiple one-way SOAP messages to increase the solution's robustness.

WebSphere Business Integration Message Broker supports both SOAP/Http and SOAP/JMS, so the essential LGI infrastructure has not had to change. It will be

relatively easy to redeploy the solution onto SOAP/JMS by changing some SOAP bindings and changing some input nodes and message flows on the broker. This could happen when the infrastructure is migrated to version 6 of the WebSphere products and WebSphere Platform Messaging is used to connect the WebSphere family nodes. See 13.2, "Tooling and middleware changes" on page 495.

7.2 Install SAH414A

The environments to install on SAH414A are shown in Figure 7-2.
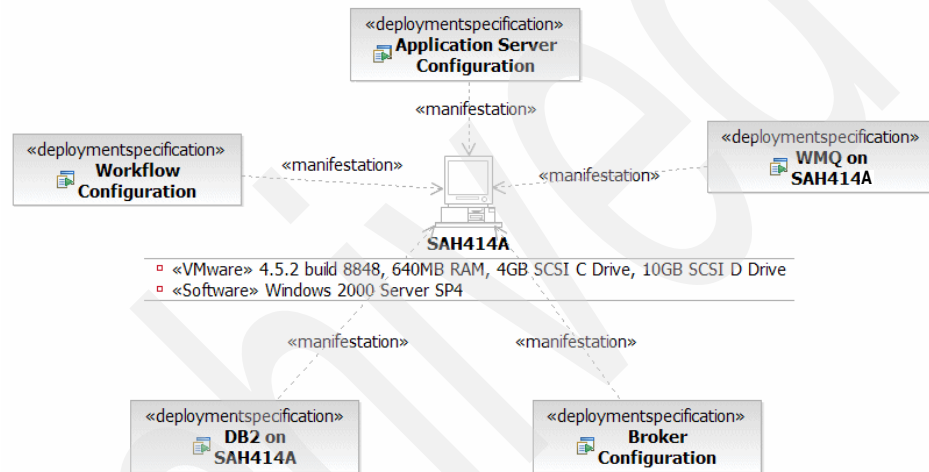
Figure 7-2   Environments to install and configure on SAH414A

All the software products are installed on the VMware D drive, which is configured as a dynamic spanned SCSI. If the D drive exceeds the 10GB allocated for it, then it can be grown.

It is recommended to install the product software in the order described here. There are some prerequisite dependencies of which to be aware.

> **Important:** Ensure that your Windows user ID is eight or less characters in length, specifically, not an Administrator ID. Long user IDs can lead to difficulties with message broker and DB/2.)

7.2.1 WebSphere MQ

The basic WebSphere MQ configuration is shown in Figure 7-3 on page 224. The queue manager, cluster and listener ports will be configured automatically by

WebSphere MQ Workflow configuration. We add the queues needed to integrate with WebSphere Business Integration Server Foundation in Chapter 12, "Integrate and test the business processes" on page 469. The queues needed for the message broker are configured in 7.2.5, "Install and configure the Message Broker" on page 230.
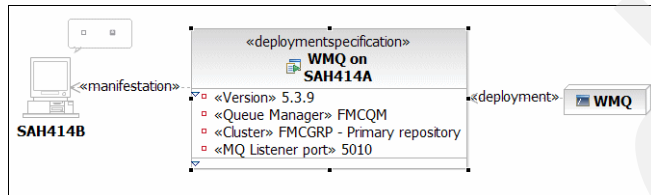
Figure 7-3   WebSphere MQ configuration

The installation of WebSphere MQ is straightforward, as is installing Fixpack 9. Visit this Web site to get Fixpack 9:

```
http://www-1.ibm.com/support/docview.wss?rs=172&context=SW900&q1=%22WebSphe
re+MQ+v5.3%22+CSD+Fix+Pack+Maintenance&uid=swg24008835&loc=en_US&cs=utf-8&l
ang=en+en
```

Install all the WebSphere MQ components, but do not install the default configuration. If you are installing the same VMware configuration as in this redbook, install all software products on the D Drive.

7.2.2 DB/2

The installation of DB/2 Extended Enterprise Edition Version 8.1 and Fixpack 6 is straightforward.
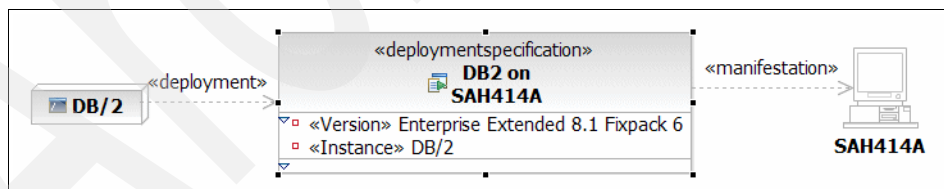
Figure 7-4   DB/2 Configuration

Remember the DB/2 administrator and password you set. Create the sample database to check the installation.

> **Tip:** Use the DB/2 configuration assistant to set DB2_MON_HEAP to around 900 from the default 66 to avoid problems.

7.2.3 WebSphere Application Server

The server configuration is shown in Figure 7-5.
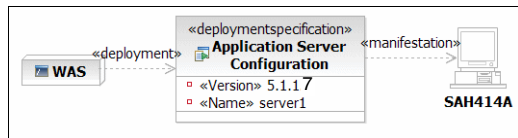
Figure 7-5   WebSphere Application Server configuration

To install this level of WebSphere Application Server and its fixes, we need to install them in this order.

1. WebSphere Application Server 5.1, available on your product installation CD.

   We installed into D:\WebSphere

2. WebSphere Application Server Fixpack 1 for 5.1. This converts 5.1 into 5.1.1.

   We unzipped the fixpack to D:\WebSphere\WAS511FP1.

   We should logically have called this directory WAS51FP1.

   Get the Fixpack here:

   ```
   http://www-1.ibm.com/support/docview.wss?rs=180&context=SSEQTP&dc=D420&q1=f
   ixpack&uid=swg24007195&loc=en_US&cs=utf-8&lang=en
   ```

3. WebSphere Application Server Cumulative Fix 7. This converts 5.1.1 into 5.1.1.7

   We unzipped the cumulative fix into D:\WebSphere\WAS511CF7

   Get WebSphere Application Server Cumulative Fix 7 here:

   ```
   http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24008771
   ```

This convention helps in keeping track of and installing fixes.

Installation of WebSphere Application Server is straightforward. But *do not install embedded messaging*. We are using WebSphere MQ. You will need to install the Http server. The installation is quicker if you decline to install the sample applications.

To install a Fixpack, run the WebSphere Application Server update wizard. To keep our paths simple, we installed the products near the root directory of the D drive. To run the update wizard, first you set the Windows environment variables by running setupcmdline.bat as in Figure 7-6 on page 226.

Figure 7-6   Running setupcmdline.bat

You can then run the update wizard by typing something similar to:

```
was511fp1\updatewizard
```

The exact command depends on what you called your unzipped Fixpack directory. The update wizard does take some time to start and run (Figure 7-7).

Figure 7-7   Run the update wizard

When you have finished, verify the installation:

1. Start **First Steps** → **Start the Server** and wait for the message `ADMU3000I Server server1 open for e-business; processid is xxx`

2. **Verify Installation** → wait for Installation verification is complete.

3. Close all the windows.

4. Now you are happy with the configuration, change the start up property of the WebSphere Application Server server1 service to automatic and we will not explicitly start the service again.

5. Because we are running the Application Server in a VMware image, it might be easier to administer it using a browser running on the native machine than using a browser in VMware. To confirm that WebSphere Application Server is running on SAH414A and to save a shortcut to the Administration Console, open a browser window and type:

```
http://sah414a:9090/admin
```

Save this as a shortcut.

> **Tip:** You should bring the embedded WebSphere Application Server test servers in Rational Software Architect and WebSphere Studio Application Development Integration Edition up to the same level as the runtime servers. To do this locate the runtimes directories where you have installed the tools and apply the same fix process. For example, to upgrade the WebSphere Application Server 5.1 runtime in WebSphere Studio Application Development Integration Edition, you might find the runtimes in the default directory location which is C:\Program Files\IBM\WebSphere Studio\Application Developer IE\v5.1.1\runtimes\base_v51 for integration edition.

7.2.4 Install and configure WebSphere MQ Workflow

We need to start by installing WebSphere MQ Workflow 3.5 with Fixpack 4, which is straightforward. Install all the optional components. Visit this Web site to get WebSphere MQ Workflow 3.5 Fixpack 4:

```
http://www-1.ibm.com/support/docview.wss?rs=795&context=SSVLA5&dc=D420&uid=
swg24007450&loc=en_US&cs=utf-8&lang=en
```

The configuration will look like Figure 7-8.
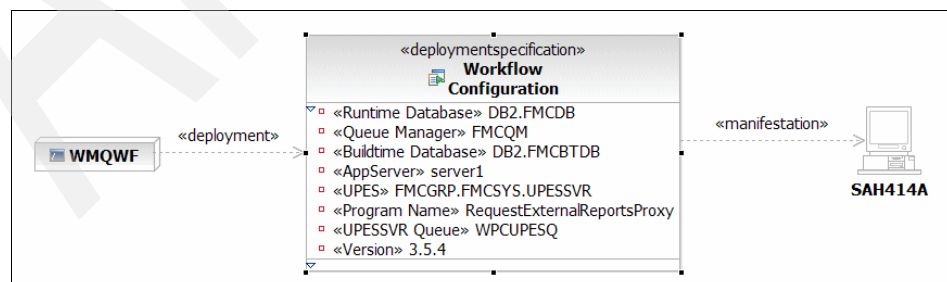
Figure 7-8   WebSphere MQ Workflow configuration

The configuration process builds a workflow buildtime, a Web client and an administration utility. We use the buildtime to create and modify workflows. This is described in Chapter 11, "Modify the Claim Investigation process" on page 453. First of all we have to configure the workflow buildtime.

1. Check all the prerequisite installations we have just installed. While these are not strictly prerequisite, this is the installation configuration we can confirm as successful:

   a. WebSphere Application Server 5.1 with Fixpack 1 and Cumulative Fix 3.
   b. WebSphere MQ 5.3 with Fixpack 9
   c. WebSphere MQ Workflow 3.5 with Fixpack 4
   d. DB/2 8.1 Enterprise with Fixpack 6.

2. The only prerequisite configuration is the default server1 application server that should have been verified using First Steps.

3. Run the **WebSphere MQ Workflow configuration utility** from the WebSphere MQ Workflow program group using the default settings except where specified otherwise:

   a. General Tab
      i. Select **New...** → **FMC** → **OK.**
      ii. Select all the options to configure as in Figure 7-9.
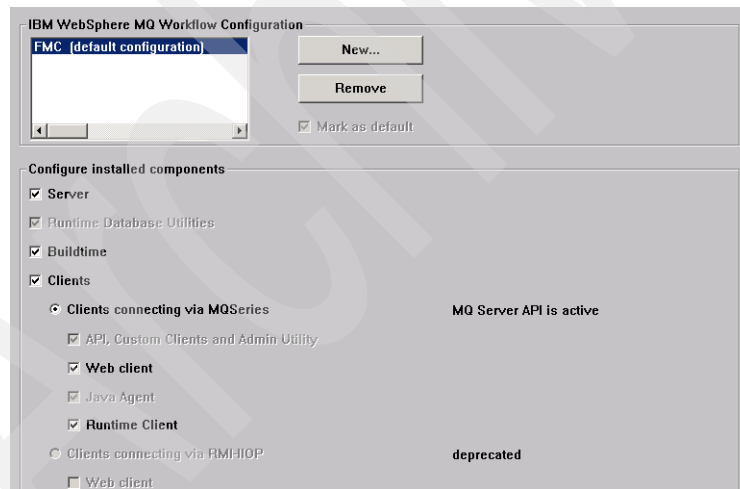
Figure 7-9   General Tab options

   b. Runtime Database Tab
      i. DB2 should already be present as the DB2 Instance, so select it.
      ii. Provide your DB2 user ID and password as the DB/2 connect parameters → **New...** Accept the defaults → **OK** → **Next.** See Figure 7-10 on page 229.
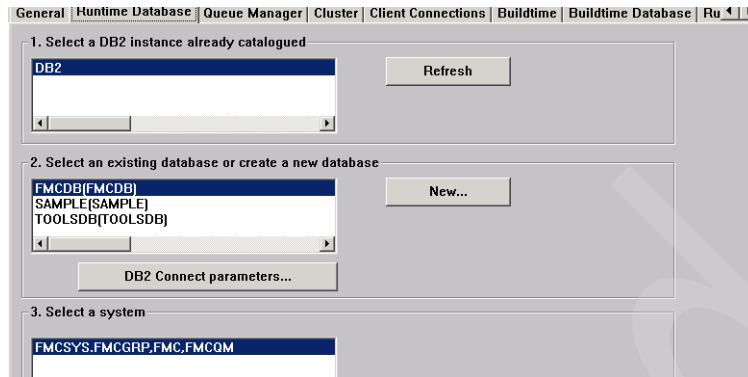
Figure 7-10   Runtime database properties

   c. Queue Manager Tab

      i.   Accept all the defaults → **Next**

   d. Cluster Tab

      i.   Accept the defaults → **Next**

   e. Client Connections Tab

      i.   Accept the defaults → **Next**

   f.   Buildtime Tab

      i.   Accept the defaults → **Next**

   g. Buildtime Database Tab

      i.   DB2 should already be present as the DB2 Instance → select it.

      ii. Provide your DB2 user ID and password as the DB/2 connect parameters → **New**... Accept the defaults → **OK** → **Next.** See Figure 7-11 on page 230.
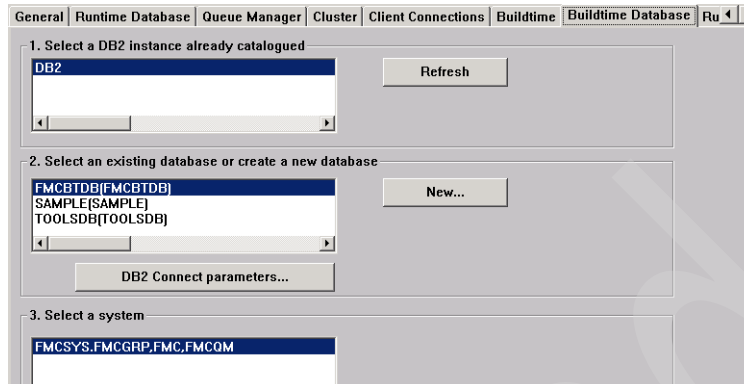
Figure 7-11   Buildtime database properties

    h. Runtime Client Tab
       i.   Accept the defaults → **Next**
    i. Web Client Tab
       i.   Accept the defaults → **Next**
    j. WebSphere Tab
       i.   Accept the defaults → **Next**
    k. JDK/JRE Tab
       i.   Accept the defaults → **Done**

> **Tip:** To find the WebSphere MQ configuration explorer without cluttering the desktop, add the WebSphere MQ Explorer Snap-in to the WebSphere MQ Services Monitor in the Windows System Tray:
>
> 1. Open the WebSphere MQ Services Monitor in the System Tray
>
> 2. Select **Console** → **Add/Remove Snap-in** → **Add...** → **WebSphere MQ** → **Finish** → **Close** → **OK.**
>
> 3. Select **Console** → **Save** → Respond **YES** to update to 1.2 format MMC.
>
> The next time you open the WebSphere MQ Services monitor, the WebSphere MQ Explorer will be ready to use as well.

Browse with the WebSphere MQ explorer to see all the WebSphere MQ definitions have been made for the cluster FMCQM.

7.2.5 Install and configure the Message Broker

We start by installing Version 5.0 of WebSphere Business Integration Message Broker. See Figure 7-12 on page 231.
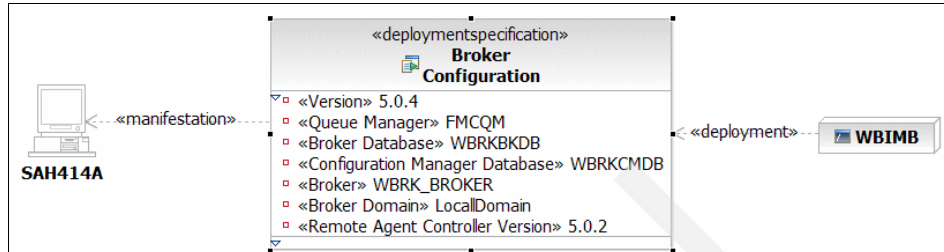
Figure 7-12   WebSphere Business Integration Message Broker configuration

The broker checks for prerequisites. You can find any missing installations on the Message Broker supplementary CD. Fixpack 4 needs to be installed after completing the broker installation. Fixpack 4 pre-reqs version 5.0.2 of the IBM Remote Agent Controller [RAC] which is also available on the same web page as the broker fixpack 4. Use the RAC from the Message Broker Web page rather than any other copy of RAC 5.0.2, to avoid an installation warning message later in the installation. However, as long as you are running RAC 5.0.2 the warning can be ignored.

To get Fixpack 4 and RAC 5.0.2, visit this Web site:

```
https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=wbimb&
S_PKG=dlwinww&cp=ISO-8859-1
```

You will need to sign on to http://www.ibm.com.

Broker configuration

Before starting, check that the user ID to be used by the broker has the right authorities. Check the groups that the user ID has rights to using the Windows user manager, or run the broker security wizard installed with WebSphere Business Integration Message Broker.

To configure the broker, we use the default configuration wizard in the WebSphere Business Integration Message Broker toolkit getting started page. See Figure 7-13 on page 232.
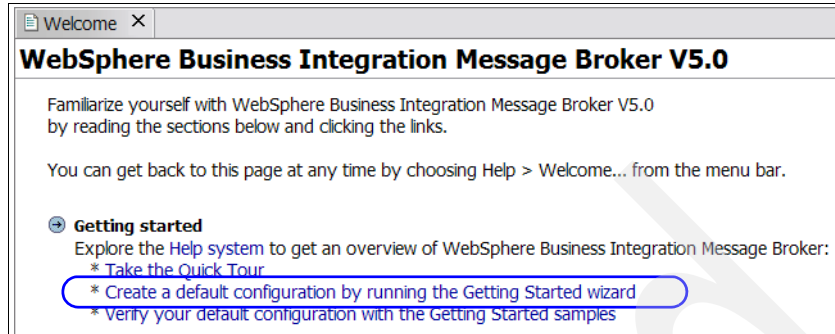
Figure 7-13 Finding the getting started wizard

Step-by-step instructions to create a configuration are provided in the information centre. It is worth reading them through before launching the getting started wizard so you understand it.

1. In our installation, we have used `Admin` as the main user ID and `db2admin` as the user ID to access the database. To copy this, deselect **Also use this account for accessing the DB2 databases** in the first panel (Figure 7-14).
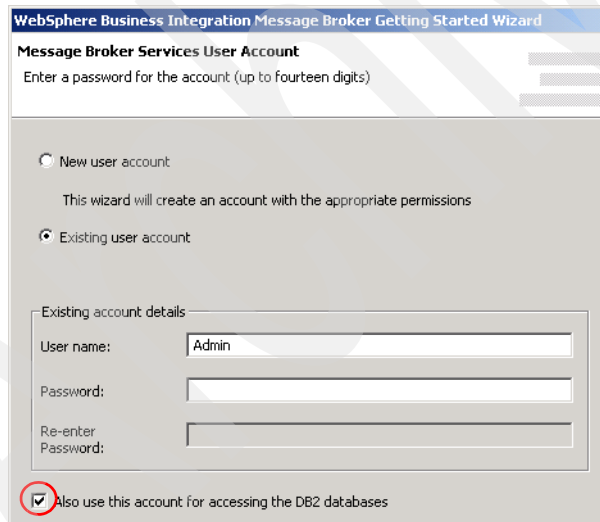
Figure 7-14 WebSphere Business Integration Message Broker getting started wizard

2. The wizard assumes that it is creating a new queue manager. If you are creating the same installation as we are, or if there is already a queue manager installed on the node you are configuring for the broker, this is not a big problem. When the wizard prompts for a queue manager name, give it the

name of the existing queue manager (`FMCQM`) and the listener port configured for it (5010).

When the wizard completes, there is a warning mark in the WebSphere MQ Alert Monitor (Figure 7-15).
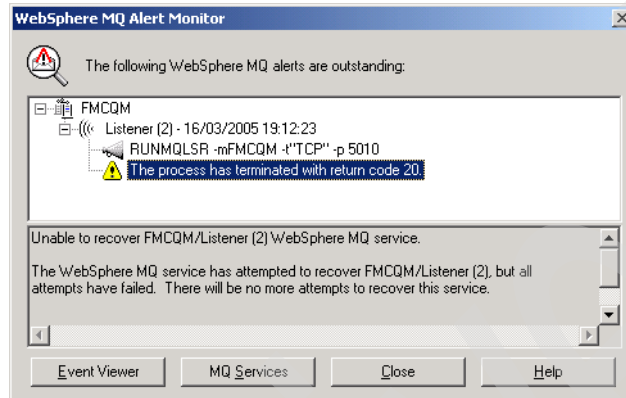
Figure 7-15   Warning in WebSphere MQ Alert Monitor

This is because the getting started wizard has added a listener to FMCQM on the same port as the existing listener. You need to delete the extra listener.

3. From **Alert Monitor** → **MQ Services** → **WebSphere MQ Services (local)** → **FMCQM** → in the right panel, highlight the second listener and right-click → **Delete** (Figure 7-16).
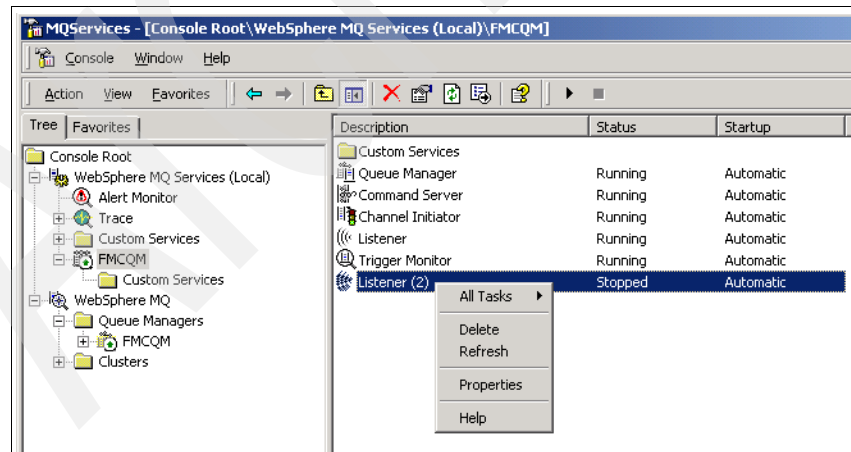
Figure 7-16   Deleting the extra service

4. Complete the initial broker configuration by changing the properties of the Broker WBRK_BROKER and the Configuration Manager services to start automatically in the Windows Services Manager. (**My Computer** → right-click → **Manage** → **Services** and find the two services to change).

   The default configuration sets the DB2 locking variable DB2_RR_TO_RS to YES which is recommended when using the Aggregation function in WebSphere Business Integration Message Broker. If you do not use the wizard you will need to set this variable in a DB2 command window:

   ```
   DB2SET DB2_RR_TO_RS=YES
   ```

5. The default configuration also sets the DBHEAP to 900. Check this in a DB2 command window by issuing DB2 get db CONFIGURATION for WBRKBKDB and verifying the DBHEAP size. If it is less than 900 set it using the command:

   ```
   db2 update database configuration for WBRKBKDB using dbheap 900[1]
   ```

Toolkit configuration

The toolkit is automatically configured by the getting started wizard. To use the broker toolkit running on a different machine. For example, using the native machine to host the toolkit, and connecting to the broker running in the VMWare image SAH414A, we need to install just the toolkit on the native machine, and configure it to attach to the broker running in SAH414A.

1. Define the user ID you are going to connect to in SAH414A and run the broker security wizard to give it authority to the broker groups.

2. In the WebSphere Business Integration Message Broker toolkit **File** → **New** → **Other** → **Broker Administration** → **Domain** → **Next**. Type `FMCQM` for the **Queue Manager Name**, `SAH414A` (or your name) for the **Host** and `5010` (or your port number) for the **Port** → **Next**. Change the **Server Project** to `LocalProject` and the **Connection name** to `LocalProject.` Click **Finish** → **YES.** These are the names for the folder that will hold the message flows and sets locally and the file that holds the connection information to the broker.

We have now completed installing and configuring the message broker and the toolkit ready to start developing the message flows in Chapter 9, "Build the Enterprise Service Bus" on page 261.

7.3 Install and Configure SAH414B

The environments to install on SAH414B are shown in Figure 7-17 on page 235.

---

[1] Note: DBHEAP not DHEAP (there is a typo in the information centre)
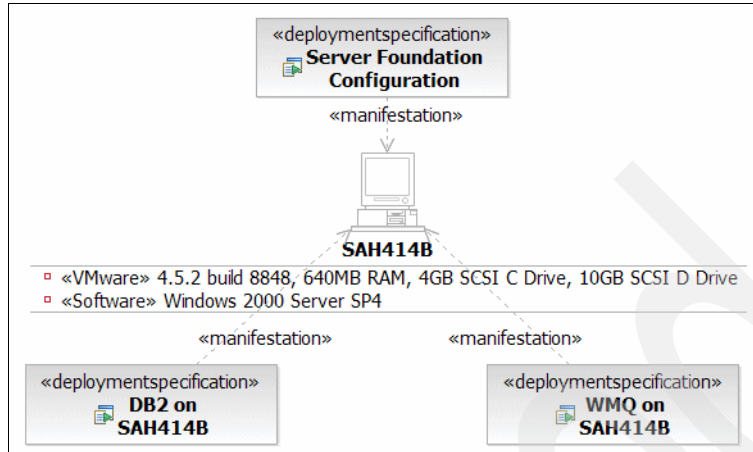
Figure 7-17   Software environments to install on SAH414B

Much of the installation is similar to SAH414A. We only note the differences.

7.3.1 WebSphere MQ

Install WebSphere MQ exactly the same way as on SAH414A.
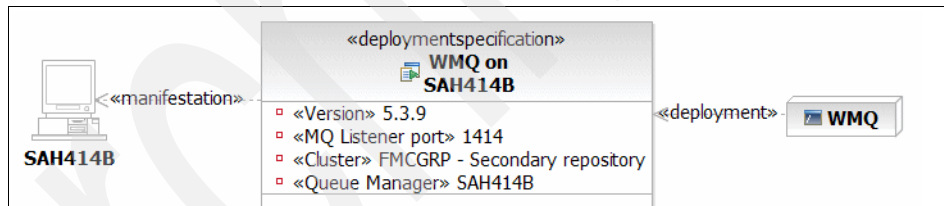
Figure 7-18   Deployment of WebSphere MQ on SAH414B

The next tasks are to define a queue manager and join the FMCGRP cluster on SAH414A. The wizards and first steps will not accomplish this because their assumptions and naming conventions do not match the solution. So we have to perform the task manually using the WebSphere MQ Explorer.

Create a Queue Manager

1. Right-click the **Queue Managers** folder in the WebSphere MQ Explorer →
   New → **Queue Manager.**

2. Step 1: Enter the Queue Manager name `SAH414B`, and Dead Letter Queue
   name `SYSTEM.DEAD.LETTER.QUEUE` as in Figure 7-19 on page 236. The Default

Transmission Queue is not essential as we are going to be using clustering to locate queues on other servers. Click **Next.**

Figure 7-19   WebSphere MQ configuration on SAH414B: Step 1

3. Step 2 → **Next** → Step 3 → Check the **Create Server Connection Channel option** → **Next** → Step 4 → specify 1414 as the **Listener** port → **Finish.**

Join the FMCGRP cluster

1. Start the SAH414A VMware image and check the IP connection to SAH414A by pinging it, and vice versa

2. Check that you can connect to the FMCGRP cluster.

   In the WebSphere MQ Explorer → right-click the **Cluster** folder → **Show Cluster ...** → Type FMCGRP as the **Cluster name** → check the **Repository Queue manager is remote** button → Type FMCQM as the **Queue Manager** and SAH414A(5010) as the **Connection Name** (Figure 7-20)

Figure 7-20   Show the FMCGRP cluster

3. Now have SAH414B join the cluster.

   a. Start the Add Queue Manager to Cluster Wizard. Right -lick the **Queue Managers in Cluster** folder under **FMCGRP** → **All Tasks...** → **Add Queue Manager** → **Next** and the wizard connects to the FMCGRP cluster.

   b. Step 1 → Select the queue manager is **Local** button → **Next.**

   c. Step 2 → **Next** → Accept the defaults in Step 2a for the name of the cluster receiver channel → **Next** → Accept the defaults in Step 2b for the repository's cluster receiver channel → **Next.**

   d. Step 3 → Add the queue manager to the cluster. Check the configuration → **Finish.**

   e. **Refresh** the Queue Managers in Cluster folder and SAH414B is shown in the cluster.

4. Clusters should always have at least two repositories. Making SAH414B a repository queue manager will sometimes improve its performance because it will become aware of configuration changes in the cluster before using them.

   a. Right-click **SAH414B** in the queue managers folder in the WebSphere MQ Explorer → **Properties** and select the **Repository** tab.

   b. Check the Repository for a cluster button and select FMCGRP in the cluster spin box.

We have now completed the basic configuration of WebSphere MQ. The queues and JMS configuration needed for the solution will be added in Chapter 12, "Integrate and test the business processes" on page 469.

7.3.2 DB/2

The installation of DB/2 is identical to SAH414A and is shown in Figure 7-21.
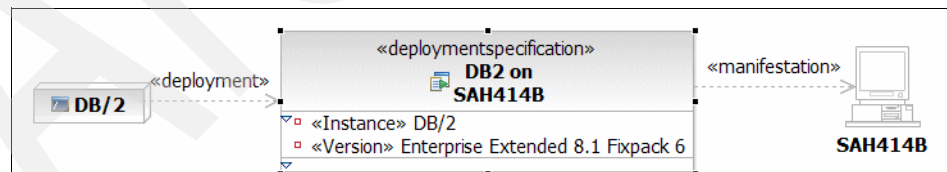
Figure 7-21   DB/2 installation on SAH414B

7.3.3 WebSphere Business Integration Server Foundation

To install WebSphere Business Integration Server Foundation successfully the first time you need to follow the sequence of steps carefully. Figure 7-22 on

page 238 shows the configuration we aim to complete preparatory to deploying the integration solution.
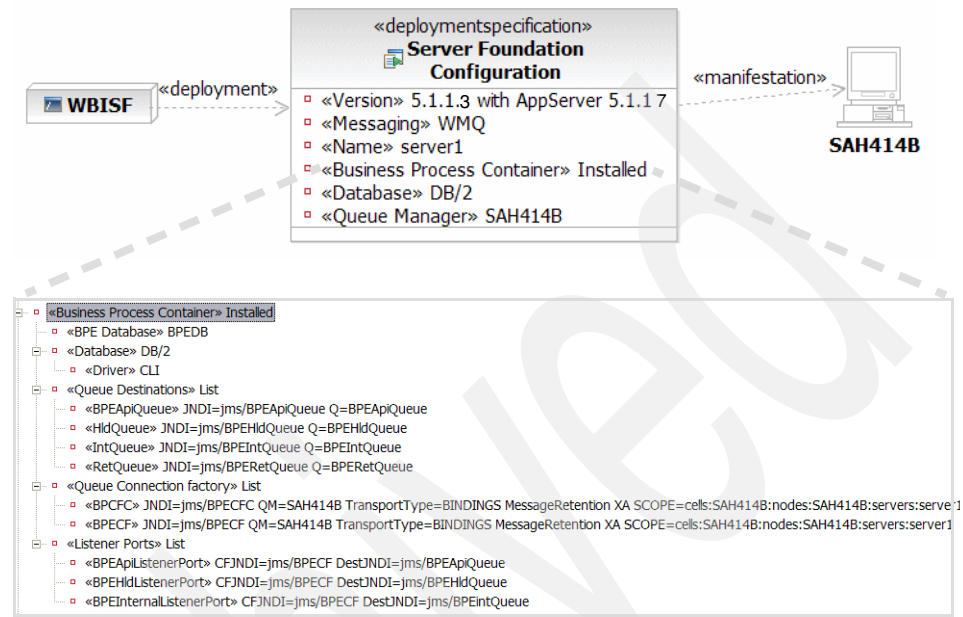
Figure 7-22   WebSphere Business Integration Server Foundation configuration

Server Installation

Follow these steps:

1. Install WebSphere Application Server 5.1 with no fixes. *Do not* install embedded messaging or the samples.

2. Install WebSphere Business Integration Server Foundation 5.1. *Do not* install embedded messaging or the samples.

   Install the Process Choreographer, but *do not* select the wizard to configure the business process container.

3. Install Fixpack 1 for WebSphere Application Server 5.1 (see 7.2.3, "WebSphere Application Server" on page 225.)

   This takes WebSphere Application Server to *5.1.1*

4. Install Fixpack 1 for WebSphere Business Integration Server Foundation following the same procedure as for WebSphere Application Server Fixpack 1.

   This takes WebSphere Business Integration Server Foundation to *5.1.1*

5. Install Cumulative fix 7[2] for WebSphere Application Server. This takes WebSphere Application Server To Version *5.1.1.7.*

6. Install Cumulative Fix 3 for WebSphere Business Integration Server Foundation. This takes WebSphere Business Integration Server Foundation to *5.1.1.3,*

7. Use **First Steps** as before, to start the server and verify the installation. Close all the windows leaving server1 running.

Configure the Business Process Container

We use a jacl script to configure the Business Process Container. Open a command prompt window and type:

```
D:\WebSphere\AppServer>bin\wsadmin.bat -f processchoreographer\sample\bpeconfig.jacl
```

This script prompts for all necessary information.

Table 7-1 lists the responses to the script for configuration of the process container.

Table 7-1   Configuration of the business process container

Prompt	Input	Comments
Install bpecontainer.ear	Yes	
Interactive Install:	No	We want to take all the defaults for more complex options during this install
Users to add to role BPESystemAdministrator:	Admin	This is the user id that will be used as the System Administrator.
Groups to add to role BPESystemAdministrator:	Enter	Just hit enter (don't type any text). That will take the default - we will not be adding any roles.
Run-as UserId for role JMSAPIUser[Admin]:	Enter	Take the default (note that Admin has been supplied as the default).
Administrator's password:	*******	Admin's password
Use a ... database:	DB2	We will be using DB2 as our runtime database. Note the number of other supported databases.

---

[2]   WebSphere Business Integration Server Foundation 5.1.1.3 pre-reqs WebSphere Application Server 5.1.1.7. WebSphere Business Integration Server Foundation 5.1.1 FP2 supports monitoring business events, while the earlier version of WebSphere Business Integration Server Foundation had a problem in this area.

Prompt	Input	Comments
Install processportal.ear	Yes	This will install the BPC Web Client component.
Interactive install:	No	Again we'll take the defaults.
Virtual Host for Web Client [default_host]:	Enter	
Node of Process choreographer to connect to [SAH414B]:	Enter	
Server of Process Choreographer to connect to [server1]:	Enter	
Create the Data Source for the Process Choreographer database:	Yes	
Database name [BPEDB]:	Enter	
Use the CLI or the Universal JDBC™ provider:	CLI	Using local bindings to the databases
DB2 user ID[db2admin]:	Enter	Or whichever Userid you have configured to access DB/2
db2admin's password:	*******	The password for DB2ADMIN
D:\Program Files ... \db2java.zip does not exist	D:\SQLLIB	Enter the install root for DB/2 - *not* the path to db2java.zip
Create the Process choreographer database?	yes	
DB2 table space directory:	Enter	
Use embedded messaging or MQSeries®?	MQSeries	
Create the Process Choreographer queue manager and queues?	No	We have already created them
Create the listener ports	Yes	This will configure WebSphere to use the queue manager we have already created
D:\Program Files\ ... java\lib does not exist	D:\WebSphere MQ	Enter the install root where WMQ is installed

Prompt	Input	Comments
Queue Manager name:	SAH414B	Don't take the default!
Will the Queue Manager join a WebSphere MQ cluster:	No	Although SAH414B is a member of a cluster, this refers to a special cluster configuration defined for Process Choreographer to do load balancing. So we say No.
Create the Scheduler for Process Choreographer:	Yes	
Enable global security using the Local OS user registry:	Yes	We need global security turned on for the staff component - although our current process doesn't use any staff nodes we will be adding some staff steps later and therefore need to configure for global security. Unfortunately the script not only enables global security but also Java 2 Security - which would prevent the CICS Resource Adapter from working. We will need to fix this later.
Server user ID[Admin]	Enter	
Enforce Java 2 Security	No	Not for testing
Set 'com.ibm.SOAP.loginUserid' in soap.client.props:	Yes	
Delete the temporary directory C:\tmp:	Yes	
Stop server server1 now:	Yes	Since we enabled global security we will need to restart the application server.

While the server is stopping, define the four additional queues that Process Choreographer requires. There is a .bat file to create the queues. In a command window, type:

```
D:\WebSphere\AppServer\ProcessChoreographer>createqueues SAH414B
```

When the server has stopped, start the WebSphere Application Server service again to verify the installation. Click the startup command in the WebSphere Application Server program group. It will notify you when the server is ready for e-business.

Open the enterprise application page in the console. Check if applications named BPEContainer_SAH414B_server1 and BPEWebClient_SAH414B_server1 are started. They are business process container and business process Web client which we have installed.

Figure 7-23   Process container and web client are enterprise applications

You have installed and configured business process container and process Web client successfully. And this ends all the infrastructure installation you need to do.

8

Test and deploy the application components

In this chapter we describe the implementation, deployment and test of the application components. The application components are not a main focus of the scenario, and this chapter just provides a brief description of the components and instructions on how to deploy the applications provided in the additional materials to support development and testing of the integration components.

243

8.1 The Assessor Automation System

As presented in Chapter 6, "Solution Architecture" on page 187, the Assessor Automation System comprises the parallel process engine that interacts with other existing and new components.

The following sections present:

► Interfaces that define the contract for these components
► Related implementation and functionality,
► Tips on configuring the packaged solution artefacts

8.1.1 Assessor Management System

The existing Assessor Management system is responsible for providing assessor-related information and analysis. The Assessor Management System is exposed as a Web service containing different operations, that range from identifying a list of potential assessors for a claim to administering assessor profiles. The administration of assessors involves both manual and automated steps depending on whether a delete assessor operation, which can only be done by an administrator is required, or if the assessment history and assessor score needs to be updated based on a current or recently completed claim.

While important, the administration of assessors is out of the scope of the Assessor automation system and in the section, we are focused on identifying a list of assessors for a claim, which is the only assessor management function used by the assessor automation system.

Identify Assessor List

The Assessor Management Service exposes an operation requestListAssessors that takes as input a requestListAssessorRequest message containing:

► claimID: The claim ID for the particular claim
► location: The zipcode or postcode indicating the location of the vehicle
► makeOfCar: A description of the make and model of the vehicle

Based on this, queries the assessor data store for a list of assessors within a 60-mile radius of the location of the vehicle, who specialize or have expertise with the particular make and model of the vehicle. The result of the query is prepared and returned as a requestListAssessorsResponse message that contains:

► claimID
► assessorList

This list is an array of potential assessors. Each assessor is an object containing the elements assessorID and assessorURL, where assessorID is a unique identifier for an assessor, and assessor URL, is the URI for that particular assessor's service.

The Assessor Management Service, and by extension the requestListAssessors operation, is a packaged J2EE application implemented with EJBs and runs WebSphere Application Server 5.1. The service interface is described and exchanged as WSDL that describes the Web service endpoint illustrated in Figure 8-1.
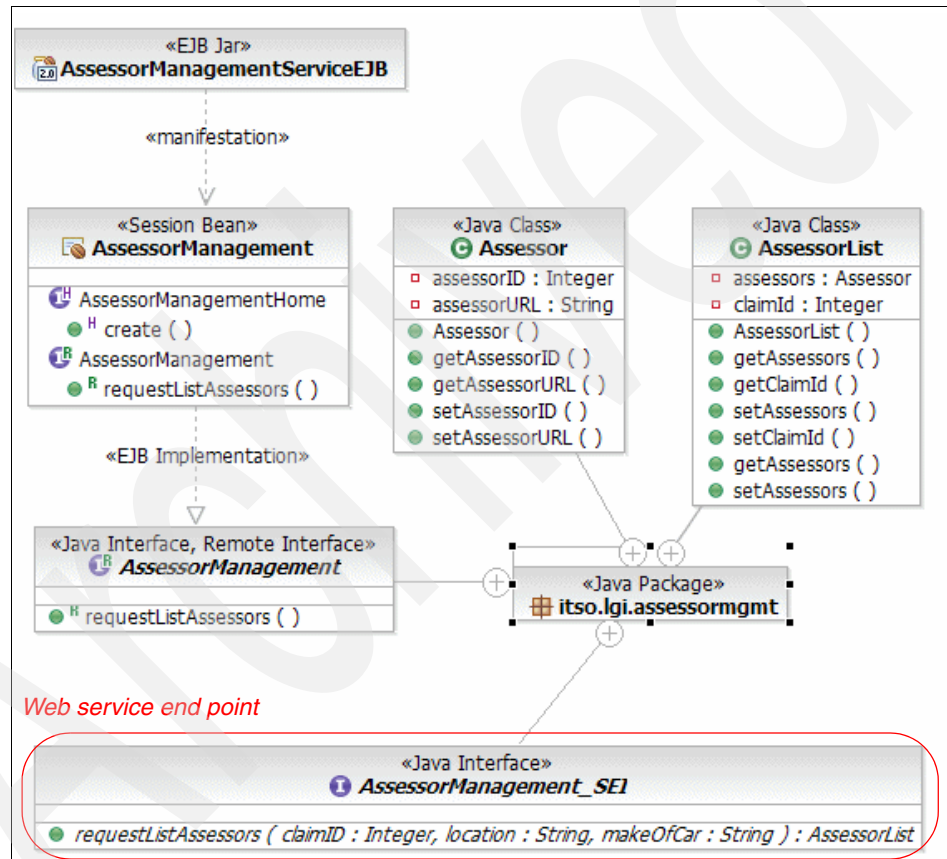
Figure 8-1   Visualization of the implementation of the Assessor Management Service

Verify Assessor Management in a test environment

To verify Assessor Management, follow these steps:

1. Create a new workspace folder for Rational Software Architect called `Applications` and open Rational Software Architect. In the **Window** →

Preferences → Workbench → Capabilities check **Advanced J2EE** and **Web services**. In the **Preferences → Web services → WS-I compliance** category you want to select WS-I compliance level to **Required**.

2. Open the J2EE perspective.

3. Import the Flow2_AssessorManagementService_SOURCE.ear from .\SG24-6636\WAS\Flow2 into Rational Software Architect and call it `AssessorManagementService`. See Figure 8-2.
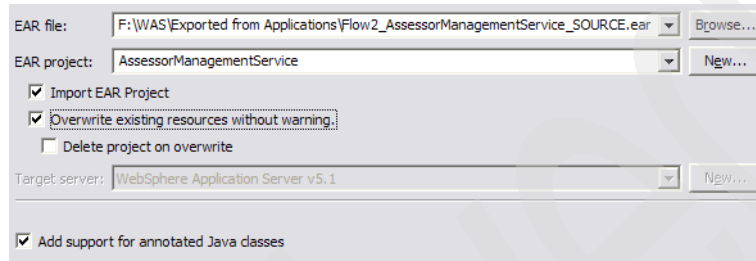
Figure 8-2   Importing the Assessor Management Service .ear file

4. Delete the default WebSphere Application Server 6.0 test environment in the Servers tab on the bottom right and then right- click → **New** and select the 5.1 test server environment.

5. Although the import of the .ear file has generated a WSDL file for you (Under **Web services → Services → AssessorManagementService → WSDL**) our experience is the Web service often needs to be regenerated to avoid a deployment problem with missing Web service classes:

   a. Open **EJB Projects → AssessorManagementServiceEJB → DeploymentDescription → Session Beans →** and right-click **Assessor Management → Web services → Create Web service.**

   b. In the dialog boxes that follow, you need to generate an EJB Web service and you do not need to test it. We have a different test method. Everything else in the dialog boxes is defaulted, but check that **requestListAssessors** method is selected when configuring the Java Bean as a Web service and use the Document/Literal style of Web service.

   c. In the Servers view, right-click the **5.1 test environment** and select **Add and Remove projects.** Ensure the AssessorManagementService is configured in the right hand box.

6. Again in the Servers view, publish and start the server, then check the console output. There should be no E level messages. If there are, they probably are to do with either missing Web service interfaces or there is something wrong with the deployment. Either redo the Web service wizard, or

remove the project from the 5.1 server and deploy it again. It comes right in the end!

7. Finally, test the Web service by launching the Web services explorer (Figure 8-3).
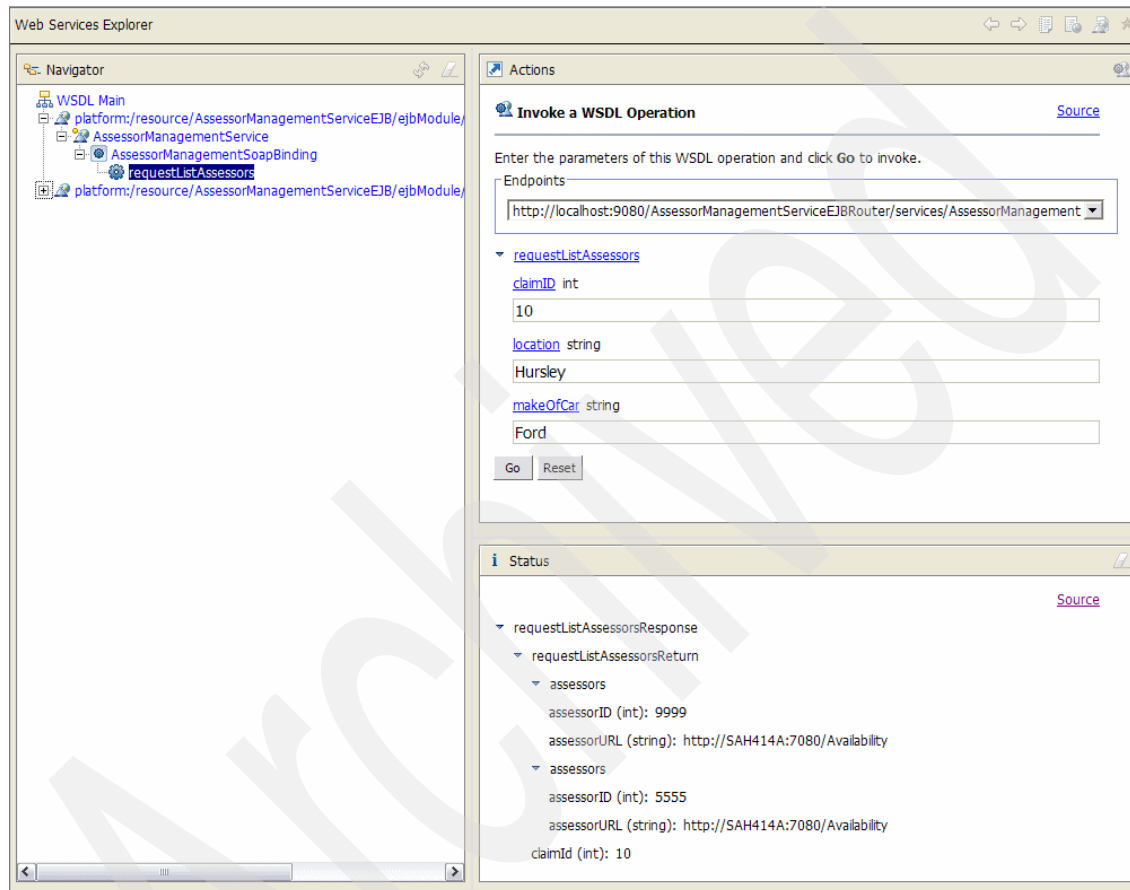
Figure 8-3   Testing the AssessorManagementService with the Web services explorer

a. Right-click the **AssessorManagement** WSDL file → **Test with Web services explorer.**

b. Click **requestListAssessors** in the Explorer navigator and type in correctly types data into the input fields (e.g. 10, Hursley, Ford).

c. The results are either an assessor number, 9999, or 5555 back in a list.

8.1.2 Business Rules Engine

WebSphere Process Server Version 6.0 provides a customizable Business Rules Engine. The Business Rules Engine component holds all business rule functions needed for the claims processing. The Business Rules engine is not a business rules product, but rather a collection of applications that apply rules in different situations and are implemented in a mix of EJBs, Business Rule Beans, and COBOL applications. For the Assessor Automation system, our focus is on two functions the Business Rules Engine provides, policyRules and assessorRules.

Policy Rules

Policy Rules refers to the function that provides policy-based metrics for processing a claim. For instance, in Assessor Automation it is important to provide how much time an assessor can take to do the work, and how much time the assessor has to bid for the work based on the policy profile. For example, a client with a premium insurance policy will expect a fast processing of the claim. A client with a third-party only policy is not hampered if the processing of the claim can take slightly longer. The interactions with the assessors need to reflect this.

The RequestResponseTimePTService provides the function to determine the duration to wait for assessors to bid for the work by tendering their availability and predicted cost. The RequestResponseTimePTService takes as input a RequestResponseTimeRequest message containing:

► claimID:
► policyID: A unique identifier of the clients insurance policy

This information is used to query the policy type based on policyID, calculate duration and response time based on predefined rules around policy type, and return a RequestResponseTimeResponse message that contains:

► claimID
► responseTime: duration in hours for when an availability response must be set. The duration is added to the time the requests are sent out to give the responseTime.

The visualization of the session bean is shown in Figure 8-4 on page 249. The procedure for testing it is the same as for the assessor management services.
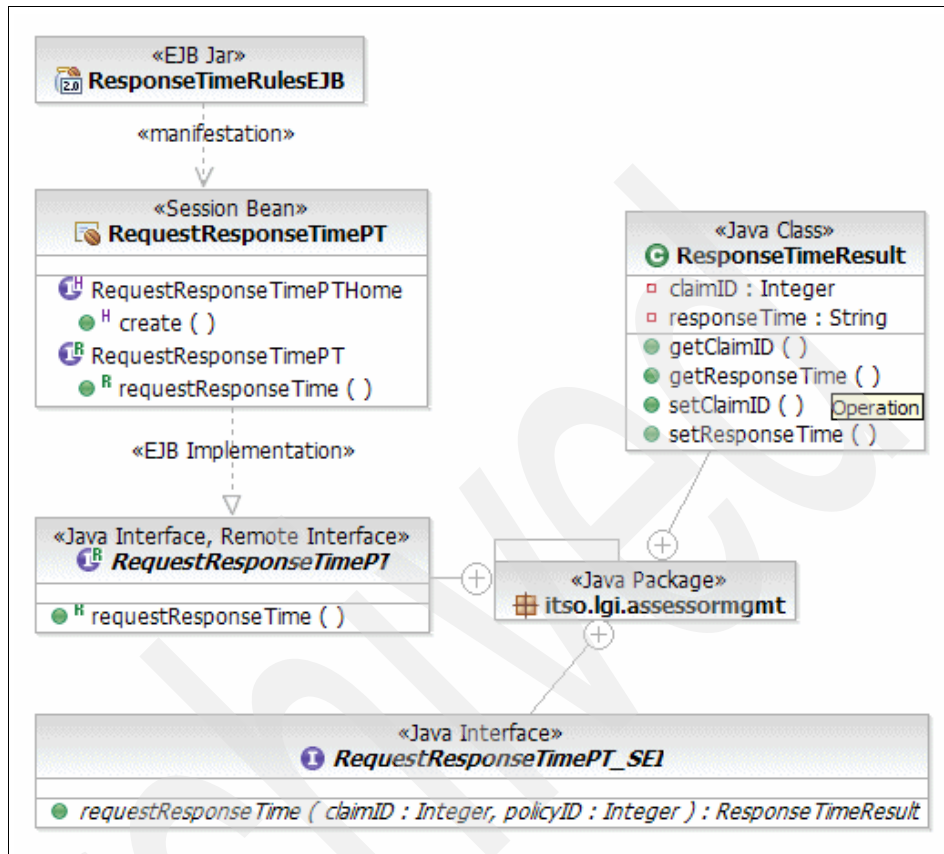
Figure 8-4   Visualization of the ResponseTimeRules implementation

Assessor Rules

When a list of potential assessors to do the assessment have been identified, the best assessor for the job has to be selected. This function is handled by the PreferredAssessorService where the assessors are prioritized, based on the predicted completion date tendered by the assessor, the performance of the assessor retrieved from historical data in the assessors profile, cost, the distance of the assessor to the vehicle, and assessor specialization in different types of vehicle.

The PreferredAssessorService takes as input a SelectAssessorRequest message which contains:

► claimID

► AssessorAvailabilityList: An array of available assessors. Each available assessor is an AssessorAvailability object with fields, assessorID,

assessorURL, predCost, predDate, where predCost is the predicted cost of the assessment and predDate is the predicted date of assessment completion

The serves returns a SelectAssessorResponse message containing these elements:

► claimID
► assessorID.

The visualization of the PreferredAssessor service is shown in Figure 8-5below:
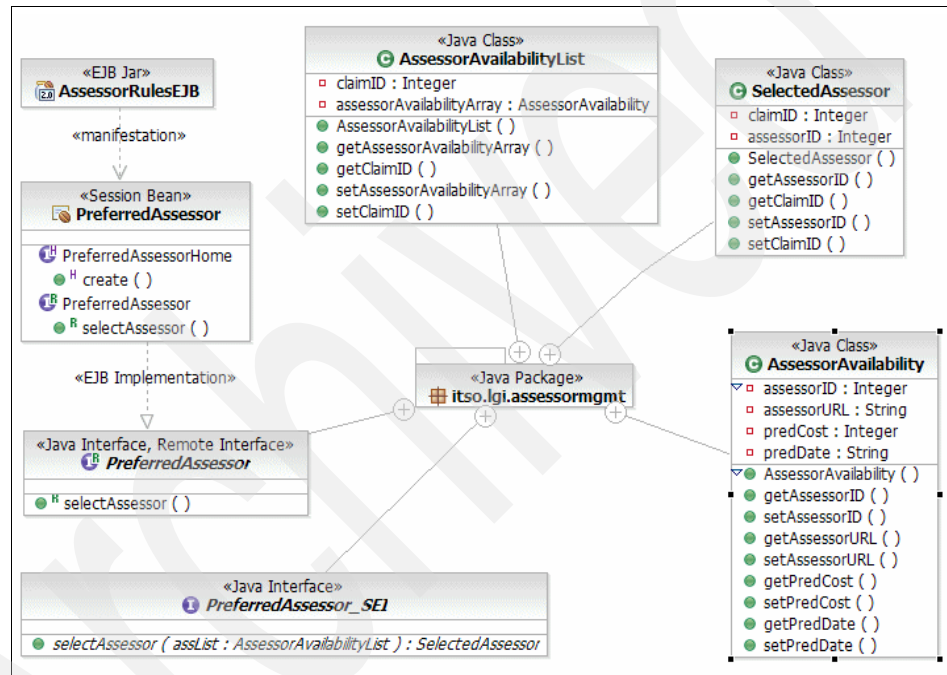
Figure 8-5   Visualization of the PreferredAssessor implementation

8.1.3 Document Management System

IBM provides a ready built Document Management System for DB/2 that integrates into many popular document clients, such as Microsoft Word, and provides extensive team and project capabilities that can be customized with no programming. See the following Web site for further information:

    http://www.ibm.com/software/data/cm/docmgr/

The centralized processing and storage of claims related documents including Medical Reports, Police Reports, Assessment Reports, is handled by the Document Handler System.

StoreAssessorReport

For the Assessor Automation System, we are particularly concerned with the StoreAssessorReportService that receives an assessment Report from an assessor, stores the report in a dedicated file system, and sends the url of the report back to the requester of the service, in this case, the <<automated external claim assessor business process>>. The StoreAssessorReportService takes as input a StoreAssessorReportRequest message that contains four elements:

► claimID
► assessorID
► xmlReport: An xml document representing the report from the assessor
► completionDate: the date the assessor completed the assessment and report.

The service returns a StoreAssessorReportResponse message that contains:

► claimID
► assessorID
► reportLocation: The url for the location of the stored document.

The StoreAssessorReportService is implemented as a SOAP/Http service running in WebSphere Application Server version 5.1. See Figure 8-6.
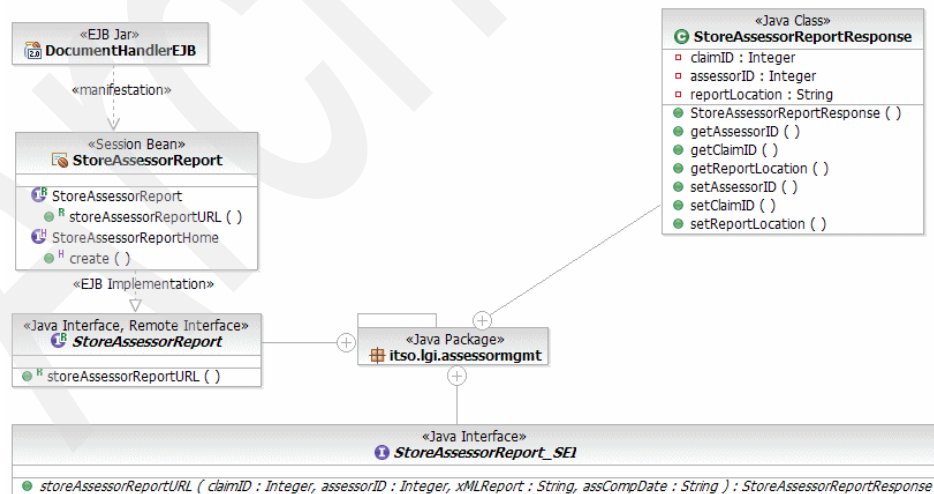
Figure 8-6   Visualization of the Store Assessment Report implementation

8.2 External Assessor System

The implementation of the External Assessor System is decided by the Assessors themselves. LGI provide the WSDL interface to the services to be hosted by the assessors, and the WSDLs for the services for which the Assessors need to provide clients. These WSDLs are listed in Table 8-1.

Table 8-1   Services comprising the LGI/Assessor boundary

WSDL	Hosted by	Description
Availability(4).wsdl	Assessor	Request Availability
AssessorAvailabilityPT(4a).wsdl	LGI	Response
DeliverAssessment(7).wsd	Assessor	Request Assessment
DeliverAssessmentResponse(7a).wsdl	LGI	Acknowledge
AssessorReport(8).wsdl	LGI	Send report

In the additional materials, an Assessor implementation is provided for WebSphere Application Server and for WebSphere Business Integration Message Broker. The rather unlikely choice of creating a message broker implementation is that it is very easy to build and test with the broker flows that are used to implement the proxyAssessorSystem.

One distinct advantage in the broker implementation is that it is easy to control the delay between sending flow 7a and flow 8 back to the broker using the MQ Explorer to enable and disable message queues.

Assessor Availability
In Figure 8-7 on page 253 you can see the interface to requestAssessorAvailability requires the carDetails and a number of other parameters to do with the claim.
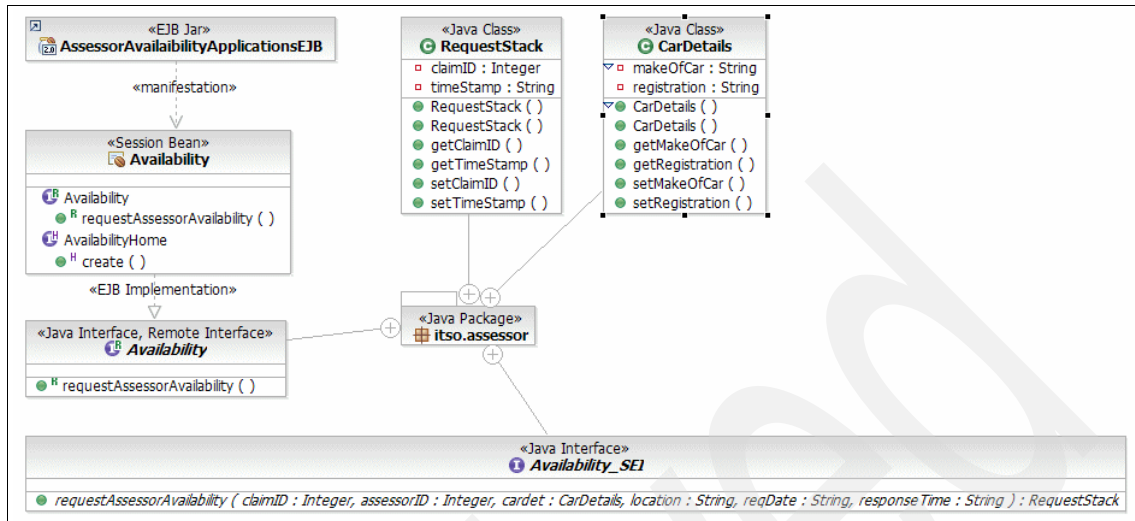
Figure 8-7   Visualization of AssessorAvailabiliy implementation

Unlike the earlier Web services, the SOAP response is simply an acknowledgement. The actual availability is passed as a return call.

Deliver Assessment

The Deliver assessment application receives the request for the selected assessor to perform the vehicle assessment. It returns a simple acknowledgement to the request like the Assessor Availability application.

See Figure 8-8 on page 254.
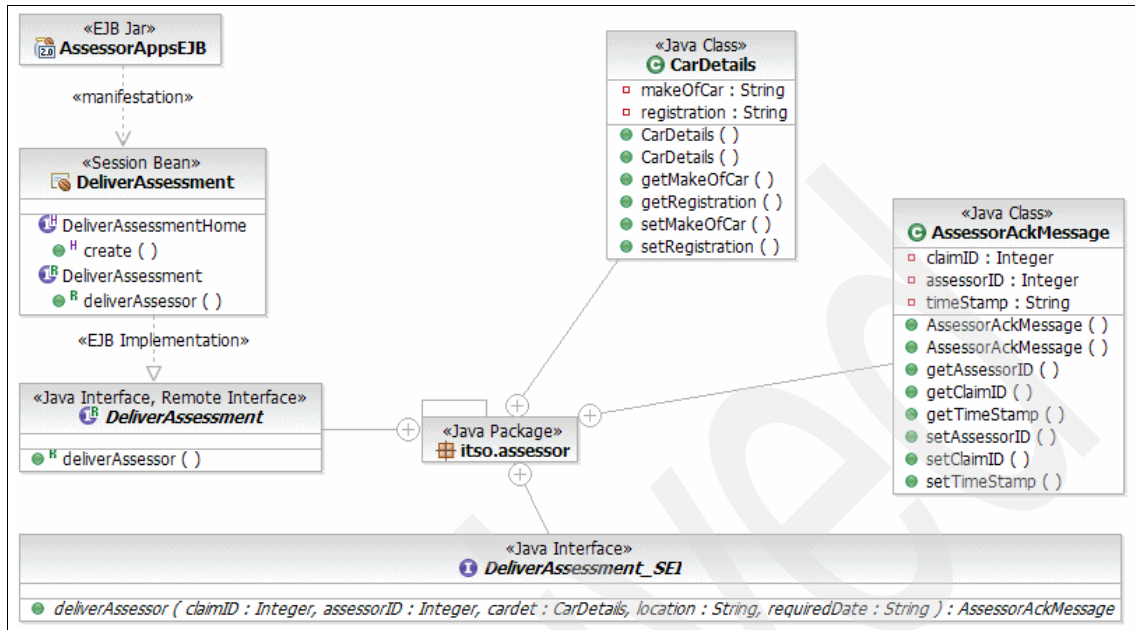
Figure 8-8   Visualization of the Deliver Assessment application

The interaction which returns the decision to either commit to doing the assessment or not is sent in the next interaction by the DeliverAssessorResponse client application. The Assessment report client then follows this by sending the assessment report as shown in Figure 8-9 on page 255.
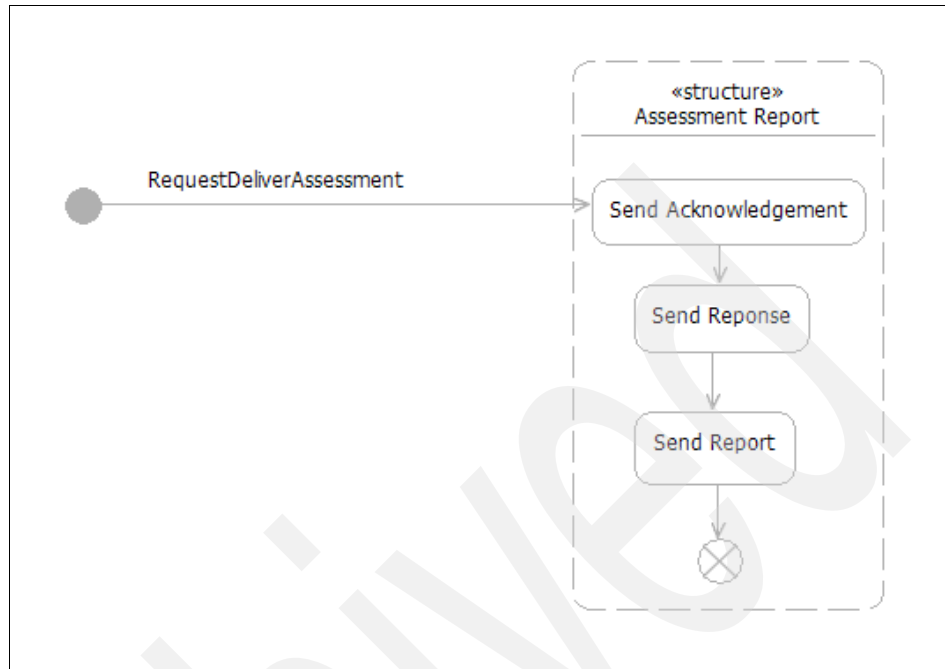
Figure 8-9   Visualization of Assessors activities

Assessor Client Applications

The client applications provide the assessors responses back to LGI. The
services are hosted by the proxyAssessorSystem. Figure 8-10 shows the client
interfaces of the three applications.
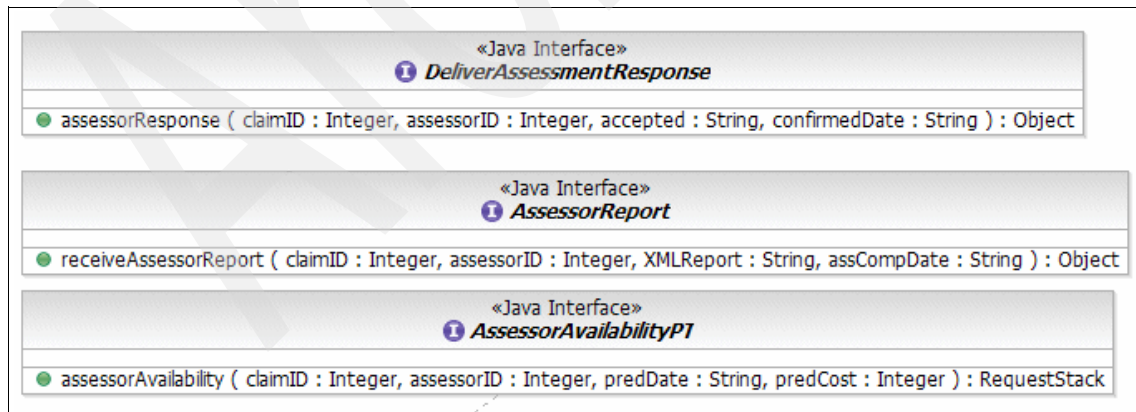
Figure 8-10   Visualization of Assessor Client Application interfaces

8.3 Deploy and test application components

Figure 8-11, constructed from Table 6-2 on page 198 shows the .ear files to be deployed. All the EJBs have been deployed to a single WebSphere Application Server to simplify the configuration.
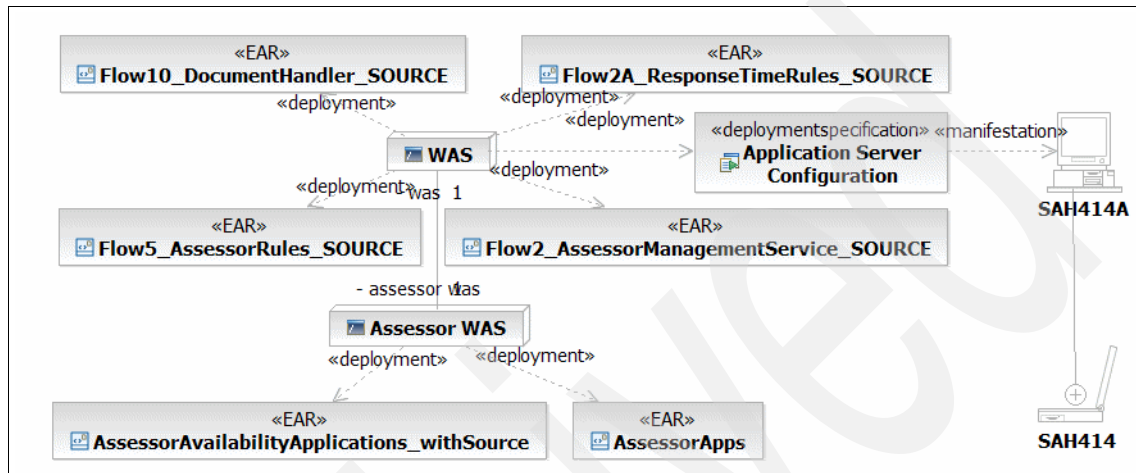
Figure 8-11   Application deployment

8.3.1 Deploying to WebSphere Application Server

To deploy the EAR files to WebSphere Application Server on SAH414A, open the administration console of the Application Server. It is easiest to open the console on the machine which has the EAR files and the console browser will upload them to the application server.

Start with Figure 8-12 on page 257.
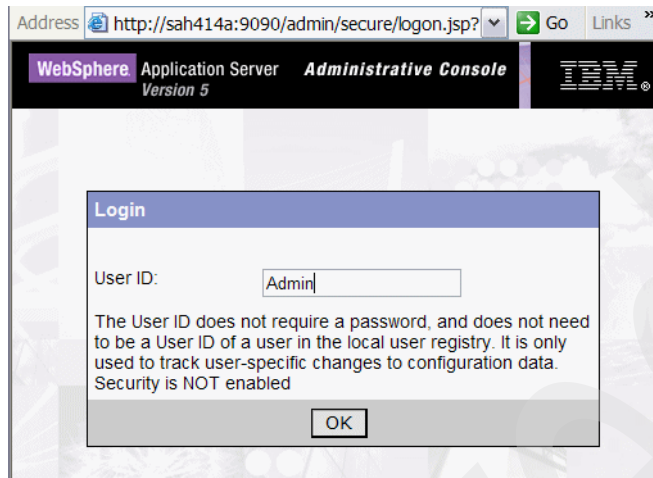
Figure 8-12   Logging into WAS Administration console remotely

1. On the left navigation panel, choose **Install New Applications,** and on the right enter the path to the .ear file to install (AssessorApps.ear in this example). Click **Next** (Figure 8-13)**.**
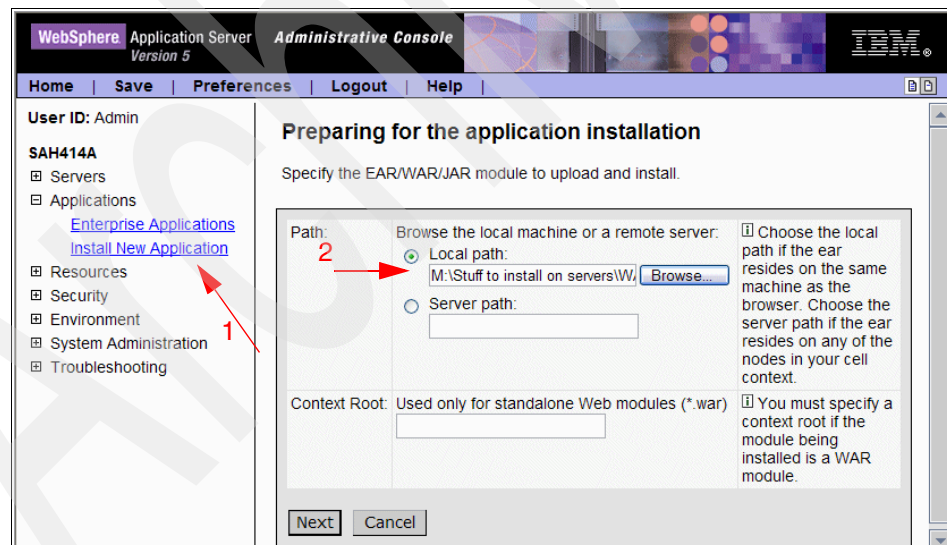
Figure 8-13   Select AssessorApps to install

2. On the next panel (Figure 8-14 on page 258) check default bindings as we are not going to change any of the default settings → **Next.**

Figure 8-14 Accept default bindings

3. On the next panel (Figure 8-15) jump straight to **Step 6 Summary.**

Figure 8-15 Jump over configuration steps

4. On the summary panel, click **Finish** and Save the changes to the master configuration (Figure 8-16).

Figure 8-16 Save to master configuration

5. Remember to confirm on the following panel in Figure 8-17 on page 259.
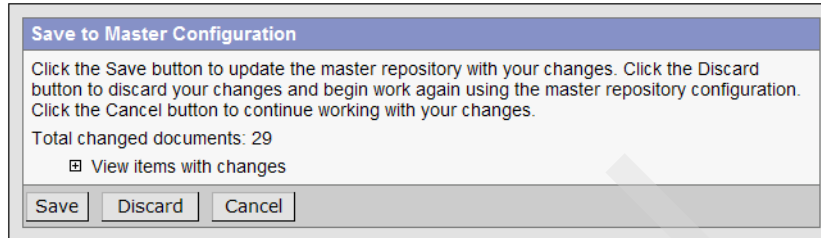
Figure 8-17   Save confirmation

6. To start the newly installed application, on the left navigator panel click **Enterprise Applications.** On the right-hand panel, check **AssessorApps** → **Start** (Figure 8-18).
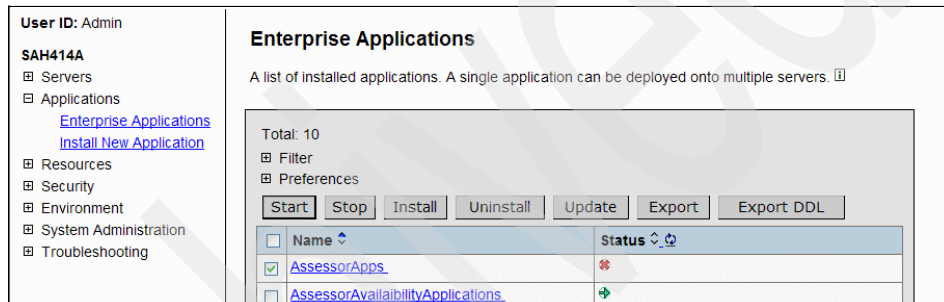
Figure 8-18   Start AssessorApps

7. Repeat this process for the remaining five applications.

8.3.2 Testing the deployed applications

To test the deployed applications, rerun the Web services explorer and change the address of the Web service to the SAH414A server.

In Figure 8-20 on page 260 be sure to check the box next to the new service and click **Go** to add the endpoint to the list of service endpoints. Select the endpoint to invoke the Web service as in Figure 8-19.
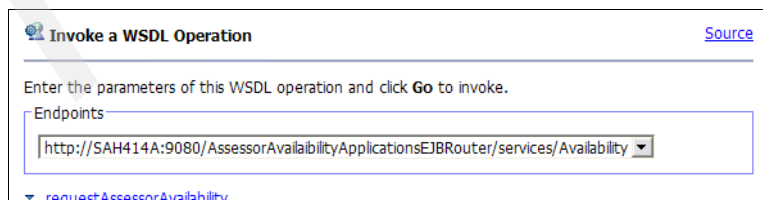
Figure 8-19   Invoking Assessor Availability on SAH414A
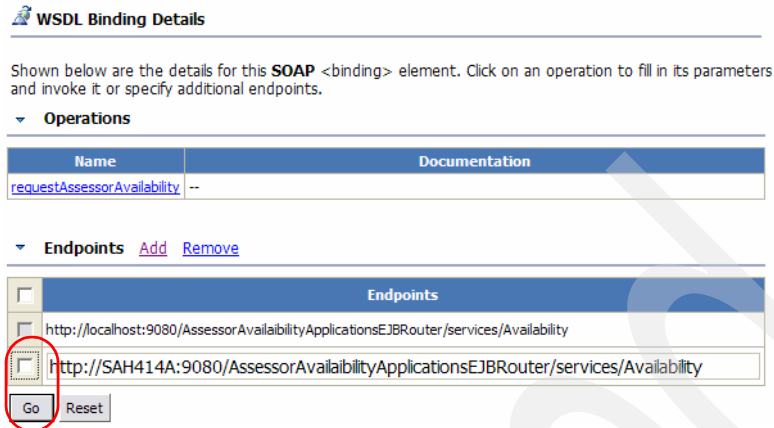
Figure 8-20   Adding SOAP endpoint address in the Web services explorer.

8.4 Summary

In this chapter, we have described the applications used by the RequestExternalReports process, and demonstrated how to test and deploy the applications from the additional materials supplied with this redbook.

9

Build the Enterprise Service Bus

This chapter describes how to build the broker components of the enterprise service bus.

The position of the ESB in the solution architecture and the capability required of the broker are described first. The next section 9.2, "WebSphere Business Integration Message Broker" on page 266 briefly describes the concept of a message broker and its key components. The design of the solution components is described in 9.3, "Component Design" on page 274. The next four sections describe the implementation in detail:

- ► 9.4, "Implementation of the message sets" on page 289
- ► 9.5, "Implementation of the database tables" on page 314
- ► 9.6, "Create the message flows" on page 319
- ► 9.7, "Create the ESQL code for the message flows" on page 351
- ► 9.8, "Deploy message set and flows" on page 365 describes how to create an Broker archive file and deploy it to the broker runtime.

Finally, we describe how to test and debug the message flow.

9.1 Architecture

Here, we take a moment to recap where the ESB fits into the system and solution architecture.

System Architecture

Figure 9-1 shows the system architecture for the External Claim Assessor solution. Recalling the discussion in Chapter 5, "System Architecture" on page 153, for Version 5 of the WebSphere platform we decided to limit the ESB to the Extended Enterprise pattern, using process integration for the Application Integration pattern. The Extended Enterprise pattern is responsible for the flows to and from the external claim assessors. In version 6 of the WebSphere platform, we intend to extend the ESB to include all the components in the solution using robust connections between all the services.
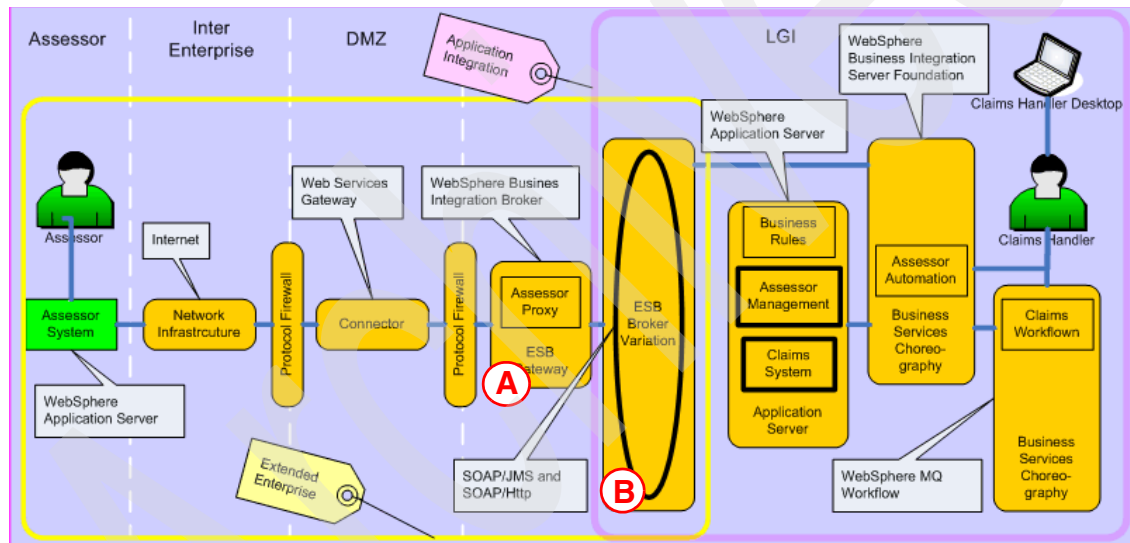
Figure 9-1   External Claim Assessor::Product Mapping

The task for this chapter is to implement the router and aggregation services and to connect the LGI zone of the solution with the Assessors. Again, purely for resource reasons, we will not focus on the implementing the Web services gateway, security, or other types of connection to the assessors such as implementing a browser, EDI, file transfer, email, fax, or interoperability of Web services.

Examining Figure 9-1 on page 262 more closely, we use the message broker to perform two functions corresponding to the boxes labelled A and B.

1. It provides specific routing, distribution, and aggregation services for the ESB.

   a. It is responsible for creating an address to route requests to individual assessors based on the protocol, quality of service and destination of the assessor, and placing the request on the appropriate transport connection. Where the native connection technology being used cannot provide the quality of service required then the broker is responsible for making a best effort.

      The connection technology used by the broker will almost certainly retranslate the destination address. This could be the Web services gateway proxying the address, for maintainability and security reasons, or DNS converting a host name into an IP address. The broker is not solely responsible for routing; it needs to cooperate with the services provided by the connection technology. What it does do is provide a stable services interface for components whether they are implemented as services themselves, or by being wrappered as services by the ESB.

   b. One of the sets of capabilities offered by an ESB is mediation. Mediation provides flexibility in the logical to physical mapping for service interfaces. Mediation can be as simple as rearranging the order of parameters offered by a service, or as complex as providing data augmentation, data conversion, and type conversion. The more sophisticated forms mediation we require are one-to-many and many-to-one mapping, or distribution and aggregation.
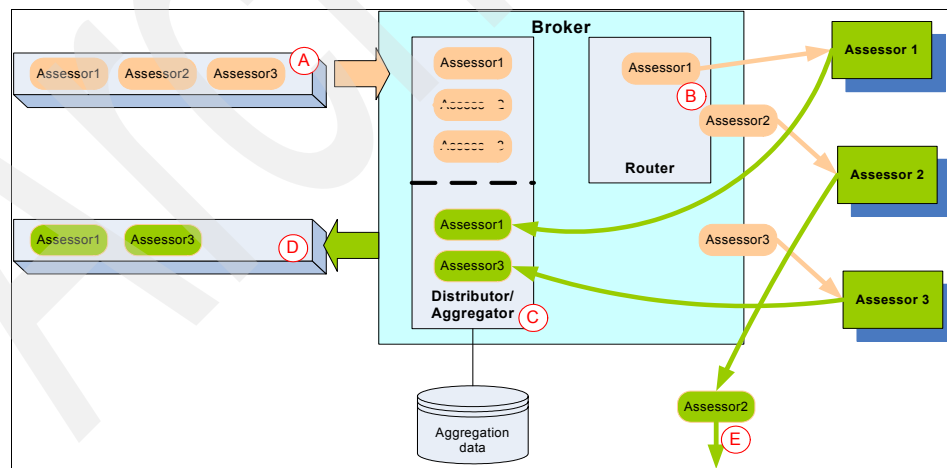
Figure 9-2   Schematic view of distribution, aggregation, and routing in the broker

The request for assessment is received (A) from the RequestExternalAssessment process as a single service request containing a list of potential assessors. The list is broken up into separate requests which are sent out to a individual assessors (B). Over a period of time the responses are returned (C). When the contractually agreed time to response elapses, the broker aggregates the responses it has received and invokes a response service provided by the RequestExternalAssessment process passing a single list of responses from the :assessors (D). Late arriving responses are discarded (E).

2. The second capability is to provide a bus for different kinds of transport. WebSphere Business Integration Message Broker provides a number of communication protocols - we are going to offer web services on SOAP/Http and SOAP/JMS over WebSphere MQ. The design of the flows in the broker will enable services to connect using either protocol.

Solution Architecture

The sequence diagram shown in Figure 9-3 on page 265 shows interactions with the ESB proxyAssessorSystem component. The interactions in red represent services the proxyAssessorComponent is providing.
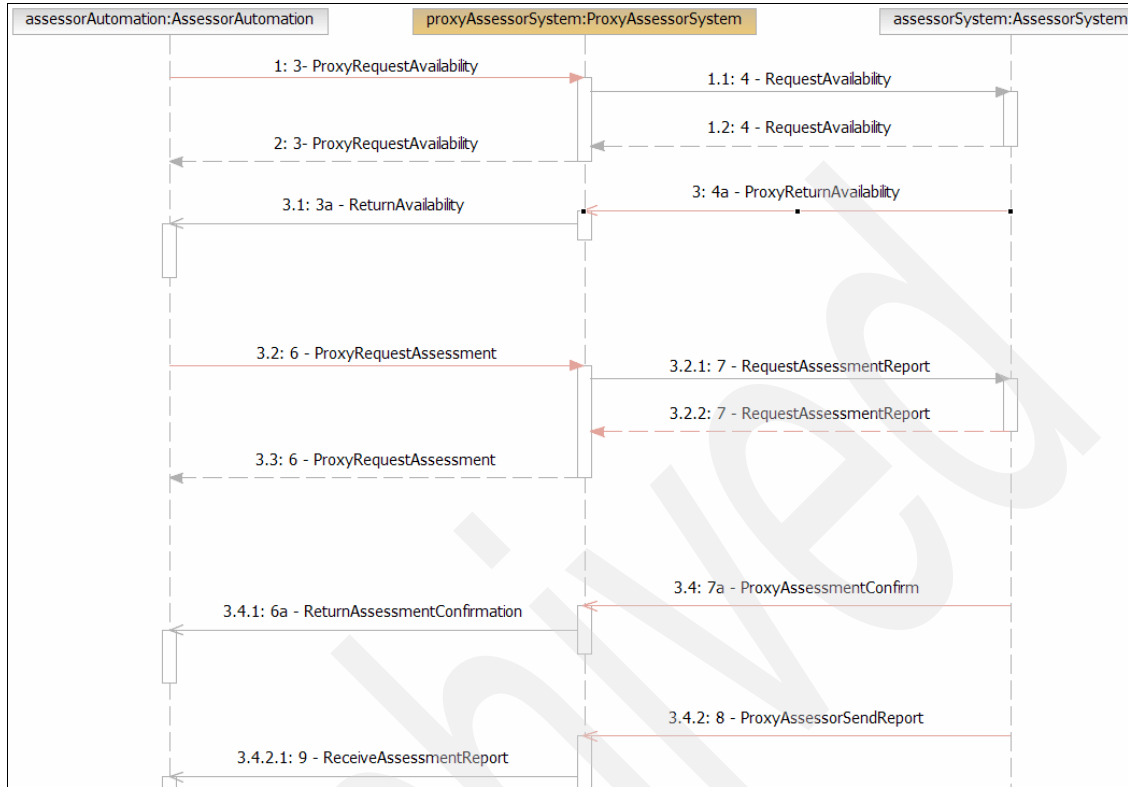
Figure 9-3   proxyAssessorAutomation sequence diagram

The interactions are described in detail in Table 6-1 on page 189. Table 9-1 (adapted from Table 6-2 on page 198) summarizes the interactions with the proxyAssessorAutomation system and identifies the WSDL files that define the services. The proxyAssessorSystem is noted where it is responsible for providing the service. Service flows need to be developed for the proxyAssessorSystem interactions and client flows for the others.

Table 9-1   WSDL file details

Flow	Component	Interface details - WSDL file name
3	Proxy Assessor System	AssessorAvailability(3).wsdl
4	Assessor	Availability(4).wsdl
4a	Proxy Assessor System	AssessorAvailabilityPT(4a).wsdl
3a	Assessor Automation	AssessorAvailablityList(3a).wsdl
6	Proxy Assessor System	AllocateAssessmentReport(6).wsdl

Flow	Component	Interface details - WSDL file name
7	Assessor	DeliverAssessment(7).wsdl
7a	Proxy Assessor System	DeliverAssessmentResponse(7a).wsdl
6a	Assessor Automation	AllocateAssessorResponse(6a).wsdl
8	Proxy Assessor System	AssessorReport(8).wsdl
9	Assessor Automation	AssessorReport(9).wsdl

9.2 WebSphere Business Integration Message Broker

WebSphere Business Integration Message Broker is the message brokering product within the WebSphere Business Integration family of products. This family consists of a number of server products that perform integration roles at different levels.

The broker product belongs to the level of Application Connectivity Services within the WebSphere Business Integration architecture as shown in Figure 9-4 on page 267. This layer performs functions such as routing, mediation, transformation and publish/subscribe. These functions are typically performed on top of a messaging infrastructure, such as WebSphere MQ.
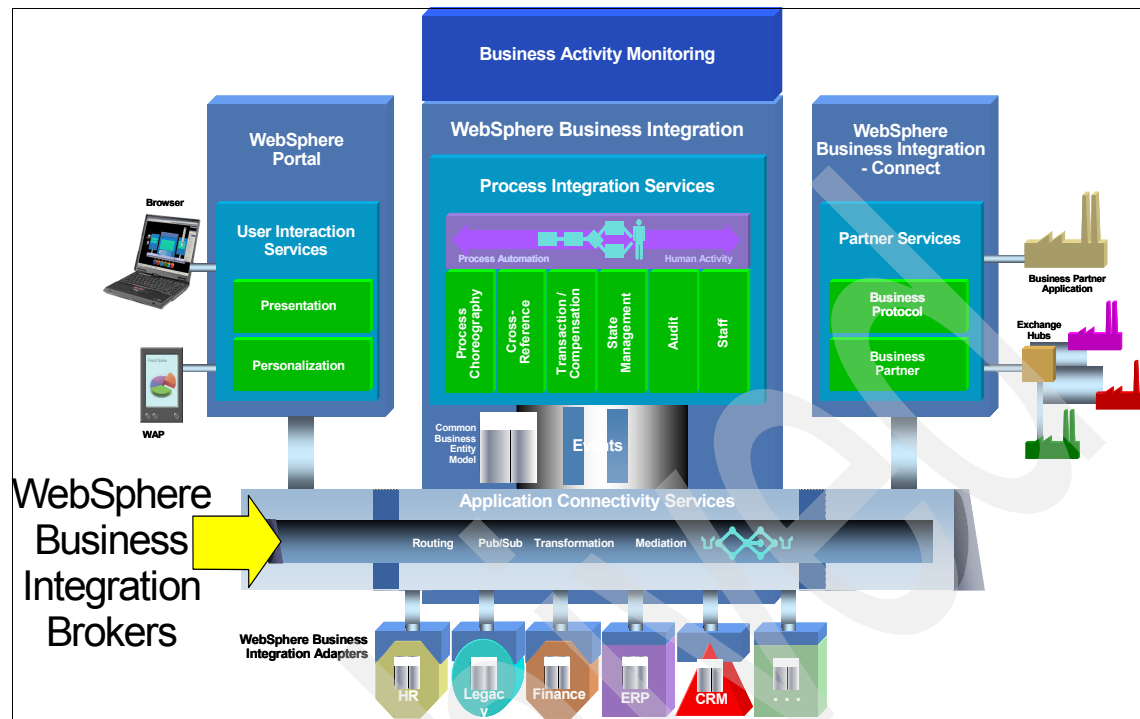
Figure 9-4   WebSphere Business Integration reference architecture

With the change towards service-oriented rather than message-oriented architectures, the broker has a new role. It bridges between the traditional methods of integrating applications using adapters and message-oriented middleware and the emerging world of services integration using enterprise service buses. Starting with Version 5, the broker has offered run time integration with SOAP/Http transport.

The flexibility of the messaging model and the broker's tooling means that SOAP services can provided and consumed by broker message flows. It is this capability we shall use to provide SOAP services over http which can easily be adapted to JMS.

9.2.1 Components of message broker

WebSphere Business Integration Message Broker V5 consists of the following components:

► Message Brokers Toolkit for WebSphere Studio, which is a new feature replaces the Control Center of WebSphere MQ Integrator V2.1, as well as adding extended functionality in many new areas.

- Configuration Manager contains a repository for configurations and messages.
- Message brokers, the runtime component consisting of execution groups which contain deployed message flows.
- Publish/Subscribe functionality.
- WebSphere WebSphere MQ queue managers, which provide the underlying transport infrastructure for the WebSphere Business Integration Message Broker.
- User application programs, which generate messages and request them to be transformed and routed to other destinations in a messaging infrastructure, according to specific business rules.

Figure 9-5 shows how these components work together.

Figure 9-5   Components and structure of a broker domain

Broker Workbench

The IT integration specialist uses the Eclipse based broker toolkit to create and save solution components in a local workspace or in a version control system.

The broker toolkit consists of a number of plug-ins for WebSphere Studio. These plug-ins can be loaded in a separate WebSphere Studio installation, or you can use the WebSphere Studio runtime that is shipped with the WebSphere Business Integration Message Broker product CD.

Configuration manager

To deploy solutions, the IT specialist stores completed message sets and message flows in the configuration manager's repository, which is a DB2 database. They can then be deployed to one or more brokers. Brokers, which are the runtime component, are available on a number of platforms, including:

- ▶ Windows
- ▶ Several versions of UNIX®, including Linux
- ▶ z/OS

A collection of brokers assigned or connected to a given configuration manager is called a *broker domain*. From within the workbench, you can configure connections to one or more broker domains, for example to test, development and production domains.
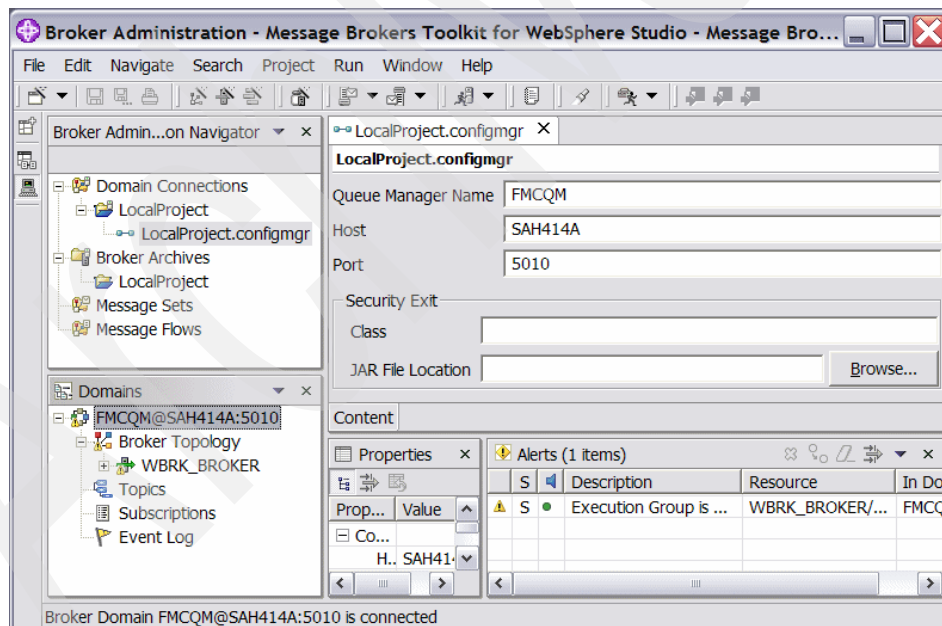
Figure 9-6   Toolkit connected to a broker domain

In Figure 9-6 there is only one broker domain listed under Domain Connections. The Domain Connection Editor shows the information to connect to the WBRK_BROKER we configured in 7.2.5, "Install and configure the Message

Broker" on page 230. The broker domain connection consists of an WebSphere MQ client connection to the queue manager that sits underneath the configuration manager. The broker domain for our solution consists of the configuration manager and only one broker, along with their corresponding databases.

When the broker administrator uses the workbench to deploy a solution to a broker in a particular domain, the workbench interacts with the configuration manager using the connection for that domain. The configuration manager is responsible for synchronizing configurations with brokers. The configuration manager will send the new solution artifacts with a deployment command to the to the broker using WebSphere MQ.

Broker

A broker itself consists of a number of components, as shown in Figure 9-7. A broker process controls and monitors a number of processes, each called a DataFlowEngine. A DataFlowEngine corresponds with an administrative Execution group. An execution group can process multiple deployed message flows at the same time, corresponding to a multi-threaded execution model. Multiple execution groups can run in a single broker, and can be managed separately such as deployed, started, stopped, removed and so forth.
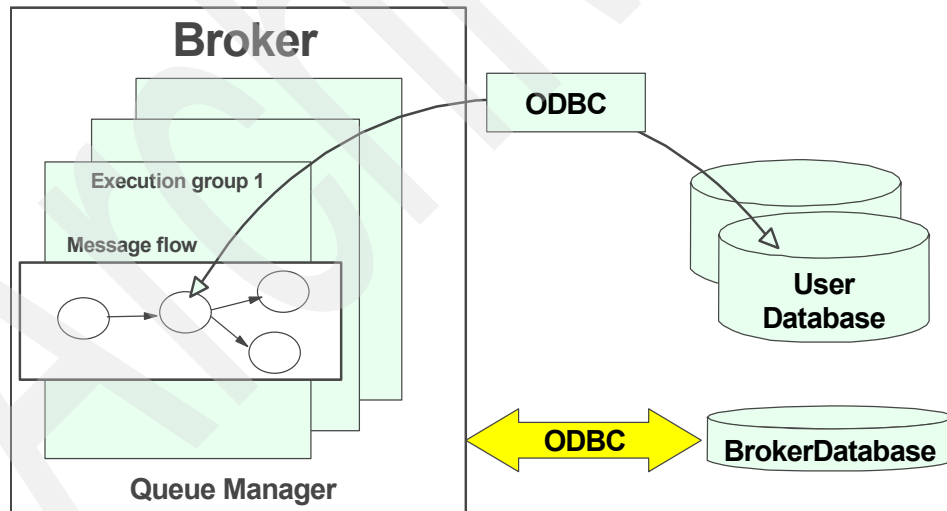
Figure 9-7   Structure of the broker

Message Flows

Message flows themselves are directed graphs. Each node in the graph represents some functionality that the broker will perform. The broker supports

many built-in nodes, such as Compute nodes, Database nodes, visual programming nodes (Check, Filter, Label, RouteToLabel nodes) and user-written nodes. A message flow usually starts with some type of an input node and ends with some kind of output node. For people who like to think pictorially, it is sometimes useful to think of message flows.

Flows often begin with an MQInput node, which means that the message flow starts by reading a message from a queue. This message is then passed along the directed graph to execute the logic that is modeled in the flow. Another commonly used type of input node is the HTTPInput node. This node is used to receive HTTP data streams. We use both MQInput nodes and HTTPInput nodes in our implementation.

Message flows in a broker can use databases as well perform look-ups and store data from flows. A number of database products are supported, depending on the actual platform. DB2 and Oracle are common database products that are used with a broker. On Windows, SQL Server is supported as well. We use DB2 for our database.

Message Model

Figure 9-8 on page 272 shows an overview of the different components of the message model. Each of these is described in more detail in the following text.
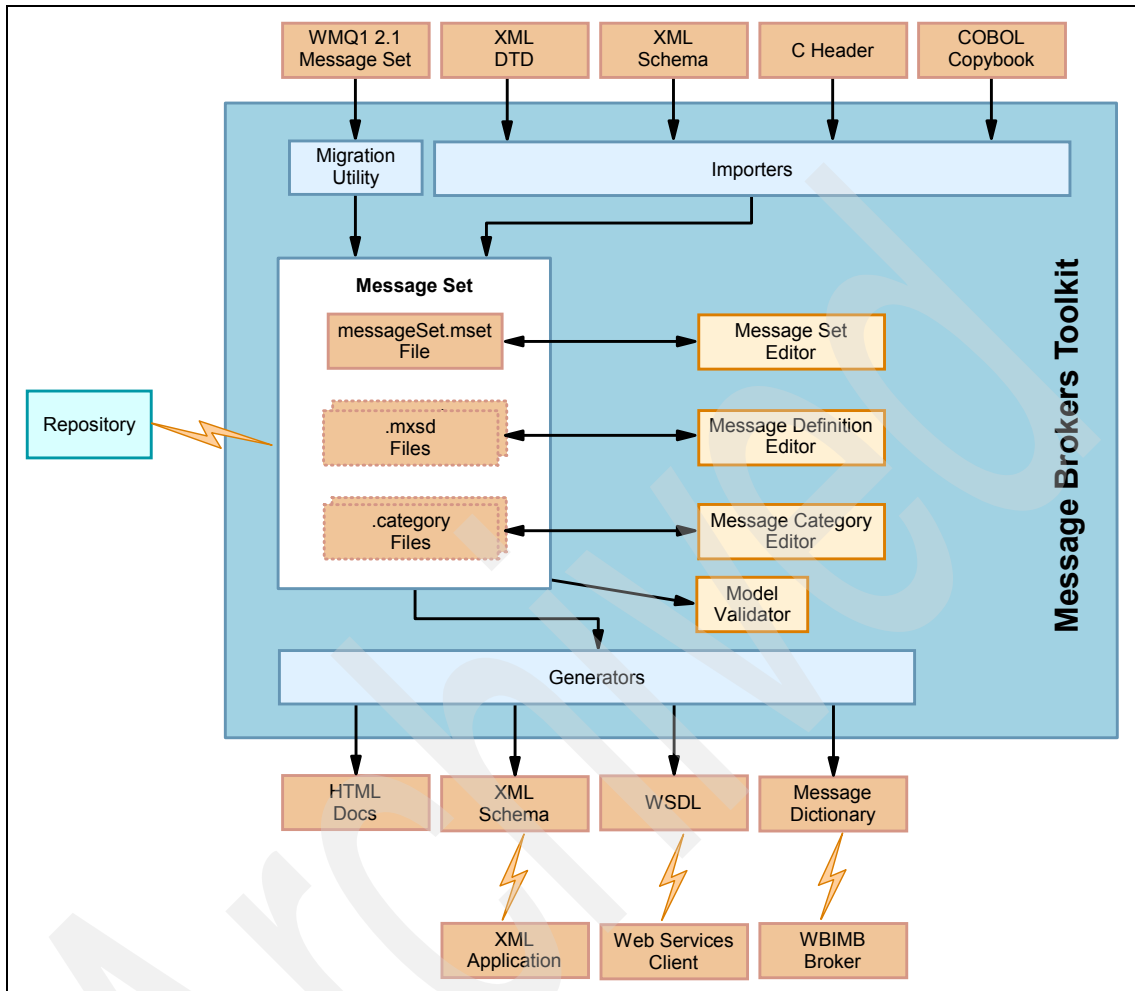
Figure 9-8   Overview of message model components

All the message resources are stored in files within the workspace repository - this is part of the integration with the Eclipse family.

Message sets

A *message set project* is a container for all the resources associated with *exactly one* message set. A *message se*t is a logical grouping of messages and the objects that comprise them (elements, types, groups). The contents of a message set are:

► Exactly one message set (messageSet.mset) file
► One or more message definition (.mxsd) files

► Zero or more message category (.category) files

The single message set file (.mset) provides information that is common across all the messages in the message set. This information is edited using the message set editor.

Each message definition (.mxsd) file contains definitions of messages, elements, types, and groups. Every message set requires at least one message definition file to describe its messages, although it is possible to distribute messages across multiple message definition files for better manageability. Message definition files internally use the XML Schema language to describe the logical format of messages. The physical format of the messages is stored using XML Schema annotations. The message definition editor can be used to create and edit the logical structure and physical formats of the messages.

Messages can be grouped into message categories, both for convenience and to aid in the generation of WSDL. The groupings are defined in .category files.

Message importers

The message definition files can be created and populated using one of the supplied importers. Importers are available for XML DTD, XML Schema, C structures, and COBOL structures. Importers can be used either from the mqsicreatemsgdefs command or from the Message Brokers Toolkit.

A command line migration utility mqsimigratemsgsets is also supplied to migrate WebSphere MQ Integrator V2.1 message sets into WebSphere Business Integration Message Broker V5 message sets.

Message editors

The message set files, message definition files, and message category files have their own editors that are used to create and maintain their resources. Although internally these files are XML, the supplied editor should always be used to edit these resources.

Message exporters

Generators are supplied to export message sets to standard external formats. The formats generated include:

► Message dictionary for deployment to a broker
► XML Schema to validate XML messages for a wire format layer
► Web services Description Language (WSDL) for Web services clients
► Documentation (as HTML)

The message model is validated each time a message set file, message definition file, or message category file is saved. The validation ensures model integrity of the logical structure and the physical formats.

9.3 Component Design

The component design for the proxyAssessorSystem should to take into account a number of constituents of the final implementation:

1. Message sets corresponding to solution interfaces
2. Message flows and transport independence
3. Distribution and Aggregation
4. Database Tables

This section describes the high-level design and associated design issues for implementing these four areas.

- ► 9.4, "Implementation of the message sets" on page 289
- ► 9.5, "Implementation of the database tables" on page 314
- ► 9.6, "Create the message flows" on page 319
- ► 9.7, "Create the ESQL code for the message flows" on page 351

9.3.1 Message Sets

Each of the ten interfaces listed in Figure 9-1 on page 265 has a request and possibly a response SOAP message. We need to be able to read and send messages with message formats and content corresponding to the ten interfaces.

Choice of parser

In WebSphere Business Integration Message Broker SOAP access to message content from the message flow is provided either by the XML parser or by the MRM parser. The XML parser is efficient, but does not have the capability to validate a message, and does not require a message definition to be provided. This can be regarded either as an advantage or disadvantage. It interprets the SOAP message at run time using the XML tags in the runtime message. The MRM parser, in contrast, is capable of validating messages against a message definition, requires a message definition, and during the creation of the message flow is able to provide content-assistance in the completion of the SQL statements querying and writing messages.

Best practice, if the message formats are known beforehand, is to use the MRM to develop and test the message flows. In production, there may be performance advantages in switching to the XML parser.

Creating message definitions for the SOAP interfaces

WebSphere Business Integration Message Broker will import schema files, but does not have a WSDL importer. In order to create message definitions for the WSDL files we need to extract schema definitions from the top-level message

elements in the WSDL file provided by the solution architect and import the definitions into a WebSphere Business Integration Message Broker message set. Each interface (request or request/reply are each counted as one interface) corresponds to a different definition within the same message set. Then, by importing the schema for the SOAP 1.1 WSDL definition and merging the SOAP schema with each interface schema, we create the message definitions for each interface.

All the necessary schema files have already been created from the corresponding WSDL files, and saved in the additional materials directory .\SG24-6636\Broker\Schemas. The procedure to convert a WSDL to a schema is described in the following sections.

The main problem we faced, which all projects hit at some time, is having multiple copies of the same definitions with the potential for definitions getting out of step. It is essential to have some procedure, either operational or technical, to keep the definitions in step. One approach is to define everything in schema files and use import statements in WSDL files. We took the approach of treating WSDL as the primary source and asking the System Architect to be responsible for all interface definitions.

The implementation of all the message definitions is described in 9.4, "Implementation of the message sets" on page 289.

9.3.2 Message flows and transport independence

The sequence diagram (Figure 9-3 on page 265) shows all the interactions with the proxyAssessorSystem. As an example, 3, 3a, 4 and 4a in Figure 9-9 show a magnified detail from the whole diagram.
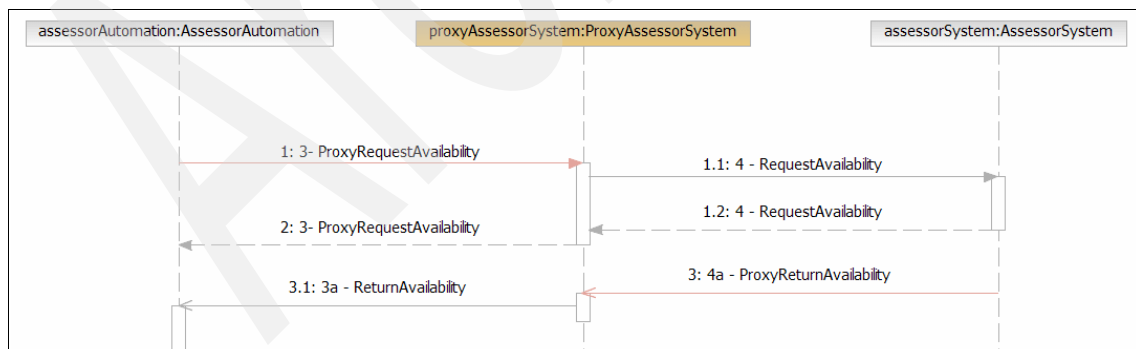
Figure 9-9   Flows 3,3a,4 and 4a

How should the interfaces shown in Figure 9-9 be mapped to message flows in the broker?

► There will be a minimum of one flow for each service offered by the broker as shown by the five bold entries in Table 9-1 on page 265. We call these Service Flows because they are implementations of Web services. Two of them are shown in Figure 9-9.

► There are five flows that call the output interfaces. These are called Client Flows because they are clients of Web services provided by other servers.

► To support interactions within LGI being migrated from SOAP/Http to SOAP/JMS the input and output nodes for LGI are split out and packaged in a special transport specific message flow project.

► There are Fault and Reply flows common across all the Service Flows in the solution which are implemented as transport specific flows. They return replies or faults to the clients of the services implemented by the broker.

► The Client flows need to catch any errors that occur in the flows or are returned by the service they call.

These errors are not returned to the Service Flow which called the Client Flow because the opportunity to use the Service Flow fault handler has gone. The Service Flow has already been returned an acknowledgement reply to its client, and the client is no longer waiting for a reply.

Fault handling by the Client flows is implemented by a TryCatch node attached to the sub flow input node. Errors are passed to a separate error handler we have called the ClientError flow.

► For this project, all the interactions with the assessors are limited to SOAP/Http and handled as part of the main flow.

One way to design the output flows to the assessors to support additional transport types is to use the "RouteToLabel" node to invoke a different output flow for each transport based on a label chosen in the flow. For input, from the assessors, we would reuse the concept of splitting the input flow from the input *interface* flow, and have a separate input flow for each transport type.

► In addition to these flow types, *incoming* acknowledgements to requests sent over an MQ or JMS messaging implementation have separate Ack flows because there is no combined Request/Reply node. The reply has to be fielded by a separate flow. We do not need these additional flows for SOAP/Http, because SOAP/Http has a synchronous HttpRequest node. Consequently, there are no Ack flows for the SOAP/Http and we have not written any Ack flows for this solution.

The various types of flow and their relationships are shown in Figure 9-10 on page 277.
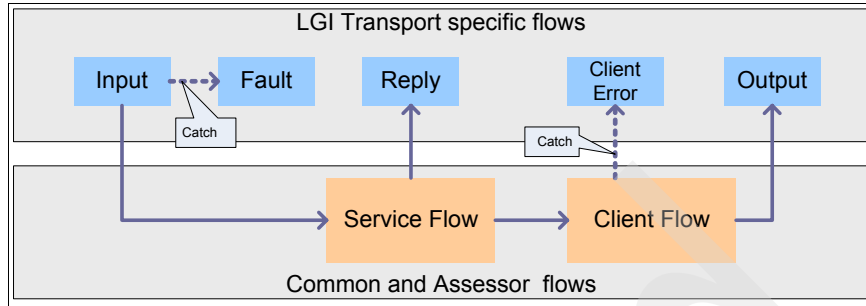
Figure 9-10   Separation of common and transport specific flows

The advantage of packaging the LGI transport specific (blue top rows) flows separately from the Common and Assessor (orange bottom) flows is that changing the LGI bus from SOAP/Http to SOAP/JMS should only necessitate changing the blue flows.

Using a little artistic license with the architect's original UML sequence diagrams, in Figure 9-11 we have shown the AssessorAvailability flows mapped to part of the sequence diagram from Rational Software Architect to try and show pictorially the correlation between the flows we have designed for the broker and the interactions specified in the solution architecture.
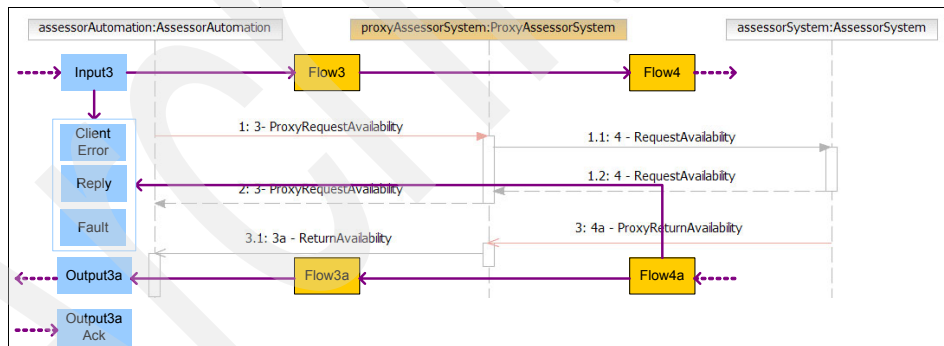
Figure 9-11   AssessorAvailability flows

Figure 9-12 on page 278 shows the equivalent Assessor Report flows.
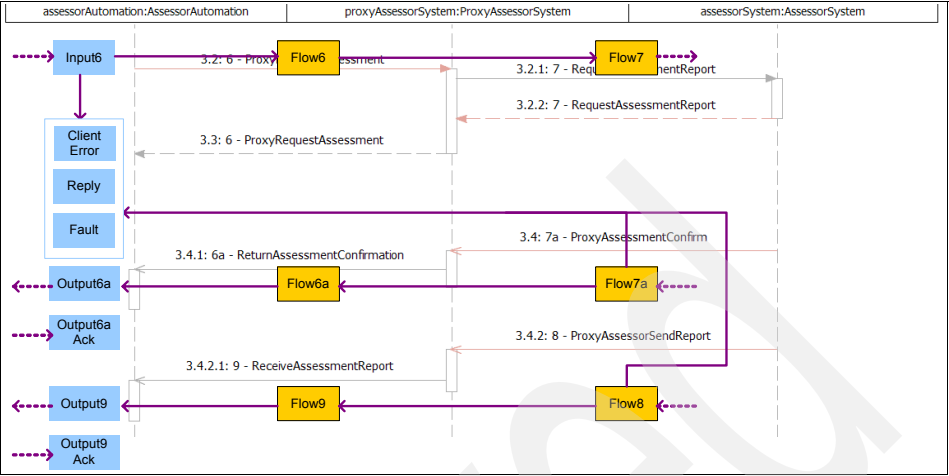
Figure 9-12   Assessor Report flows

Brief flow descriptions

Table 9-2 lists the common flows, Table 9-3 the transport independent flows some of which also interface to the assessors, and Table 9-4 the flows to interface to LGI.

Details of the flows are described in the implementation section 9.6, "Create the message flows" on page 319.

Table 9-2   Common flows

Flow	Invokes	Description
ClientError		Catches errors from Client flows
Fault		Prevents a poison message loop, creates and outputs a SOAP fault and outputs an error message
Reply		Sends a reply message in the correct format

Table 9-3   Transport independent flows and flows to and from assessors

Flow	Invokes	Description
Assessor Availability Flows		
Flow3	Flow4	Receives request for list of available assessors
Flow3a	Output3a	Aggregates and sends list of available assessors back
Flow4	EA[a]	Distributes availability request to assessors

Flow	Invokes	Description
Flow4a	Flow3a	Receives availability responses from assessors
Assessor Report Flows		
Flow6	Flow7	Receive request for assessment report
Flow6a	Output6a	Return acceptance from assessor
Flow7	EA	Request assessment from assessor
Flow7a	Flow6a	Receive acceptance from assessor
Flow8	Flow9	Receive assessment report from assessor
Flow9	Output9	Send assessment report from assessor

a. EA - External Assessor

Table 9-4   Transport specific flows to and from LGI

Flow	Invokes	Description
Assessor Availability Flows		
Input3	Flow3	Receives request for list of available assessors
Output3a	AAS[a]	Aggregates and sends list of available assessors back
Flow3aAck		Handles acknowledgement to returned assessor list
Assessor Report Flows		
Input6	Flow6	Receive request for assessment report
Output6a	AAS	Return acceptance from assessor
Flow6aAck		Handles acknowledgement to returned acceptance
Output9	EA	Send assessment report from assessor
Flow9Ack		Handles acknowledgement to returned report

a. AAS - Assessor Automation System

9.3.3 Database tables

The proxyAssessorAutomation system requires three database tables to manage the correlation of data flowing between services.

CLAIMASSESSOR

CLAIMASSESSOR is used to store details of availability requests sent to assessors (flow4), and is updated with details of the assessor's responses to these.

Table 9-5   CLAIMASSESSOR table

Column name & key	Type	Column Updated by
claimID (index ca1)	Integer	flow4
assessorID (index ca1)	Integer	flow4
assessorURL	Long Varchar	flow4
location	Char(100)	flow4
reqdate	Date	flow4
makeOfCar	Char(15)	flow4
registration	Char(7)	flow4
predDate	Date	flow3a
predCost	Integer	flow3a
replytoq	Char(48)	flow4
replytoqmgr	Char(48)	flow4
correlid	Blob(24)	flow4
requestcompletetime	Timestamp	flow3a

ACTIONASSESSOR

ACTIONASSSESSOR is used to store details of confirmation requests sent to assessors (flow7), and is updated with details of the assessor's responses to these

Table 9-6   ACTIONASSESSOR table

Column name & key	Type	Column Updated by
claimID (index aa1)	Integer	flow7
assessorID (index aa1)	Integer	flow7
assessorURL ,	Long Varchar	flow7
location	Char(100)	flow7
reqDate	Date	flow7

Column name & key	Type	Column Updated by
makeOfCar	Char(15),	flow7
registration	Char(7)	flow7
ackfromassessor,	Char(10)	flow7ack
confirmedDate	Date	flow6a
accepted	Char(5)	flow6a
replytoq	Char(48)	flow7
replytoqmgr	Char(48)	flow7
requestcompletetime	Timestamp	flow6a
rejected	Char(20)	flow8
rejectedDate	Date	flow8

RESOLVEASSESSOR

RESOLVEASSESSOR is used when there are a number of assessor types, each requiring messages in a different format, we need a way of identifying to which type of assessor a particular message is to be sent.   This table gives the type of assessor for each assessor ID, as well as technical details such as HTTP address. In this implementation, all the assessors are accessed using SOAP/Http and we do not use this table.

Table 9-7   RESOLVEASSESSOR table

Column name and key	Type
assessorID (index ra1)	Integer
assessorType	Char(10)
assessorhttpaddress	Long Varchar
lgibrokerhttpaddress	Long Varchar

9.3.4 Distribution and aggregation

An important capability provided by the broker is to distribute and aggregate messages. The proxyAssessorSystem uses this capability to send requests to tender for doing a claim assessment to eligible external claim assessors and then to aggregate the responses. The ExternalClaimAssessor process sends a list of all the eligible assessors to the proxyAssessorSystem. The proxyAssessorSystem splits up the list into individual requests, which in a full

implementation will be sent over a variety of transports, and sends them to the assessors using SOAP/Http, receiving an acknowledgement from each assessor that it has received the request. At some later time, which is set contractually between LGI and the Assessors, and which can vary depending on the class of insurance the customer is carrying, each assessor can send in a tender. The tenders are collected together by the proxyAssessorSystem and sent to the ExternalClaimAssessor process as a single list.

Aggregation nodes

The broker provides three nodes to implement aggregation:

1. Aggregate Control node
2. Aggregate Request node
3. Aggregate Reply node

The Aggregate Control and Request nodes are used in the distribution flow, and the Aggregate Reply node in the aggregation flow to assemble the replies. The aggregation nodes provide the framework for implementing distribution and aggregation. The broker IT specialist must provide the message flow to connect the nodes, and handle the construction of the fan-out messages and the final composed message. Before considering the design for the proxyAssessorSystem, it is worth considering a number of design issues.

One flow or two?

The distribution and aggregation flows can be one and the same physical flow. The lifetime of the whole distribution/aggregation process is the lifetime of the flow. Alternatively, it can be implemented as two flows, with a control message passed from the distribution flow to the aggregation flow to coordinate the two halves of the process. A single flow is less complex to implement, but two flows provide more flexibility to implement transactional behavior, and provides more manageability options at runtime, such as running in different execution groups.

Transport Protocols

Out of the box aggregation supports *any* transport protocol for receiving and returning the list of assessors to the ExternalClaimAssessors process. The intermediate flows to and from the assessors have to run over one of the built-in transports that support a request/reply message protocol:

► WebSphere MQ Enterprise Transport (MQInput and MQOutput nodes)
► WebSphere MQ Mobile Transport (MQeInput and MQeOutput nodes)

The broker does not support the built-in protocols that do not have a request/reply message model, or user defined transport protocols:

► WebSphere MQ Web services Transport (HTTPInput, HTTPReply, and HTTPRequest nodes)

- ► WebSphere MQ Real-time Transport (Real-timeInput and Real-timeOptimizedFlow nodes)
- ► WebSphere MQ Telemetry Transport (SCADAInput and SCADAOutput nodes)

We need our aggregation service to support any transport type because we allow assessors to connect to LGI using a variety of protocols. The solution is to convert the availability flows to use WebSphere MQ within the broker and converting the requests to and from the assessors between WebSphere MQ and HTTP/SOAP. The Version 5 infocenter documents the aggregation node interfaces to enable non-WebSphere MQ aggregation flows to be built. The implementation of the interface is now available in WebSphere Business Integration Message Broker Version 6.0, including supporting aggregation over SOAP/Http transport, removing the need to use intermediate WebSphere MQ flows.

Robustness and timeouts

An important part of designing aggregation is the consideration of time. By its nature, we are dealing with a drawn out process and need to consider how long to wait for replies, what to do with late replies, and because we might be talking in terms of hours or days to assemble replies, the consequence of an interrupted aggregation process.

Timeouts

Timeouts should be considered as part of the requirements for the process, and set by the Business Analyst and Architect. In our case, the timeout values are incorporated into the contracts with customers and assessors. There are two timeouts to be set. The green bubbles in Figure 9-13 on page 285 show the nodes on which the timeouts are set.

The first timeout to set is on the Aggregate Control node. It controls the overall duration of the aggregation process; how long the process will wait for replies from the assessors. The value contracted by LGI with the assessors needs to satisfy the response time expectation of the customer to whom the insurance policy has been sold. Once the timeout is exceeded the aggregation reply node routes replies to a different output terminal for processing as late replies.

In the case of the claims process, there are two classes of customer with different projected response times and therefore there needs to be two different timeout settings. Because the timeout value is hard coded as an Aggregate Control node property then for two different contracts with assessors, two Aggregation Control nodes advertising two different timeout values are required. In the implementation, for simplicity, only one response time policy is supported.

A second timeout value, this one set on the Aggregate reply node, is difficult to choose, but not too critical. It specifies how long the Aggregation reply node will wait upon receiving a message before it decides it cannot associate it with an control message sent from the aggregation control node. After the timeout expires, the aggregate reply node discards the message on its Unknown terminal. This is a global property of the aggregate reply node.

As an illustration, think about this situation: you plan to catch a train to London from an unmanned train station. You know the trains to London are very reliable and plan to catch the 10:00 AM train. If you actually arrive a little late at 10:01AM, and the train is not at the station, have you missed it, or is the train running late? How much longer should you hang around the station before deciding you have missed the train? In this example, the train arriving is the control message, the time it spends at the platform the overall duration of the aggregation, and the passengers arriving to catch the train, the replies to be aggregated. The second timeout value is the length of time you are prepared to wait before deciding you have missed the train, or the train is not coming.

Your choice of value for the second timeout will determine when reply messages start appearing on the Unknown terminal and would signify that something has gone wrong. In our scenario, setting the second timeout value to the same value as the overall duration is a reasonable choice.

Interruptions

The Aggregation Reply node holds its state in a the broker's database, so short lived interruptions will not cause undue problems. When the flow is resumed, the Aggregation reply node will continue until the timeout expires. For more long-lived interruptions, if the flow restarts after the timeout would have expired, then messages arriving after the flow stopped executing will be treated as late replies.

Design of Distribution and Aggregation flows

There is good documentation of the technical details of aggregation and how to construct a solution in the broker infocenter. Figure 9-13 on page 285 pulls all these pieces together into a single diagram which we will use to tie together the individual pieces of the implementation that are distributed over multiple flows, esql files, and message nodes. To describe the design, we step through the flows, node by node using Figure 9-13 on page 285.
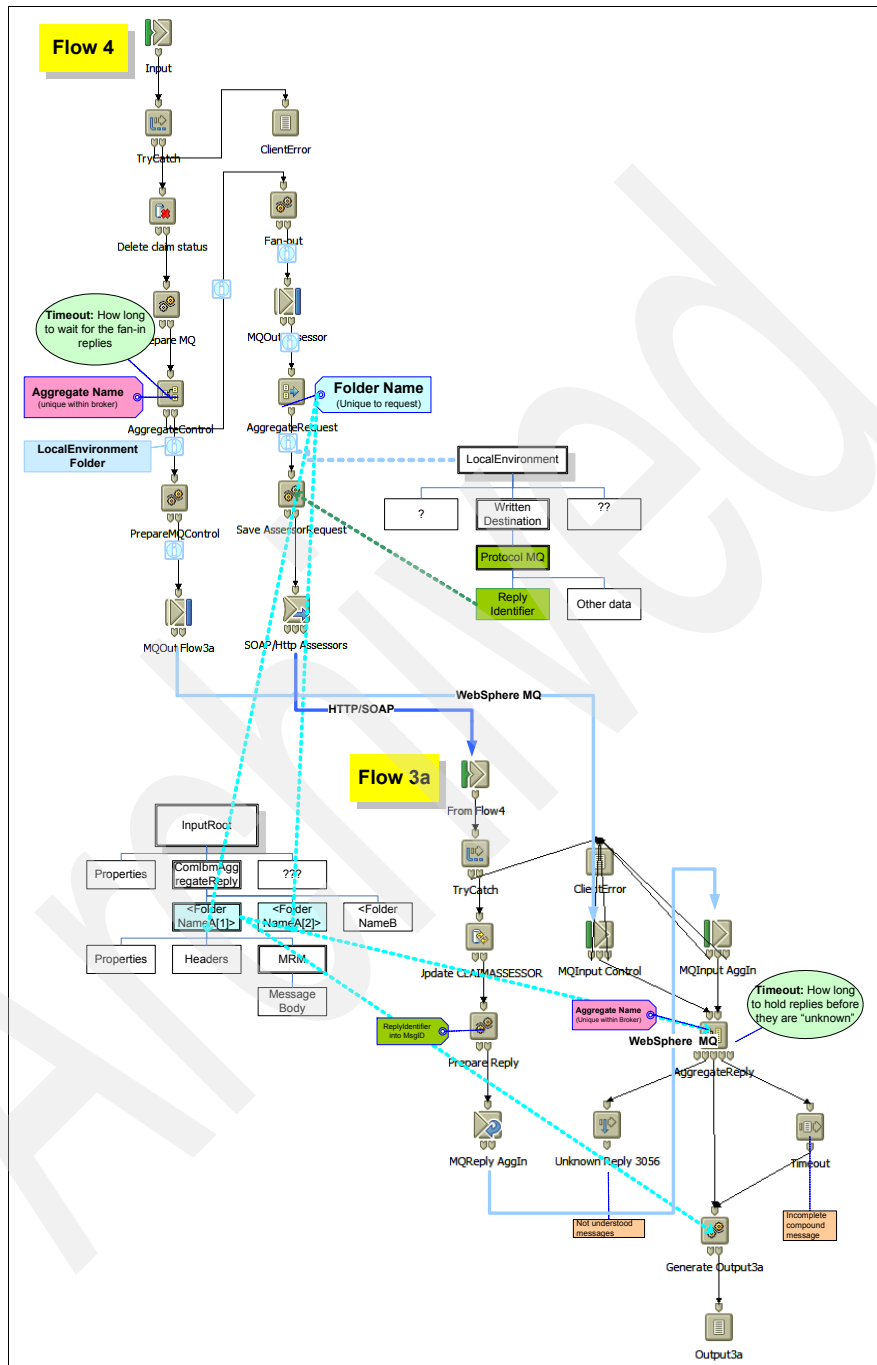
Figure 9-13   Complete view of message distribution and aggregation

Flow 4

The availability request from the ExternalClaimAssessor process has been received and validated. Flow 4 is invoked from Flow 3.

Preparation

1. The HTTP/SOAP message is received from the input server flow after validation.

2. The TryCatch node sets up the error context for the rest of the processing, acting as a SOAP client for requesting the assessor availability from each of the assessors.

3. The database delete node, Delete Claim Status, is used to clean out any previous claim status before the new request is processed.

4. The Prepare MQ node removes all vestiges of the HTTP request from the broker folders[1], and sets up the MQ folder to create a new MQ message.

Aggregate Control

5. The Aggregate Control [AC] node starts the distribution/aggregation process.

    a. As described earlier, the timeout for the replies is specified on the AC node.

    b. A name is also given to the aggregation to distinguish it from other aggregations that may be happening. For example, if we had implemented different aggregations for different response times.

    c. The AC node puts information into the LocalEnvironment folder which must be flowed to the Aggregate request node and to the MQ Output node which will sent the aggregation control message to the Aggregate reply node.

    d. The PrepareMQControl node creates the control message which is sent over MQ to Flow4.(MQMD). Example 9-1 shows how to construct the new MQMD.

Example 9-1   Creating empty MQMD

```
CREATE NEXTSIBLING OF OutputRoot.Properties DOMAIN 'MQMD';
SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID;
SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION
```

Fan Out

6. The Fan Out node propagates individual request messages, in series, along the rest of the flow by stepping through the list of eligible assessors in the

---

[1] We found that some broker functions could get confused if there were both MQ and HTTP folders in the broker. So it is important when switching transports to clear out old, unnecessary folders.

request message. Each request message is to a different assessor at a different destination. Example 9-2 shows how to set the assessorURL.

Example 9-2   Setting SOAP/Http destination

```
OutputLocalEnvironment.Destination.HTTP.RequestURL =
InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:assessorL
ist.fl3:assessors[assessorCount].fl3:assessorURL;
```

The Fan-Out compute node must have its compute node properties set to propagate the LocalEnvironment as well as the normal setting to propagate the message tree. There must also be a line of code (see Example 9-3) to copy the LocalEnvironment which has been received from the Aggregate Control node.

Example 9-3   Copying local environment

```
-- Following statement is needed for the aggregate node
SET OutputLocalEnvironment = InputLocalEnvironment;
```

MQOut Assessor

7. The MQOut Assessor node actually sends an WebSphere MQ Message to the Assessor queue. The message is never sent anywhere, but it creates a Written Destination folder in the LocalEnvironment which is used by the Aggregate Request node to set up correlation information for the aggregation.

We extracted the generated MsgID from the Written Destination folder to make sure it is exactly the one saved by the Aggregation Request node. We had some problems storing our own MsgID in the MQMD and setting the MQOut node property flag not to generate a new MsgID. It didn't seem to match the one in the Aggregation Request node table. To be sure that the hidden MsgId stored by the Aggregation Request node is the same as the one we store in the CLAIMSASSESSOR table, we saved the MsgID from the Written Destination. That way we can be quite sure we are saving the same MsgID as the Aggregation Request node. We never convinced ourselves why we were having these failures to match on the MsgID, but designing the flow to use the Written Destination eliminated the possibility of a problem.

AggregateRequest

8. For the Aggregate Request node, all we provide is the name of the folder which will be created to contain the composite reply message (Example 9-4)

Example 9-4   Composite reply folder

```
InputRoot.comIbmAggregateReply.<folder name>
```

All our request messages will use the same folder name, so individual reply messages are indexed off this folder name in the aggregate response message.

Save AssessorRequest

9. The Save AssessorRequest node saves the MsgId (= ReplyIdentifier) in the WrittenDestination folder to the ClaimsAssessor database as the CorrelId field, keying it off the ClaimID and AssessorID. The value is restored into the MQMD.CorrelId field for each reply as it is received back in flow 3a. See Example 9-5.

Example 9-5 Storing correlation information into CLAIMASSESSOR

```
INSERT INTO Database.EMERGE.CLAIMASSESSOR (
    claimID, assessorID, assessorURL,location, reqdate, makeofcar,
    registration,replytoq, replytoqmgr, correlid) VALUES (
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:claimID,
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:assessorID,
    InputLocalEnvironment.Destination.HTTP.RequestURL,
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:location,
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:reqDate,
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.
        fl7:makeOfCar,
    InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.
    fl7:registration,
        'NoReplytoQ',
    'NoReplytoQmgr',
    InputLocalEnvironment.WrittenDestination.MQ.DestinationData.msgId);
```

SOAP/HTTP Assessor

10. Finally each request message is sent off to an assessor. If we were supporting multiple connection types, we would use a RouteToLabel node to select the output node.

Flow 3a

Flow 3a receives control from Flow4a which implements the return availability service for the assessors.

Preparation

1. The TryCatch node sets up the exception handling context for the rest of the process which is a client to the ExternalClaimsAssessor process.

2. Update CLAIMASSESSOR updates the database with the information received from each assessor, for no other reason than auditing. The saved information is not used in the rest of the scenario.

3. The Prepare Reply node clears out any HTTP/SOAP information from the broker folders, and effectively creates a Request message, with all the response information from an assessor before passing it onto the MQReply node. This is to make it seem as though an MQ Request message had been processed all along, rather than the request being sent over HTTP.

4. MQReply AggIn puts the response onto the AggIn reply queue, following the MQMD option to copy the MsgID (which is now the original ReplyIdentifier) in to the CorrelId.

   It would probably work to omit creating real a WebSphere MQ reply message, and simply pass the correctly constructed folders to AggregateReply. We were having some problems with getting aggregation to work properly, so we did not attempt this optimization.

MQInput AggIn

5. MQInput AggIn receives the reply message on the AggIn queue and passes it to the AggregateReply node.

MQInput Control

6. MQInput Control receives the Control message on the FLOW3A.CONTROLQ and passes it to the AggregateReply node.

Aggregate Reply

7. The Aggregate Reply node builds the composite response message in the folder specified by the Aggregate Request node. As previously discussed, there are two timers running. The unknown timer directs late replies to the unknown terminal. The timeout timer sends an incomplete composite message to the Timeout terminal. In the diagram, both are shown as going to trace nodes. In practice we shall build the request message to the ExternalClaimAssessors process from either the Out terminal or the Timeout terminal.

Generate Output3a

8. Generate Output3a constructs the request message using the aggregated response folder constructed by AggregateReply.

9.4 Implementation of the message sets

The six steps we need to take to create the message sets containing the message definitions for all the interfaces listed in Table 9-1 on page 265 are:

1. Convert the WSDL message definitions into schemas.
2. Create the message set project.

3. Create the message set.
4. Import the interface schema.
5. Convert schemas into message definitions
6. Customize the message definitions to create SOAP envelopes

9.4.1 Convert the messages in wsdl files into schemas

WebSphere Studio Application Development Integration Edition or Rational Software Architect are the best tools to use to convert WSDLs into schemas because both have a specialized WSDL editor and schema editor. WebSphere Business Integration Message Broker only has a schema editor.

1. In WebSphere Studio Application Development Integration Edition create new schema files for each of the WSDL files that are to be converted.

   Create the schema files with the same file names but with the extension .xsd. There is a wizard to help you do this. Click **File** → **New** → **XML** → **XML** Schema name the schema → **Finish.** See Figure 9-14.
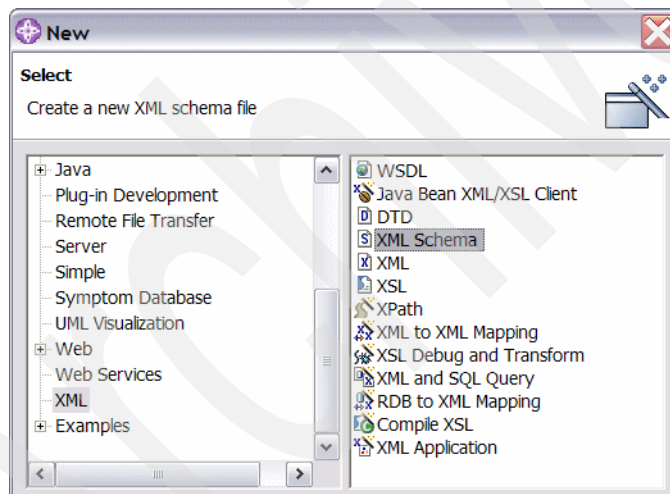
Figure 9-14   Creating a new schema file in Integration Edition

   Save the files in the wsdl directory. Figure 9-15 on page 291 shows the project tree in WebSphere Studio Application Development Integration Edition with some of the WSDL and .xsd files.
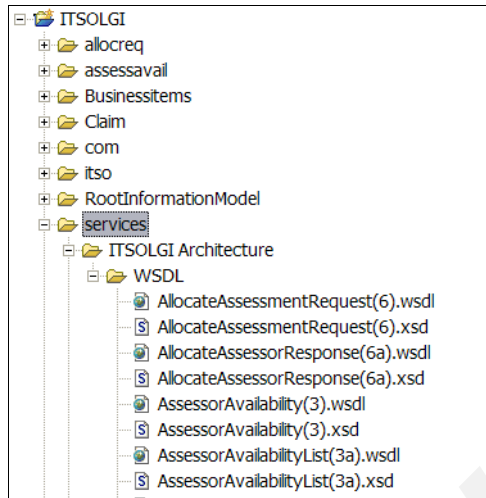
Figure 9-15   Some of the schema and WSDL files in the project

2. There are two types of WSDL files. Some have the full message definitions embedded in them, and a few refer to the Businessitems.xsd schema. Starting with the files with the full message definitions is easiest. When those are converted, it will be clear how to edit the remaining files.

   a. Taking AssessorAvailability(3).wsdl as an example, copy everything between `<wsdl:types>` and `<\wsdl:types>`. Open the AssessorAvailability(3).xsd file and select from `<schema ...` to `<\schema>` (Example 9-6) and replace with the copied definitions.

Example 9-6   Selected schema statement

```
<schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://www.
                ibm.com" xmlns:test="http://www.ibm.com">

</schema>
```

   b. The file should save without errors. As an optional task, tidy up the schema statement and the rest of the file: The first part of the schema file is shown in Example 9-7.

Example 9-7   Tidied schema file

```
<?xml version="1.0" encoding="UTF-8"?>
<schema elementFormDefault="qualified"
targetNamespace="http://broker.lgi.itso.assessavail"
xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:intf="http://broker.lgi.itso.assessavail" >
    <element name="requestAssessorAvailability">
        <complexType>
```

```
        <sequence>
            <element name="claimID" nillable="true" type="int" />
            <element name="location" nillable="true" type="string" />
            <element name="responseTime" nillable="true" type="string" />
            <element name="requiredDate" nillable="true" type="string" />
            <element name="makeOfCar" nillable="true" type="string" />
            <element name="assessorList" nillable="true"
type="intf:AssessorList" />
        </sequence>
    </complexType>
</element>
... continued ...
```

  i.   Duplicate namespaces have been removed.
  ii. Because `http://www.w3.org/2001/XMLSchema` is the default namespace all
       the `xsd:` prefixes can be removed.
  iii. Make sure the target namespace is unique across the message set,
       and it matches the namespace associated with the prefix used for any
       embedded complex types.

Attention: We suggest that each schema file that you create has its own
target namespace prefix when the schema is imported into the broker MRM.
The prefix naming scheme we have adopted is flxx where xx is the flow
number. Changing the prefix has no impact on the semantics of the schema,
but does assist with readability and avoiding confusion.

The best way to avoid naming problems is to manage namespaces and type
definitions across the whole integration space, but this is just not possible in
many cases for organizational and historical reasons. In this project we have
some clashes, reflecting the real world as we find it, and we tackle the issue of
how to resolve naming clashes.

From a software engineering perspective. an important goal is to have only
one named type for each type of data, in other words to avoid duplicate
definitions that are a source of error if the definitions are different. If this goal
cannot be achieved readily, then a weaker but more easily managed goal is
for every duplicate type to be in a different namespace. Ensure different
developers or development teams have a root namespace stem in which to
define unique namespace names and manage their types.

---

3. Export the schema files as a .zip file or as a file system ready to import into
   the broker.

9.4.2 Create the Message Set Project

Each message set project can contain but one message set. Each message set holds multiple message definitions. Each message definition corresponds to a single schema file. The same schema can be used in different message definitions - and lots of elements can be collected into a single schema file.
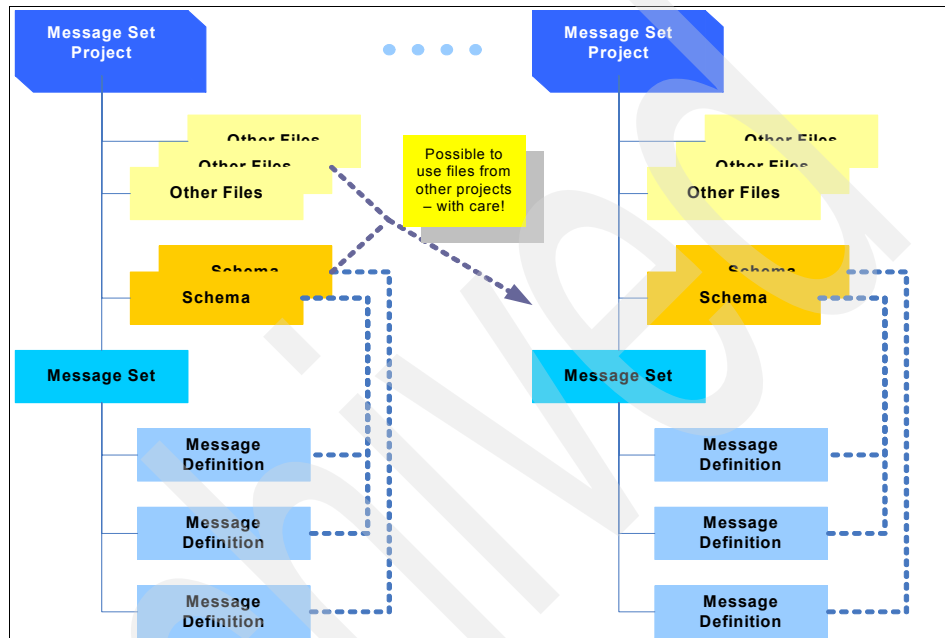
Figure 9-16   Organizing Broker message sets

How many message set projects do we want to create to hold each of the message definitions we need? We could put all the definitions into a single set or put each definition into its own message set.

Generally it is best to have all the message definitions in the same message set. It is easier to manage and with lots of definitions, much quicker to create. From a software engineering perspective, it enables complex types to be reused and so reduces the chances of error. However along with this benefit comes the need for discipline in the creation of types and messages. Duplication of types and messages in the same message set will cause errors, regardless of the types and messages being in different namespaces.

From a project management perspective this discipline can be difficult to accomplish, especially with application integration projects where you might not have control over the naming of parts. So the broker provides us with multiple message sets and projects and no sharing of parts between the different sets.

Parts from different messages sets can all be used within a single flow, but the responsibility is left with the message flow implementers to ensure that the message correct parts are used together correctly.

In the case of the ExternalClaimAssessors project, the pieces have not been designed as a whole, and there are duplicate complex types and message names. Also, the practice we have adopted of making each WSDL file *self-contained* means we have duplicates of the same types to import into the broker. So we cannot simply create all the message definitions in a single message set. We have to decide what to do. We could go back and change the definitions, rationalizing names and generalizing types, or we could create a number of different message sets.

In general it is not easy to go back and change message definitions in WSDL files. There are numerous dependencies to track down, including having to completely recreate partner links, fix-up transformer WSDL definitions and if programs such as EJBs are involved, regenerate the EJB from the new WSDL and copy user code from the old EJB into the new EJB. Then there is the likelihood of introducing errors when doing these tasks to consider.

The best practice for enterprise application integration is usually to minimize rework of existing code, and to use the capabilities of integration servers, such as WebSphere Business Integration Message Broker, to glue the pieces together.

We initially set out by defining multiple message sets which avoided type name and message name clashes when we imported the schemas into the broker. Table 9-8 shows how we split the message definitions up.

Table 9-8   Message sets and definitions

.xsd	Message set project	Message set
AssessorAvailability(3) AssessorAvailability(3a)	AssessorAvailability 3 and 3a	3 and 3a
AssessorAvailability(4) AssessorAvailability(4a)	AssessorAvailability 4 and 4a	4 and 4a
AllocateAssessment(6) AllocateAssessment(6a)	AllocateAssessment 6 and 6a	6 and 6a
AssessorReport(7) AssessorReport(7a)	AssessorReport 7 and 7a	7 and 7a
AssessorReport(8) AssessorReport(9)	AssessorReport 8 and 9	8 and 9

This worked fine until we started writing ESQL and we discovered that although the ESQL compiler was happy with using multiple message sets, autocomplete had a particular problem, largely due to all our types eventually being referenced from a single SOAP envelope message. Autocomplete would only suggest specialized Body elements from one of the message sets. As a matter of taste, we could have continued without autocomplete working effectively, and used multiple message sets to resolve message name and type clashes. Our judgement was that it was necessary to take the hit, resolve the name clashes and use a single message set. We expect version 6 of WebSphere Business Integration Message Broker to contain enhancements to improve its capabilities in this area making the multiple message set approach preferable.

Our goal now is to resolve the name clashes that result from using a single message set in WebSphere Business Integration Message Broker without altering the WSDL defining the Web services.

> **Tip:** By default, the broker saves its workspace in the install path. To work with multiple workspaces and store them in your My Documents folder so they get archived by backup schedules that have excluded ...\Program Files\... , modify the shortcut to the workbench by adding a -data parameter:
>
> ```
> "C:\Program Files\IBM\WebSphere Business Integration Message
> Brokers\eclipse\mqsistudio.exe" -data "C:\Documents and
> Settings\Administrator\My Documents\ITSO\SA-H414\In work\Code\Final
> tested files\wbimb\workspace"
> ```
>
> Beware: this creates a long file path. You will see later, this causes the workbench to start failing in hard to predict ways. Later in this book, there are two more tips which show how you can continue to store workspaces in your My Documents folder, but not have the problem of the long path name.

Create a single message set project called Assessor Messageset.

1. Open the broker toolkit and select the **Broker Application Development Perspective**.
2. Select **File** → **New** → **Message Set Project**. Type `Assessor Messageset` → **Next** → **Finish**.
3. You can create the message set at the same time. We will use another wizard. See Figure 9-17 on page 296.
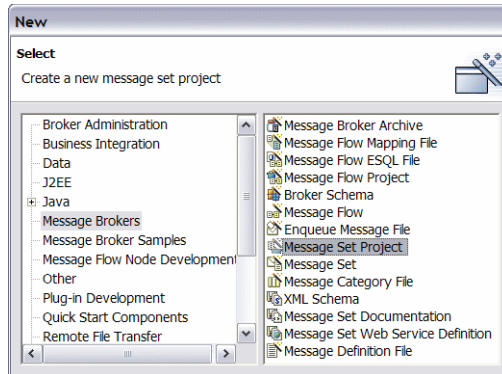
Figure 9-17   Create a new message set project

9.4.3 Create the Message Set

After creating the message set project, create the message set which will contain all the message definitions

1. Right-click each message project → **New** → Message Set and enter the message set name `proxyAssessorMessages`.

    a. Check the box entitled **Use** namespaces → **Next.** See Figure 9-18.
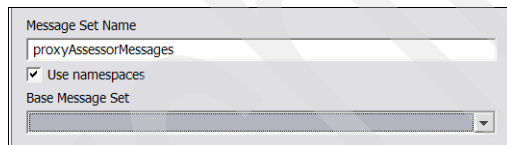
Figure 9-18   Check the use namespaces box

    b. Check the box beside **XML Wire Format Name** → **Finish**. See Figure 9-19.
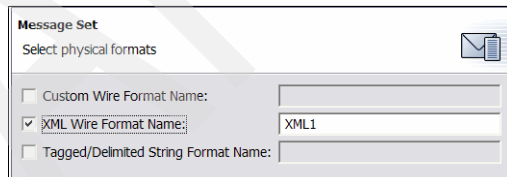
Figure 9-19   Check the XML wire format box

You can change the name of the XML wire format at this point, but we leave it as XML1. The name is associated with the wire format of the XML data stream that is used in common across all the message definitions in the message set. The wire format name must be used consistently

throughout and match any (if specified) name in the input message. It is easy to overlook that there is an unmatched message format name that is the cause a message flow not to work. The XML wire format can be customized as we shall see next. The customized format must be the same across all the message definitions in the message set.
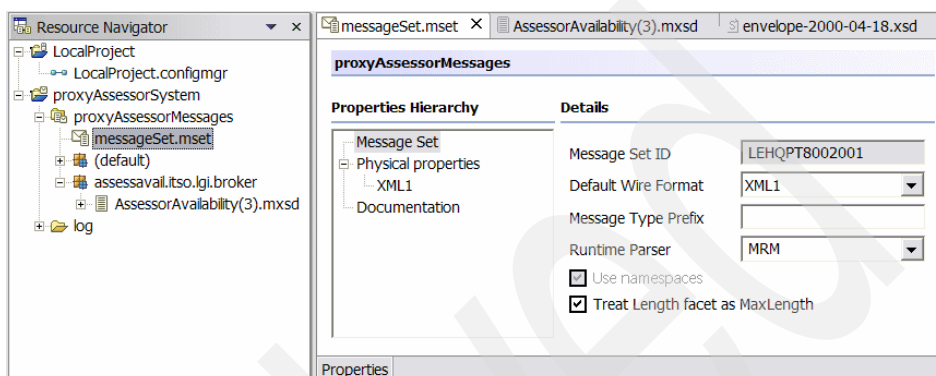
Figure 9-20   Message set properties

The new message set is created and the messageSet.mset file is displayed.

2. Set the default wire format to XML1 (the default is taken if the wire format is not named either in the input message, or in the input node of the message flow).
3. Under the Properties Hierarchy of messageSet.mset expand Physical properties → **XML1**.
4. Under **Details of messageSet.mset**
   a. Check **Suppress doctype**.

   DTD declarations are not needed; the system is schema rather than DTD-based.
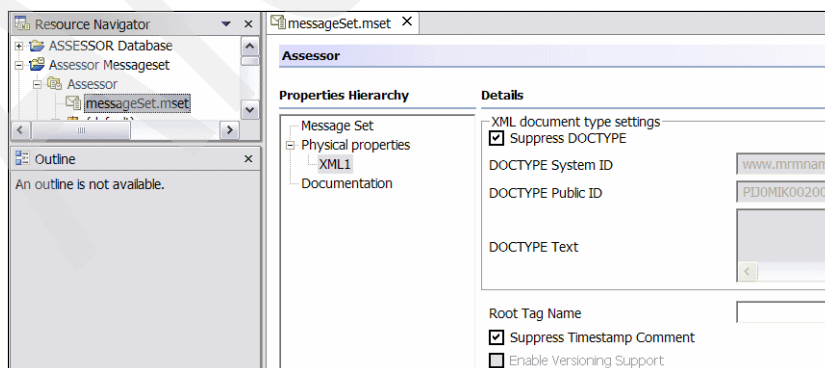
Figure 9-21   Customize the XML generation

b. Clear the **Root Tag Name**.

We use SOAP messages, where the root tag name must be 'Envelope'. Also the system has the potential to use embedded messages. It is not sensible to have a hard-coded root tag name when embedded messages are used.

9.4.4 Import the schemas into the broker

Next, import the schema files into the Broker Toolkit workspace. You can organize the schema import by placing the schema files relevant to each message set project in its respective project, or you can import all the schemas into a common folder.

1. In Broker Application Development Perspective, change to the Resource Navigator window → right-click **proxyAssessorSystem** → **File** → **Import** → **File system** → **Next**.

   a. **Browse** to the directory where the schema files are located and check the files that you want to import.
   b. The destination for the imported resources should already be listed as the **proxyAssessorSystem** → Check **Create selected folders only** → **Finish.**

2. Verify the schemas by opening and studying the graphical display and comparing with the original WSDL files. See Figure 9-22 on page 299.
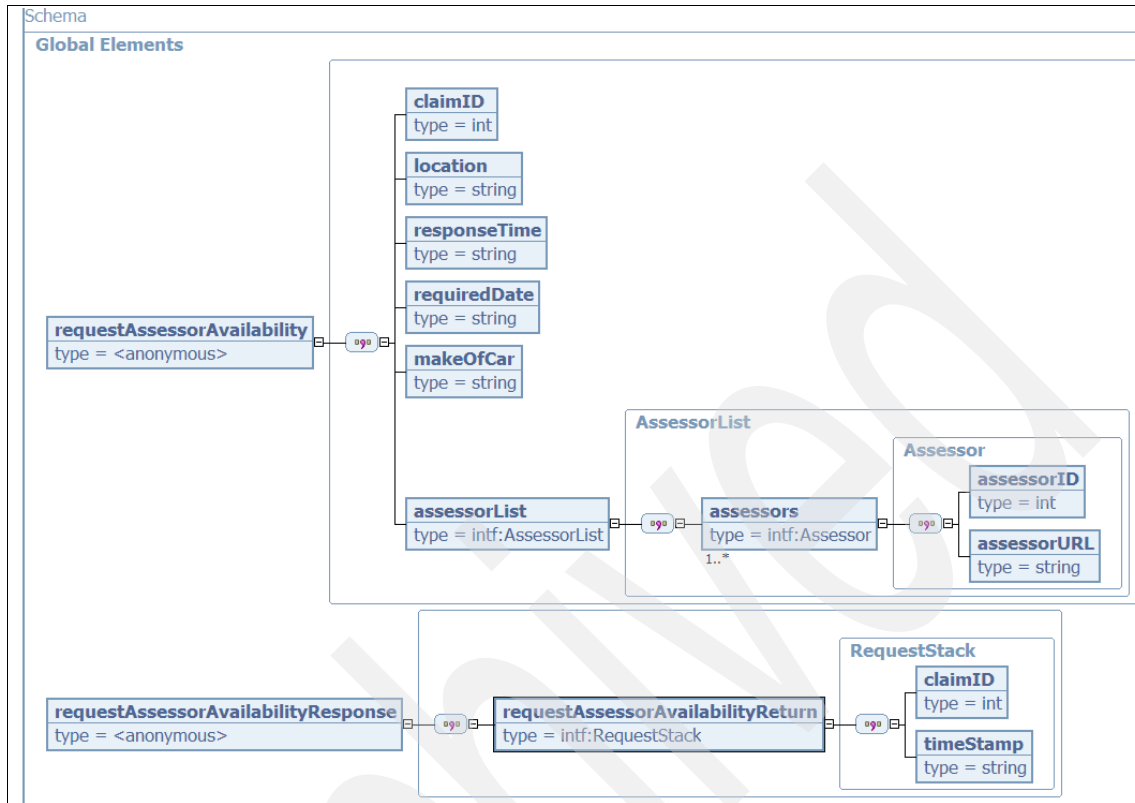
Figure 9-22 RequestAssessorAvailablity schema

3. Use the outline view and select the top level schema, or select the top level on the graphical tab. Then, open the design tab in the property editor and check that the target namespace is what you want.

4. The last thing to import is a schema file that describes SOAP messages. With your browser, visit the World Wide Web Consortium (W3C) Web site:

   `http://schemas.xmlsoap.org/soap/envelope/`

   Save the schema for SOAP 1.1.

9.4.5 Using schemas to create MDFs

Now that all of our schema files are in the Broker Toolkit workspace, we can use them to populate the message definition files [MDFs].

1. In Broker Application Development Perspective, in the Resource Navigator window, right-click the schema file that you want to use as a message definition source. Select **New** → **Message Definition File**.

2. The **XML** schema file radio button should be selected. Click **Next**. Choose the schema file to use and click **Next**. Select the Assessor Messageset that you created and click **Next**. Check the boxes to select global elements → **Finish**.

You have created a new Message Definition File (.mxsd). Repeat the steps for each schema file. You should end up with ten error messages due to naming clashes (Figure 9-23).

	C	!	Description	Resource
⊗			A Message Set can not contain two messages with the same name: 'requestAssessorAvailability'	AssessorAvailability(3).mxsd Ⓐ
⊗			A Message Set can not contain two messages with the same name: 'requestAssessorAvailabilityResponse'	AssessorAvailability(3).mxsd Ⓐ
⊗			A Message Set can not contain two global complex type definitions with the same name and (target namespace or cha...	AssessorReport(8).mxsd
⊗			A message rendered as an XMLElement, 'requestAssessorAvailability', may not duplicate the XML name of another mess...	Availability(4).mxsd Ⓐ
⊗			A message rendered as an XMLElement, 'requestAssessorAvailabilityResponse', may not duplicate the XML name of ano...	Availability(4).mxsd Ⓐ
⊗			A Message Set can not contain two global complex type definitions with the same name and (target namespace or cha...	Availability(4).mxsd
⊗			A Message Set can not contain two messages with the same name: 'requestAssessorAvailability'	Availability(4).mxsd Ⓐ
⊗			A Message Set can not contain two messages with the same name: 'requestAssessorAvailabilityResponse'	Availability(4).mxsd
⊗			A Message Set can not contain two global complex type definitions with the same name and (target namespace or cha...	DeliverAssessment(7).mxsd
⊗			A Message Set can not contain two global complex type definitions with the same name and (target namespace or cha...	DeliverAssessmentResponse(7a).mxsd

Figure 9-23   Errors from importing message definitions into the same message set

Resolving name and type duplications

The easiest errors to resolve are the six marked A. These are caused by flow 3 and flow 4 using the same message names. The high level message elements are not required as the message we shall actually use is the SOAP envelope.

The remaining four errors are caused by duplicate definitions of the types CarDetails and AssessorAckMessage within the same namespace. In this case we are lucky. These are pure duplicates and we can delete one of the duplicates, replacing it with a reference. If the types had the same names and namespaces, but been different types, then we would have had only two options. Get one of the teams to change their definitions at source and make changes wherever this impacted, or use multiple message sets to hold the conflicting types.

Resolve duplicate message names

This error is caused, despite the messages being in different namespaces, because Version 5 of WebSphere Business Integration Message Broker does not use the namespaces in this instance to distinguish the messages. This might be resolved in version 6.

To resolve duplicate message names, follow these steps:

1. Open the AssessorAvailablity(3) message definition file, and select the messages folder. In Figure 9-24 on page 301, we are using the outline view to do this.

2. Delete the requestAssessorAvailablity and requestAssessorAvailabilityReponse messages and save the definition file.

The number of errors should reduce to four.
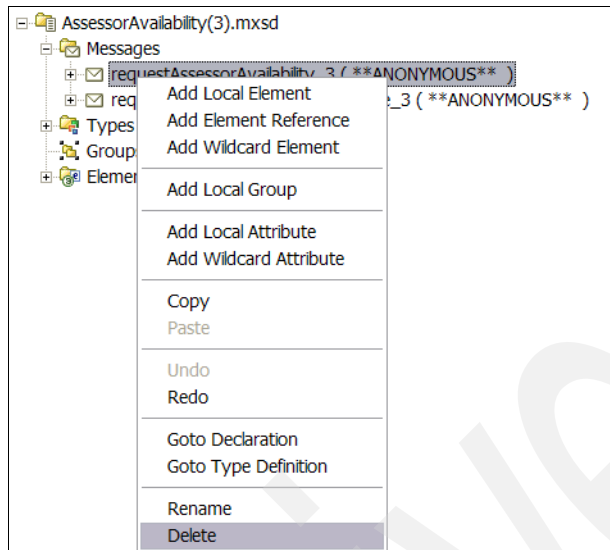
Figure 9-24   Deleting duplicate messages

Resolve duplicate types

The procedure to do this is a little trickier. Figure 9-25 shows the problem.
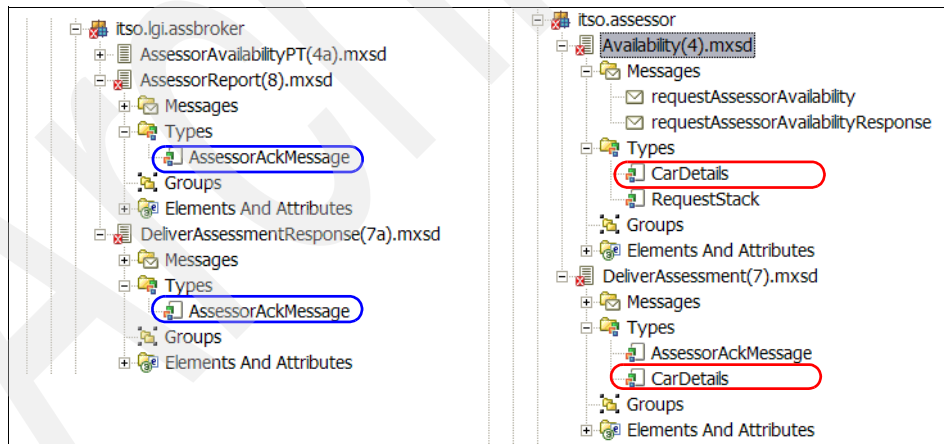
Figure 9-25   Type clashes

AssessorAckMessage is duplicated within the namespace itso.lgi.broker, circled on the left. CarDetails is duplicated in itso.assessor, circled on the right. Note we have a duplicate of AssessorAckMessage in itso.assessor, but this is no problem

because it is in a different namespace from the other definitions of AssessorAckMessage.

> **Important:** Just because the namespace difference allows us the duplicate definition of AssessorAckMessage, does not mean we necessarily have clean code. The definitions or meanings could be different. But this is business as usual when doing enterprise application integration. Assume nothing. These structures and their definitions could have come from different organizations, or reflect different versions. Naming and namespaces cannot necessarily be relied upon.

We shall delete the AssessorAckMessage type from AssessorReport(8) and replace it with a reference to the definition in DeliverAssessment(7).

1. Open the **Outline** view for **AssessorReport(8)**, and observe where the type definition for AssessorAckMessage is defined and where it is referenced. See Figure 9-26.
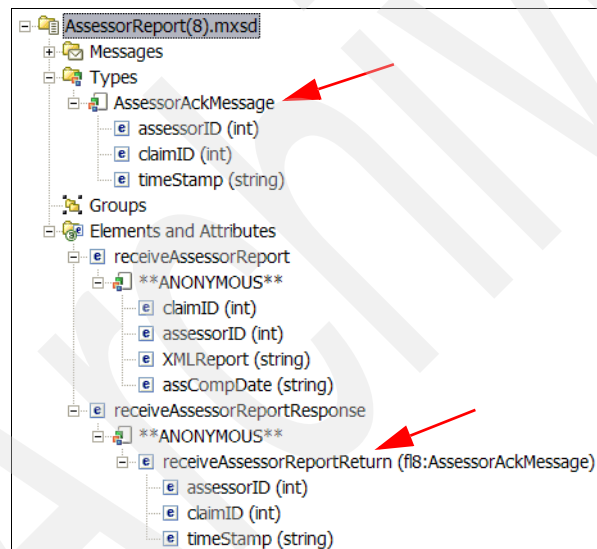
Figure 9-26   AssessorAckMessage type in AssessReport(8) message

2. Delete the type, responding **OK** to the pop-up message confirming the receiveAssessorReportReturn will also have to be deleted.

3. With the **receiveAssessorReportResponse** message selected, right click, → **Add Local Element**. Type `receiveAssessorReportReturn` to reinstate the acknowledgement element.

4. In the message definition editor window the new element will be displayed. Click **string** in the **Type** column, and from the spin box select **More...**

5. Select the **AssessorAckMessage** type, and the qualifier **http://assessor.itso.** Figure 9-27 shows the reconstructed response message.

Figure 9-27   Reconstructed response message receiveAssessorReportResponse

6. Saving the message definition file should reduce the outstanding errors to two.

Follow a similar procedure to remove one of the CarDetails definitions.
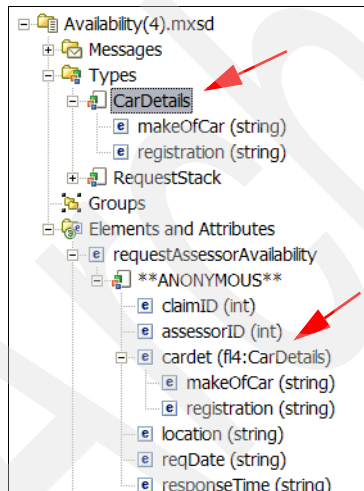
Figure 9-28   Definition and reference to CarDetails in the Availability(4) message

1. Delete the CarDetails type, responding **OK** to the warning pop-up message.

2. Open the **requestAssessorAvailability** element in the message definition editor window → right-click ****Anonymous**** → **Add Local Element** and type `cardet`.

3. Select its Type, **string**, and select **(More...)** from the spin box → Select **CarDetails** as the type, and **http://assessor.itso** as the qualifier → **OK.**

4. Drag the element **cardet** to its correct position between **assessorID** and **location.** The resulting element is shown in Figure 9-29.

Structure	Type	Min Occurs	Max Occurs
requestAssessorAvailability			
ANONYMOUS			
claimID	int	1	1
assessorID	int	1	1
cardet	fl4:CarDetails	1	1
location	string	1	1
reqDate	string	1	1
responseTime	string	1	1

Figure 9-29   Reconstructed requestAssessorAvailability element

5. Save the message definition file. All the errors should now be fixed.

9.4.6 Customizing the SOAP MDF (soap11.mxsd)

We have a number of choices how to define the SOAP envelope around the schemas for each of the messages.

1. Add the sufficient SOAP tags to parse the incoming messages and create outgoing messages using knowledge of the SOAP specification without recreating the rules and constraints for correctly formed SOAP messages.

2. Use the SOAP schema we imported to define our SOAP messages. We can use this schema to check the validity of incoming SOAP messages and make sure our own SOAP messages are well-formed.

   The message definition file created with the schema will have warning messages because the MRM message model is not an exact match for XML schemas. We can deal with this in a number of ways.

   a. Ignore the warnings. There is a filter setting in the tasks view to remove warnings from the tasks, or each warning can be deleted.

   b. Zap the original SOAP schema file, removing the causes of the warnings.

   c. Edit the message definition file using the broker's editor, and improve the validation checks using the MRM message model based on the wording of the SOAP specification which augments the rules specified in the schema definition.

We take option c, editing the message definition file to improve the validation of the SOAP messages. There are two steps:

1. Create the SOAP message definition file from the SOAP schema.

2. Modify the SOAP message definition to remove warnings and "improve" the validation checks.

Create the SOAP message definition file[2]

The SOAP 1.1 schema will cause an error when it is converted into a message definition file, so we need to modify it before proceeding. The broker does not support list attributes. Open the imported SOAP 1.1 schema in the broker schema editor, switch to the source view and make the change shown in Example 9-8. The line (<xs:list ... >) is commented out and the lines beginning with (<xs:restr ... >) are added.

Example 9-8   Changing the SOAP message definition file

```
<xs:simpleType name="encodingStyle">
    <xs:annotation>
        <xs:documentation>'encodingStyle' indicates any canonicalization
conventions followed in the contents of the containing element. For example,
the value 'http://schemas.xmlsoap.org/soap/encoding/' indicates the pattern
described in SOAP specification</xs:documentation>
    </xs:annotation>
    <!--    <xs:list itemType="xs:anyURI" /> -->
    <xs:restriction base='xs:string'>
        <xs:pattern value='http://schemas.xmlsoap.org/soap/encoding%' />
    </xs:restriction>
</xs:simpleType>
```

Create a new message definition file in the Assessor Messageset project by converting the modified SOAP 1.1 schema. Call the message definition SOAP11. There will be 13 warnings.

Note: For the SOAP.xsd file, only include the Envelope global element from which to create a message. This is done so that message flows can nominate this message as the one to be processed. Leave the Header, Body, and Fault boxes unchecked, as shown in Figure 9-30 on page 306.

---

[2] This technique was invented by Pete Edwards and Mick Lickman, see *" Merging disparate IT systems, Part 12: Model Web services with WebSphere Business Integration Message Broker"*, found at http://www-128.ibm.com/developerworks/ibm/library/i-merge12/
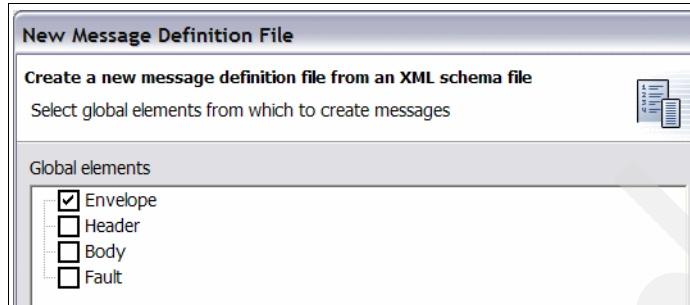
Figure 9-30   Select just the envelope in the SOAP schema

Create the SOAP wrapper for the messages

Use the message definition file editor perform the following edits which will remove the warning messages and tighten up the validity checking of SOAP messages.

1. Remove Wildcard elements from elements **Envelope**, **Header**, **Body** and **Detail** (child of element **Fault**). See Figure 9-8.

2. Remove the Wildcard attribute from the **Body** element.
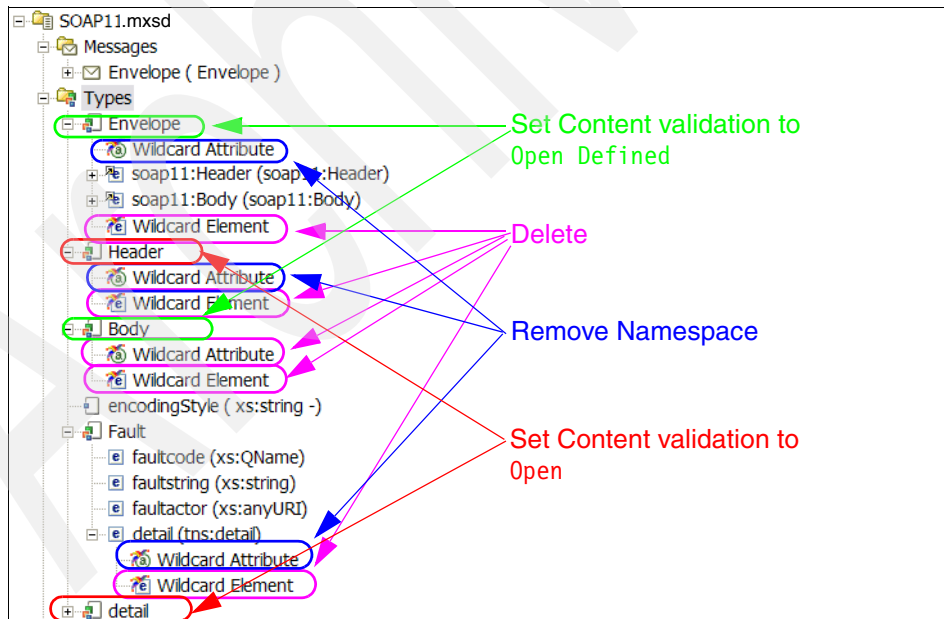
Figure 9-31   Identified items to edit in SOAP 1.1 schema

3. Remove namespaces from the three remaining wildcard attributes shown in Figure 9-31. To do this, select each attribute in term and open the properties tab rather than the overview tab in the message definition editor. See Figure 9-32.

> **Tip:** The quickest way to do this is using the outliner to select each attribute in turn, then you only need to switch to the properties tab once.
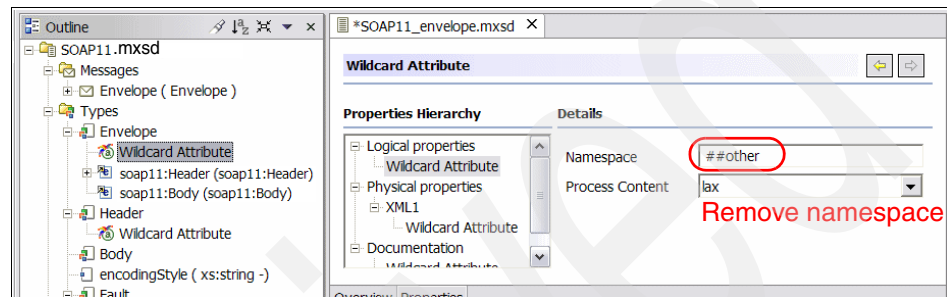
Figure 9-32   Removing namespace from wildcard attributes

4. Set the **Content Validation** of **Envelope** complex type to `OpenDefined`, from its original setting of **Closed**. See Figure 9-33. This means that only elements and attributes defined in the message set are allowed as children of **Envelope**. The explicit children are **Header** and **Body**.
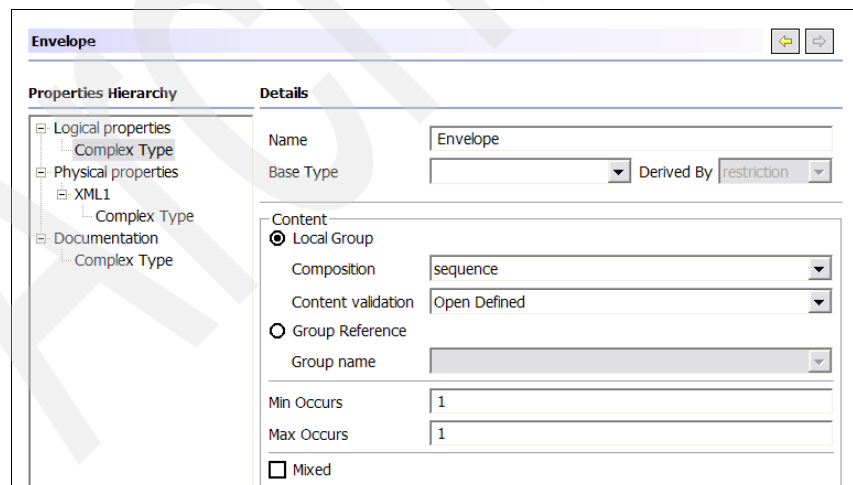
Figure 9-33   Envelope complex type Content validation set to Open Defined

5. Set the **Content Validation** of **Header** and **Detail** Complex types to **Open**, from their original setting of **Closed**. This means that any elements and attributes are allowed. SOAP headers are optional so **Min Occurs** for the header could be set to zero - but this is not supported by the MRM so we shall leave the value as 1.

6. Change the **Body** element's **Complex type** to Composition **choice** and Content validation **OpenDefined**. This allows any one of the imported schema's elements to be used as children of Body, but disallows elements not defined in any schema.

7. Delete the **Pattern facet** of the **mustUnderstand** global attribute, as it is not required and its deletion removes a warning generated by the Broker Toolkit. See Figure 9-34.
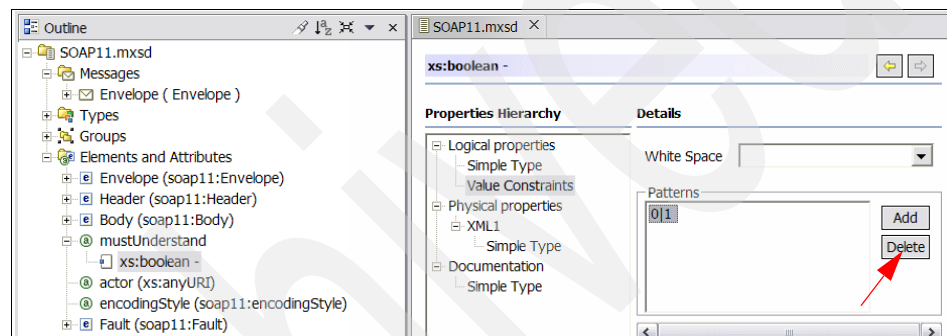
Figure 9-34   Remove pattern facet from mustUnderstand

Note: A pattern facet is a definition of what values a variable can assume. Rather like an enumerated type. A boolean has a pattern facet of 0l1

8. Add the **Fault** element of the SOAP schema as a child of **Body**. To do this, select the **Body** element → right-click and select **Add element reference.** See Figure 9-35 on page 309.
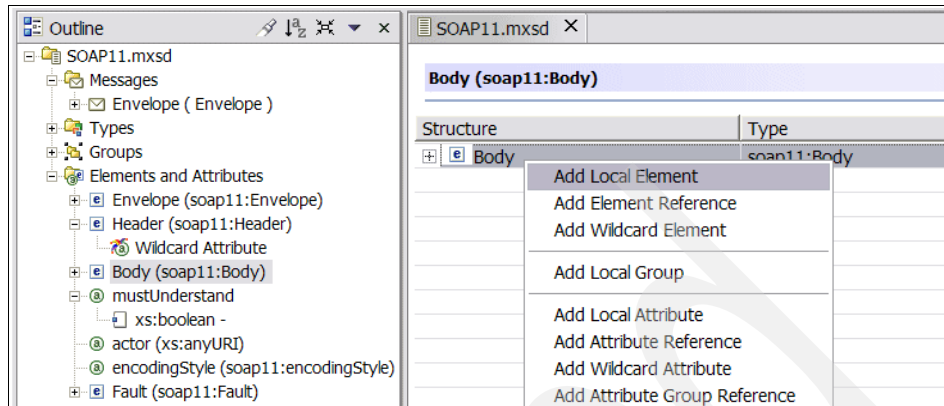
Figure 9-35   Adding element reference to Body

9. Scroll to **soap11:Fault** element (Figure 9-36).
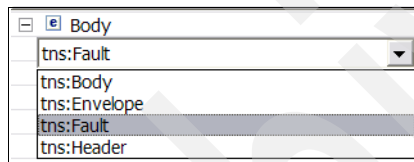
Figure 9-36   Selecting fault element reference

10. Set the **Min Occurs** of **Fault** to 0 on the same screen and save the changes. Observe that all the warnings have been fixed.

With these steps we created the SOAP wrapper for our messages. Now, we need to add each of the messages to the wrapper.

9.4.7 Create the SOAP messages

We have the SOAP wrapper there are a couple of steps to complete creating the SOAP messages in the broker:

1. Save the SOAP wrapper for use another time to save redoing most of this editing procedure again.

2. Combine the Assessor messages into the Body of the SOAP wrapper.

Save the SOAP wrapper

We export the SOAP wrapper as an XML schema file and it can be used with other message set projects in the future.

1. Select the **SOAP11.mxsd** message definition file in the Resource navigator → **New** → **XML Schema**. Select the **XML1** wire format and ensure the SOAP11.mxsd file is selected. Check **Strict Generation** → **Next** → Create a new folder to hold the XML schema (e.g. `Generated`) → **Finish.**

.

> **Attention:** The Header Complex type, created when the patched SOAP11.xsd file is imported, has **Content Validation** defined as closed. Define it as open. Also, fix the content validation of the other three types. The warning clears when you save the file.

Convert the Assessor messages into SOAP messages

The final task in creating the message set is to combine the SOAP definition with each of the ten Assessor messages. We do this by importing the Assessor message definition files into the SOAP message definition files and then make the assessor messages child elements of the SOAP body.

Start with the AllocateAssessment(6) message definition.

1. Import the MDFs representing the Assessor services into the SOAP MDF.

   a. Select SOAP11.mxsd in the Resource navigator → select the properties tab in the Message Definition editor, as in Figure 9-37.
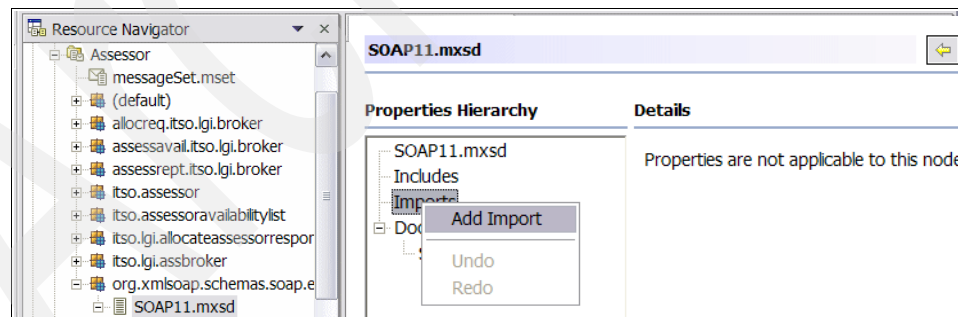
Figure 9-37 Import Assessor messages into SOAP message

   b. Right-click **Imports** → **Add ...** → Select **AllocateAssessment(6)** message definition file → **Finish.** See Figure 9-38 on page 311.
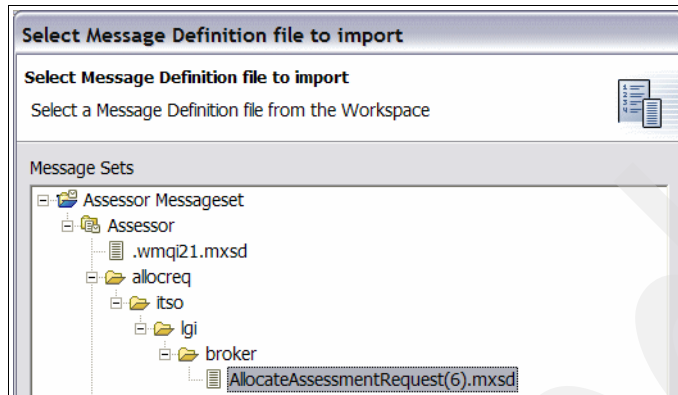
Figure 9-38   Importing Assessor message definition into the SOAP message definition

   c. Repeat this for all the message definition files in Figure 9-39.
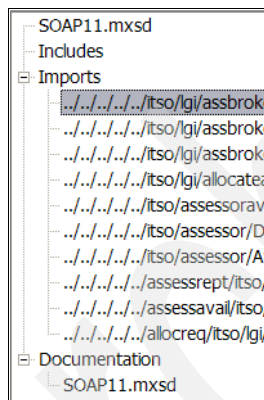
Figure 9-39   Imported all the message definition files

2. Now we have addressability from the SOAP MDF to the other Assessor messages in this project, add the global elements of the Assessor messages as children of the SOAP Body.

   a. With the SOAP11.mxsd Message Definition file still selected open the **Overview** tab and expand the **Messages** → **Envelope** structure as in Figure 9-40 on page 312.

   b. Right-click the **Body** → **Add Element Reference** → and go through all the elements.

   c. Set the minOccurs of each to 0 because each of the structures is optional.

Figure 9-40   Adding Assessor elements as children of Body and setting Min Occurs

3. Repeat this procedure with the other 16 top level elements as in Figure 9-41 and Figure 9-42 on page 313.

Figure 9-41   Wrapped up all seventeen assessor messages

Structure	Type	Min Occurs	Max Occurs
☐ SOAP11.mxsd			
☐ Messages			
☐ Envelope	Envelope		
Wildcard Attribute			
⊞ soap11:Header	soap11:Header	0	1
☐ soap11:Body	soap11:Body	1	1
⊞ soap11:Fault	soap11:Fault	0	1
⊞ fl3:requestAssessorAvailability		0	1
⊞ Types			
⊞ Groups			
⊞ Elements and Attributes			

```
Copy
Paste

Undo
Redo

Goto Declaration
Goto Type Definition
```

Figure 9-42   Checking the source of the element reference

Tip: Getting all these element references into the SOAP wrapper is hard to do right and needs careful checking.

It is difficult to tie up the names in the element references with the names of the elements:

► The prefixes we added to distinguish the messages have been shortened; the prefixes of duplicate namespaces are lost.

► There are no namespace qualifiers to assist you choosing the right element. The best technique is to right click the element reference once it has been inserted → **Go To Declaration** and check that the reference is to the right element declaration. See Figure 9-42.

► Be systematic. We added the references in the order of the flows, keeping sight of the flows Figure 9-11 and Figure 9-12 on page 278.

► Count the number of references at the end. There are 17. This should match the number of element definitions. Three of the flows are one-way SOAP messages, which is why we are short by three from twice times the number of interfaces.

► Check there are no duplicate element references.

► Check all the element references are children of Body, and none have adopted another parent.

► Check all the minoccurs are 0. These are all optional elements.

9.5 Implementation of the database tables

The database for these tables is to be located on SAH414A, local to the message broker. First create the database and schema on SAH414A, then create the tables, and finally import the schema and tables into WebSphere Business Integration Message Broker Workbench to use in message flows.

9.5.1 Create the ASSESSOR Database

To create the ASSESSOR database, follow these steps:

1. Start the DB/2 Control Center on SAH414A. Start the Create Database wizard as shown in Figure 9-43.
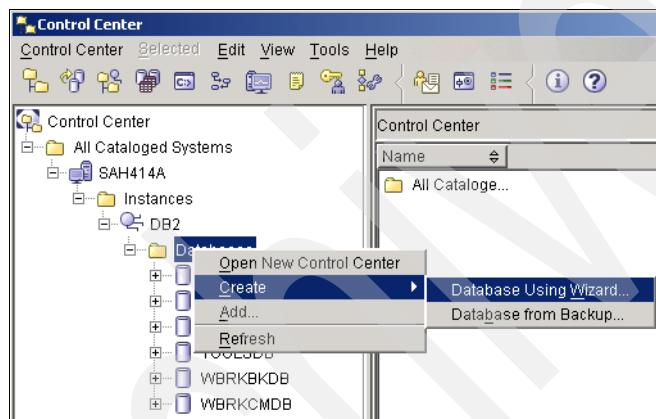
Figure 9-43   Create Database wizard

2. Name the database `ASSESSOR`. See Figure 9-44.
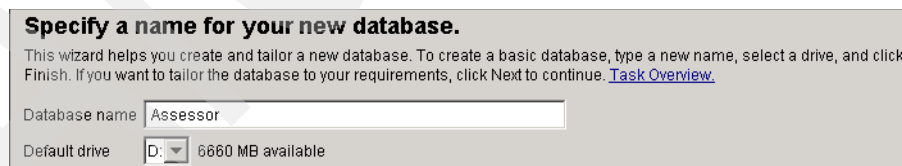
Figure 9-44   Creating the ASSESSOR database

Because the real memory is limited on SAH414A, we customize the memory used by ASSESSOR as in Figure 9-45 on page 315.
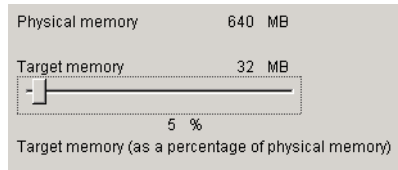
Figure 9-45   Reducing the database target memory

9.5.2 Create the schema and tables

Follow these steps to create the schema and associated tables.

1. Using the DB/2 control center, create a new schema and call it `EMERGE`. See Figure 9-46.
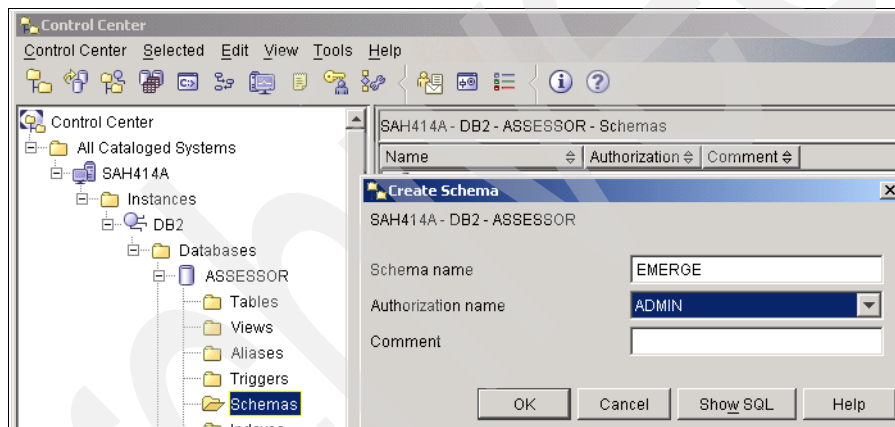
Figure 9-46   Create EMERGE schema

2. Right-click the **Tables folder** → **Create ...** → Select the **EMERGE** schema and call the new table `CLAIMASSESSOR` See Figure 9-47.
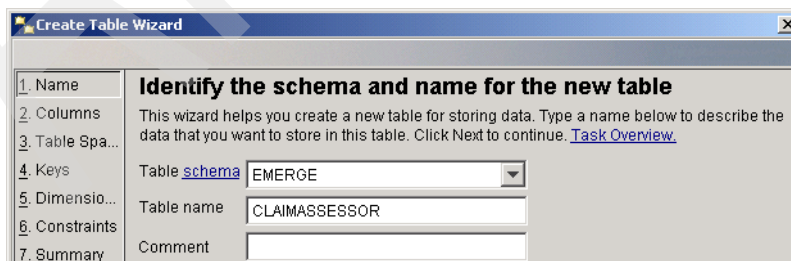
Figure 9-47   Create the CLAIMASSESSOR table

3. Using the data in Table 9-5 on page 280, create all the columns. Apart from the two key columns all the fields are nullable, and none of the character fields are bit. See Figure 9-48.

Figure 9-48   Created columns in CLAIMASSESSOR

4. Define the two key fields, naming the constraint CA1. See Figure 9-49.

Figure 9-49   Defining keys on CLAIMASSESSOR

5. Repeat this process using the data in Table 9-6 on page 280 and Table 9-7 on page 281 to create the ACTIONASSESSOR table in Figure 9-50.

Figure 9-50   ACTIONASSESSOR table

The RESOLVEASSESSOR table shown in Figure 9-51.

Column name	Data type	Length	Nullable
ASSESSORID	INTEGER	-	No
ASSESSORTYPE	CHARACTER	10	Yes
ASSESSORHTTPADDRESS	LONG VARCHAR	-	Yes
LGIBROKERHTTPADDRESS	LONG VARCHAR	-	Yes

Figure 9-51   RESOLVEASSESSOR table

9.5.3 Connect to the database from the broker workbench

To connect the database, follow these steps:

1. Open the data perspective in the workbench and right-click in the **DB Servers** view → **New connection (**Figure 9-52).
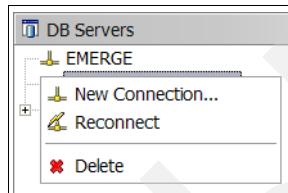
Figure 9-52   New database connection in the workbench

2. Complete the form as shown in Figure 9-53 on page 318.

Figure 9-53   Defining a database connection

Import the database into the message broker projects

Having created a connection from the workbench to the ASSESSOR database, we want to create schemas and tables directly from the workbench. To do this, we need to import the database connection into a workbench project, which we will use to develop the tables.

1. Create a Message Flow project called `ASSESSOR database.` We found that placing a database definition in a simple project led to warning messages about unresolved messages. Placing the definition in a message flow project circumvented this behavior. When you create the message flow projects, associate each one with this new folder in the Broker Application Development perspective.

2. Click **ASSESSOR** in the DB Servers view (reconnecting if necessary) → **Import to folder** → Select the **ASSESSOR** database folder → **Finish** (**Figure 9-54).**
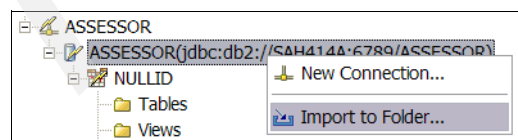
Figure 9-54   Importing a new database connection into the workbench

3. Right-click the **ASSESSOR_ASSESSOR.dbmxmi** file in the ASSESSOR database folder → **New** → **Other** → **Schema Definition** → **Next** → and call the schema EMERGE → **Finish.**
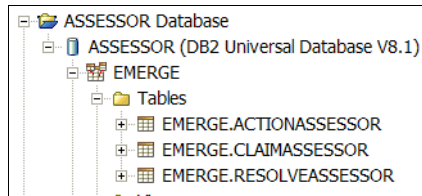
Figure 9-55   Assessor database project

The ASSESSOR database has been imported into the workbench. Database references in ESQL statements will autocomplete.

9.6 Create the message flows

This section breaks down the large task of creating all the message flows into four smaller steps.

1. Create the message flow projects and dependences.

2. Create the message flow files.

3. Wire up the message flows (Sections 9.6.2 to 9.6.5) and supply the properties for the nodes.

4. Write the esql code for any compute nodes.

9.6.1 Create the Message Flow projects and dependencies

Review the design for the organization of the message flows in 9.3.2, "Message flows and transport independence" on page 275.

Good organization of message flow projects makes it easier to understand and manage the solution. There are two sets of transport independent flows: the availability flows, 3, 3a, 4 and 4a and the assessment report flows, 6, 6a, 7, 7a, 8 and 9. We use two message flow projects for the transport independent flows so it is clear whether we are working with the Availability interactions or the Report interactions.

Corresponding to the two sets of transport independent flows will be two sets of transport dependent flows for the SOAP/Http transport. Finally there will be a message flow project for the common SOAP/Http flows.

Table 9-9 on page 322 lists all the message flow projects and flows we will be creating and their dependencies. The relationships between the flows and between the flows and the message set needs to be defined so that references to flows and sets can be correctly resolved. We also define packages of message flows so that potentially duplicate flow names are resolved correctly. The packages are called *broker schemas*.

All the flows share a common broker schema so that symbolic references are resolved within the same schema, and not confused with same-named resources in other projects.

The dependencies between message flow projects and message set projects are defined to the workbench, either when creating a new project, or at a later time, using the workbench GUI.

1. Create the first message flow project (Figure 9-56). Click **File** → **New** → **Message Flow Project** → **Type** `AvailabilityFlows` → **Next.**
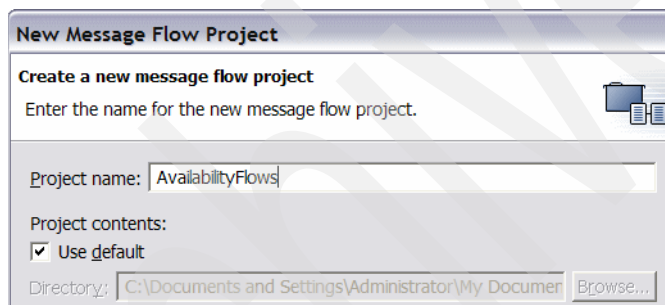
Figure 9-56   Creating a new message flow project

2. Create the other message flow projects listed in Table 9-9.
3. Add the dependencies from Table 9-9 by selecting each flow project, the message set, and opening its properties. See Figure 9-57 on page 321.
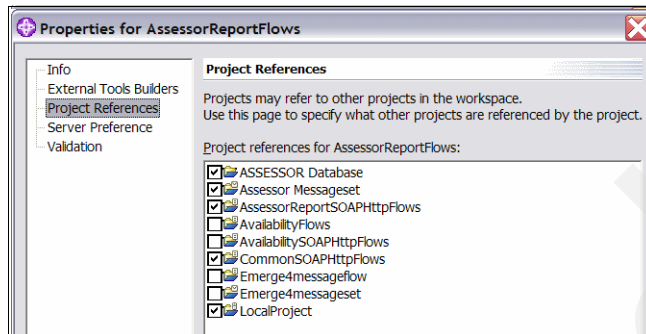
Figure 9-57   Dependencies in the properties view

Table 9-9   Message Flow Projects and Message Flows

Message Flow Project	Message Flow	Message Set and Database
Group B CommonSOAPHttpFlows Dependencies: A	Common.esql[a]	
	Fault	
	Reply	
	ClientError	
Group C AssessorReportFlows Dependencies: Group A, B, D	Flow6	
	Flow6a	
	Flow7	
	Flow7a	
	Flow8	
	Flow9	
Group D AssessorReportSOAPHttpFlows Dependencies: Group A, B, C	Flow6aAck	Group A: Assessor Message Set ASSESSOR Database
	Flow9Ack	
	Input6	
	Output6a	
	Output7	
	Output9	
Group E AvailabilityFlows Dependencies: Group A, B, F	Flow3	
	Flow3a	
	Flow4	
	Flow4a	
Group F AvailabilitySOAPHttpFlows Dependencies: Group A, B, E	Flow3aAck	
	Input3	
	Output3a	

a. This is a common .esql module rather than a flow. We use it later.

4. Create the broker schema in each of the message flow projects.

   a. **File** → **New** → **Broker Schema** → Select one of the message flow
      projects and name the schema `proxyAssessorSystem` → **Finish.**
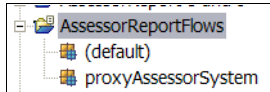
Figure 9-58   Broker schema

   b. Copy and past the schema into the other message flow projects.

5. Create all the Message flow files from Table 9-9 placing the flows in the
    Broker schema package - see Figure 9-59.
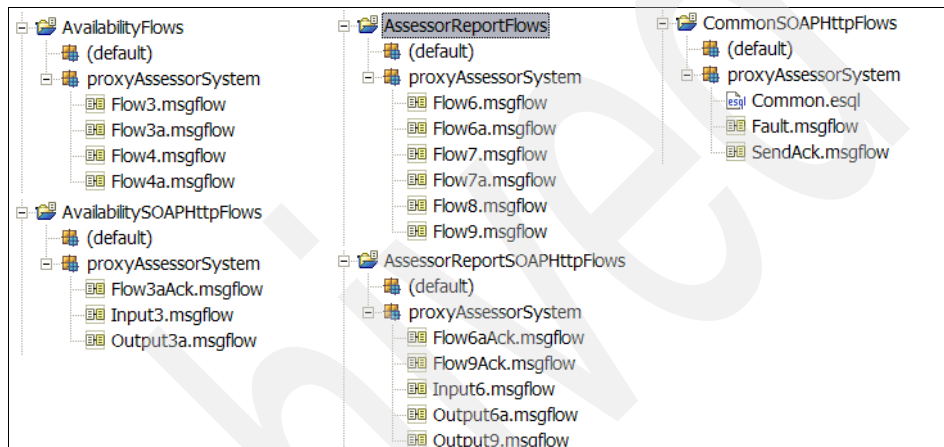
Figure 9-59   Message Flows

6. Set up the project dependencies for the message set and message flow
    projects as indicated by the table.

9.6.2 Create message flows

In sections 9.6.3 "Create the CommonSOAPHttpFlows" on page 323 to 9.6.5
"Create AssessorReport flows" on page 340 we wire and configure all the
message flows, starting with the common flows, and then the Availability and
Report interactions. All the parameters for the flows are configured, as well as
the flow topology, leaving a collection of ESQL files to program.

9.6.3 Create the CommonSOAPHttpFlows

The common flows to develop are the Fault and the Reply subflows. We also
need to create a skeleton Validate SQL routine in the common.esql file so that it
can be referenced in all the flows.

Fault

The fault flow returns a correctly formatted SOAP fault message containing a useful description of the cause of the fault and the correct SOAP fault code.

This subflow is used for basic error processing in all of the message flows. It is invoked from the input node of any message flow when an exception has been encountered. We use the same fault routine in all of our message flows. This fault routine is wired from the Failure and Catch terminals of the Input nodes. All other failure terminals in the message flows remain unconnected, so any exceptions are routed through the input node to the fault routine. See WebSphere Business Integration Message Broker online help documentation: *Handling errors in message flows* for more information.
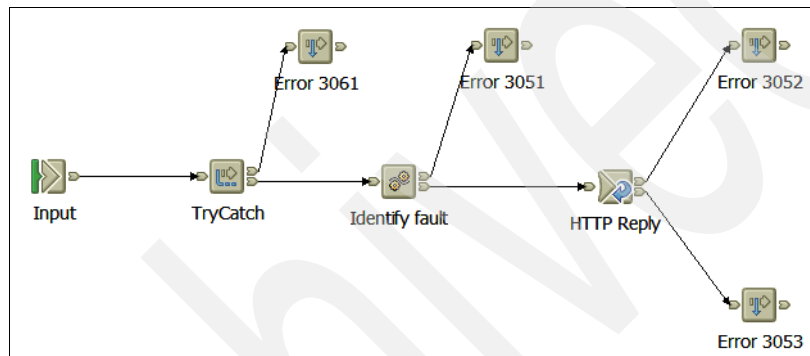
Figure 9-60   Fault flow

1. Like all subflows, begin the flow with an input node.

2. Create a TryCatch node. We do not want any uncaught errors recursively causing the Fault flow to be reinvoked, so we add a TryCatch node and direct any faults it catches to a trace node.

3. Create a compute node called `Identify fault` to create the fault message. The implementation of Identify fault, in common with all the compute nodes inserted into the flows at this stage, are described in the section on developing ESQL.

   For now, right-click the **Identify Fault** node → select **Open ESQL** , rename the ESQL module `Fault_Identify_Fault` and save the created ESQL file. Make sure the ESQL module is identified in the ESQL Module parameter shown in Figure 9-61 on page 325.
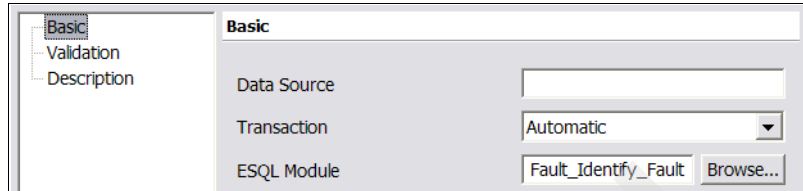
Figure 9-61   Setting ESQL Module path

> **Tip:** Give your ESQL modules distinctive names. A good practice to follow is to start with the Flow name and suffix with the Compute node name. That way the esql code is easy to find and you are less likely to make changes to the wrong ESQL module when opening the ESQL files directly from the resource navigator

4. Create an HttpReply node to return the fault.

5. Create three trace nodes to help debug the flow in the future. There are many conventions to follow to help debugging. The one we adopted is to use the reserved message numbers 3051 to 3099 in the WebSphere Business Integration Message Broker message catalog for error messages and output into the Windows event log. Each trace node is named with its error number to help identify the source of the error quickly in the flow. See Figure 9-62.
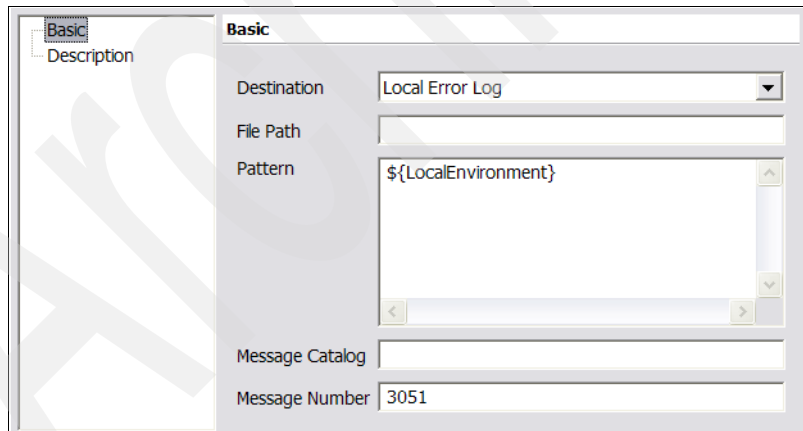
Figure 9-62   Configuring the trace node

6. Wire up the nodes and save. Pick the connection tool from the palette.

Figure 9-63   Use the connection tool to connect nodes

Reply

The reply subflow is simply an HttpReply node.
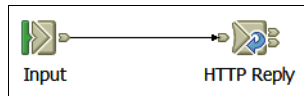
Figure 9-64   Reply subflow

The reason for making the reply a subflow is two-fold:

1. For packaging, by using a different transport specific flow project in place of the HttpSOAP project, the subflow linkage would remain the same, but a different reply node used.

2. The subflow may be more complex with other transport types, and the complexity is encapsulated by the Reply flow.

ClientError

The ClientError flow simply traps any errors returned to the broker by Web services, or detected in the Client flows and sends them to the event log (Figure 9-65).
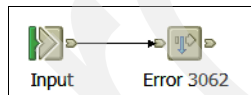
Figure 9-65   ClientError flow

By making the call to the trace the error indirectly to a subflow, more sophisticated error handling can be added later without impacting the main flow design. The 3062 trace node outputs the exception tree and other useful data. Figure 9-66 on page 327 shows how to output all the broker trees.

Figure 9-66   Setting the pattern values for the ClientError trace node

9.6.4 Create the AvailabilityFlows

Figure 9-67 shows the Availability flows that need to be defined.
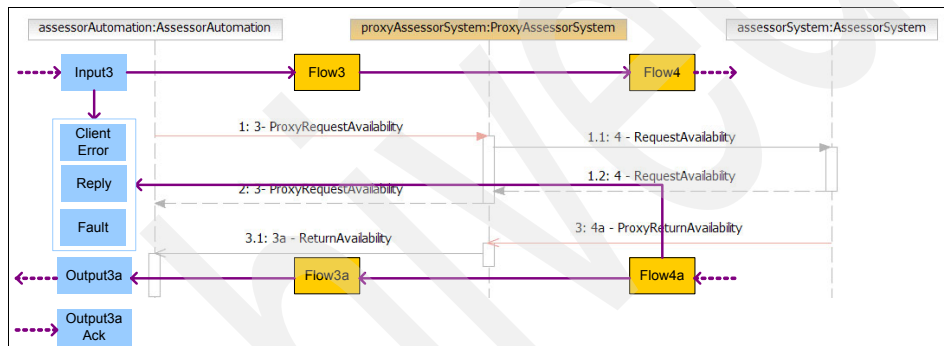
Figure 9-67   AssessorAvailability flows

Input3

The Input3 receives the RequestAvailability message from the ExternalClaimAssessors process, and passes it to Flow3 for validation and processing. Exceptions are caught and passed to the Fault routine to return a SOAP fault Figure 9-68).
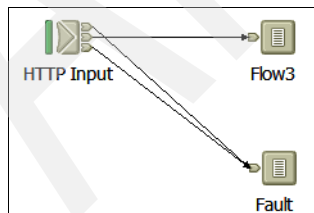
Figure 9-68   Input3 Flow

1. Configure the Http Input node as follows:

   a. On the Basic Tab, set the **URL Selector** to the value supplied by the solution architect in the AssessorAvailablity(3).wsdl file: `http://SAH41403:7080/RequestAssessorAvailability` and the timeout value to 60 seconds.

   > **Attention:** The TCP/IP port, 7080, is the default Http listener port for the message broker. It is set using the mqsicreatebroker command (-P portnumber) and changed using mqsichangebroker

   b. On the Default Tab, set the Message set properties as in Figure 9-69.

Message Domain	MRM
Message Set	Assessor (PIJ0MIK002001)
Message Type	Envelope
Message Format	XML1
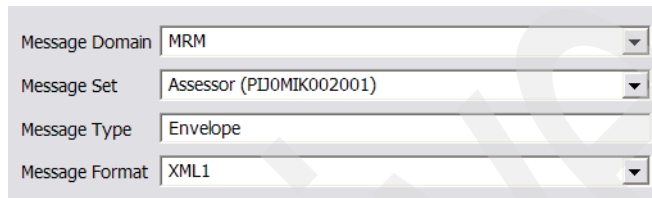
Figure 9-69   Message set properties for Http Input node.

   c. On the Validation Tab, set **Validate** to `Content and Value`.

Flow3

This message flow receives a message from the Input3 flow, validates it, sends an acknowledgement reply back, and invokes flow4 to fan the message out to the assessors (Figure 9-70).
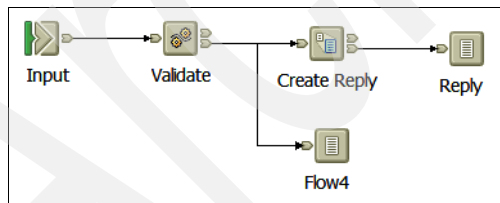
Figure 9-70   Flow 3 skeleton

1. Create the following nodes:

   a. A compute node, Validate, to check the input message

   We write the ESQL later in 9.7.4, "ESQL Error handling code" on page 361.

   i. For now, open ESQL on the right mouse button menu and save the default code. See Figure 9-71 on page 329.
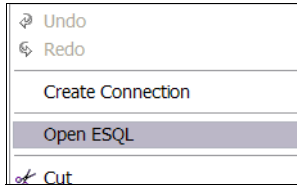
Figure 9-71 Select Open ESQL to create default ESQL code

This removes a compile error.

ii. On the basic tab of the compute properties page (Figure 9-72), set the Data Source to `EMERGE` and leave the Compute Mode to Message. This is the default option that enables any modifications of the message folders in the compute mode to be propagated to subsequent nodes. We set the compute mode to Message and LocalEnvironment on the nodes that need to modify and propagate aggregation information which is held in the LocalEnvironment tree as well as the message tree.
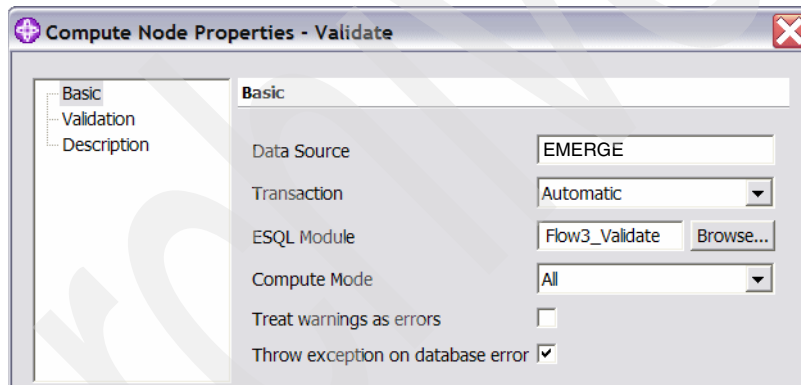
Figure 9-72 Setting basic properties of a compute node

iii. On the Validation tab, set Validate to **Content and Value** as in Figure 9-73 on page 330. Again this can be turned off when testing is complete to improve runtime performance.
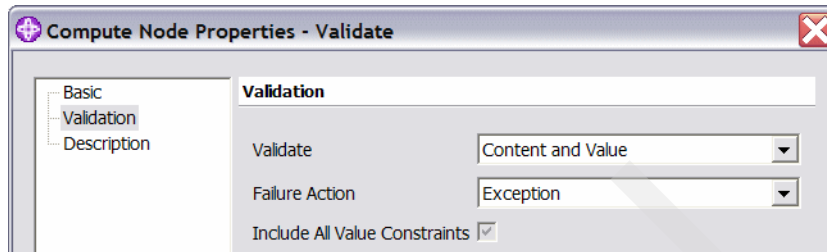
Figure 9-73   Setting validation checking

> **Note:** From now on, we assumeThe setting of the Compute Mode and
> the Validation action on all nodes that propagate folders will be
> assumed from now on.

    b. Create a mapping node, Create Reply, to map the response into the reply
        message.

    c. Create a Flow4 subflow. Create a dummy flow4, for now, by adding an
        Input node to Flow4 and saving it. Flow4 can then be added to Flow3 as a
        subflow before the details of Flow 4 are defined. Drop Flow4 into Flow3.

    d. A Reply subflow. Drop the Reply flow into Flow3.

2. Now configure the mapping node.

Configure Create Reply mapping node

Follow these steps:

1. Right-click the **Create Reply** mapping node → **Open Mappings.**

2. Right-click in the source box → **Add Message mapping input.**

    a. Select the **SOAP envelope** → **Next** → **Finish** and do exactly the same in
        the Target box - see Figure 9-74 on page 331.
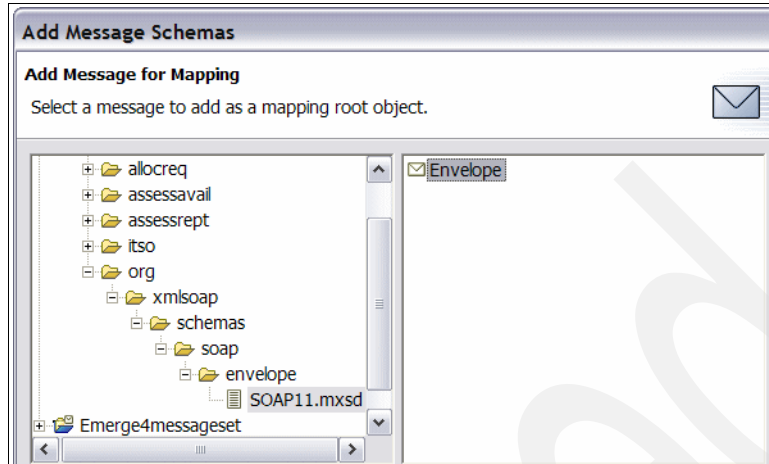
Figure 9-74   Selecting the SOAP envelope in the message mapper

b. Expand the message schemas and map the claimID from source to target.

c. Right click **TimeStamp** → **Create one-sided mapping.**
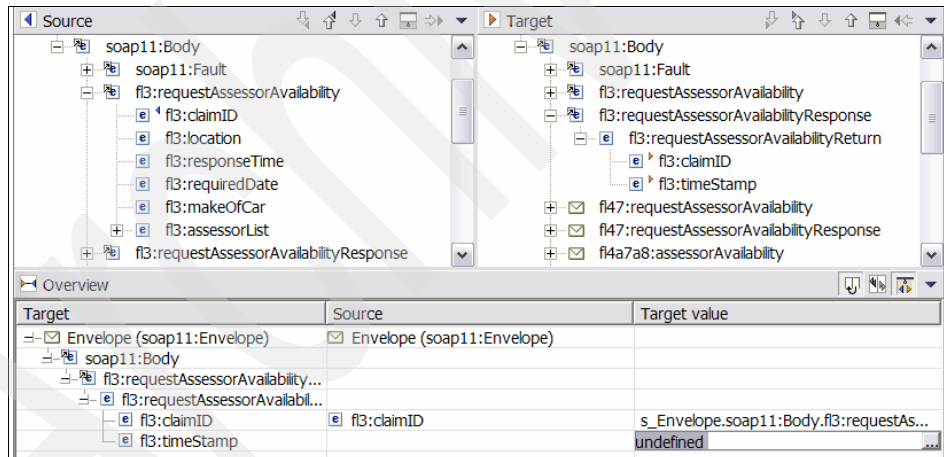
Figure 9-75   Mapping Flow3_Create_Reply

d. Click the **three dots** to the right of **undefined.** Delete **undefined** from the target field box. Select **Date/Time functions** and click the double arrow to the right/ Select **CURRENT_TIMESTAMP** and drag it into the target field box. See Figure 9-76 on page 332.
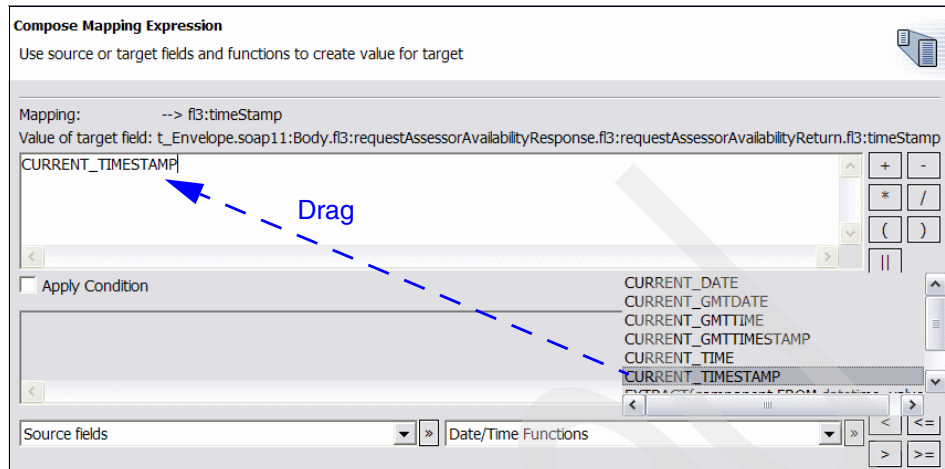
Figure 9-76   Setting the CURRENT_TIMESTAMP

   e. Save the mapping file.

Flow4

This message flow (Figure 9-77) receives a validated message from flow3 and generates one or more messages to be fanned out to assessors.
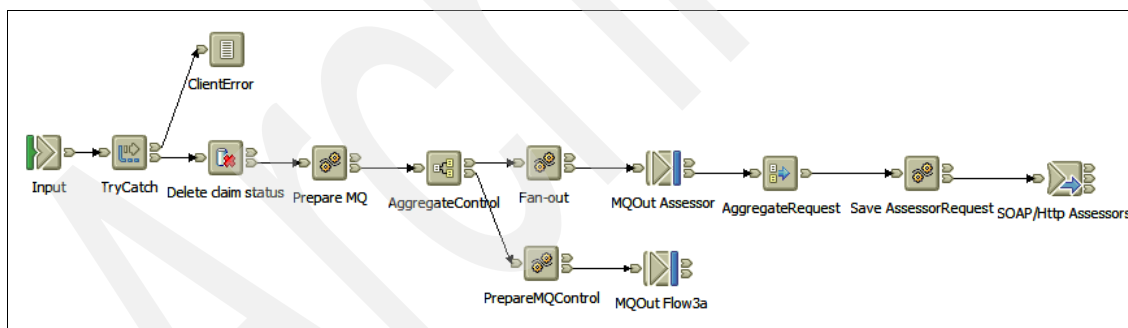
Figure 9-77   Flow 4

It also updates the database table CLAIMASSESSOR to save data to insert into the reply messages returned to the ExternalClaimAssessors process.

   a. The incoming message contains the assessor's URL. The reply to the ExternalClaimAssessors process in flow3a also needs the assessor's URL, so we store it and retrieve it later, based on the claimID and assessorID.

b. We need to store the msgids for all the Flow4 messages so we can correlate them against the replies from the assessors. We cannot assume the replies will contain an WebSphere MQ correlationId because we need to implement the solution across SOAP/Http and other transports.
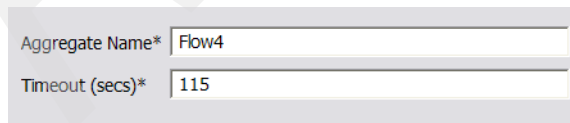
Flow4 generates a unique msgid for the message sent to the assessor by creating a real WebSphere MQ message and sending it both to the aggregation node and storing its MsgId in the CLAIMSASSSESSOR database to regenerate the correlation id for the subsequent aggregation fan-in.

The details of the flow configuration are:

1. We start with a TryCatch node to prevent any errors returning to the broker client. The Broker client is not expecting any fault message as it has already received an acknowledgement. Any errors from here in a handled by the broker by passing control to a common ClientError routine.

2. Use a DataDelete node to remove an old instance of a claim from the database. Note if a SQL delete is performed and no match is found on the claimID, a warning is returned (corresponds to a positive SQL error number). The node is configured by default to ignore warnings. (Treat Warnings as Errors must be *unchecked* in the node properties). Skip to "Configure Delete claim status DataDelete node" on page 336 for instructions how to configure the DataDelete node.

3. The Prepare MQ node removes the HTTP information from the message folders and prepares an MQ folder prior to passing the folders to the Aggregate Control node.

This is a defensive programming step. We are not sure whether the aggregation function is sensitive to seeing different types of folders on different nodes. Because we will be using WebSphere MQ as the interface to the aggregation nodes, we make the message flow look similar to WebSphere MQ to all the aggregation nodes.

4. The aggregate control node names the aggregation (`proxyAssessorSystem`) and supplies a time out for the aggregate reply node. For testing purposes we have set 115 seconds (Figure 9-78).

| Aggregate Name* | Flow4 |
| Timeout (secs)* | 115 |

Figure 9-78   Properties of the Aggregate control node

5. The compute node, Fan out, propagates a new request message for each assessor in the input assessor array. It also stores the destination HTTP URL

to route the message to in the Local Environment. At this stage simply generate default ESQL for the compute node called `Flow4_Fan_Out`: we shall return later to implement the node.

The URL has to be stored somewhere to be used when the Http request message to the assessor is constructed, as the contents of the message folder do not include the URL. We are also going to save the URL in the CLAIMASSESSOR database to be returned to the ExternalClaimAssessor process, as required in its interface. The URL could be saved in the database in this step, and read back to set up the URL destination later in the flow. We would have to do it that way had we chosen to read the request message back from the Assessor queue, rather than wiring the rest of the flow onto the output terminal of the MQOut Assessor node. Passing the URL in the local environment is slightly more efficient, so we adopted that approach.

6. The compute node, PrepareMQControl, propagates the input message to send to the control terminal of the AggregateReply node in Flow3a. Again, generate default ESQL called `Flow4_Control_Message` for now.

7. The MQOutput node, MQOut Flow3a, sends the control message to Flow3a. Set the **Queue Name** property on the Basic Tab to `FLOW3A.CONTROL.` Leave **Queue Manager** blank to use the default queue manager. All the other tabs are left to default.

   – You need to set **Set All** as the **Message context** on the Advanced tab.

8. The MQOutput node, MQOut Assessor, sends the request message to the Assessor queue. The message is never read in, and the queue will build up. To manage this in a production system, the messages could be put with an expiry date, or, rather then wire the output terminal of the node to AggregateRequest, create another MQInput node to read the message back. You could specify the generated MsgId as a correlator to ensure you really read the *right* message back.

   – You need to set **Set All** as the **Message context** on the Advanced tab and check **New Message ID**.

Figure 9-79   Settings on MQOutput Advanced tab

9. In the AggregateRequest node set the **Folder Name** to `Flow4`. This name is used as a folder in the AggregateReply node's compound message to store the reply to this request.

10. To save the request in the CLAIMSASSESSOR table we need to use a Compute Node, Save AssessorRequest, rather than a Database Update node, as one of the fields is mapped from the LocalEnvironment folder rather than from the input message.

   a. Set the data source to EMERGE and generate default ESQL called `Flow4_Save_AssessorRequest` for now. We will add the ESQL later.

11. The HttpRequest node "SOAP/Http Assessors" sends an availability request to each assessor. The property defaults are accepted except where configured as follows:

   a. On the Basic Tab:

   i. Set the **Web service URL** to `http://SAH414A:7080/UNKNOWN.` The assessor URL is set in the compute node - if this URL is used it is an error. We can either implement the URL in the broker, or simply leave it with this distinctive value and find it in the event log as a SOAP fault message.

   ii. Set the **Request Timeout** to 60 seconds for testing.

   iii. Select **Follow Http redirection** as the behavior for http redirection requests.

   b. On the Advanced tab, Check **Replace input message with web-service response**.

   c. On the Default tab, set the message properties as shown in Figure 9-69 on page 328.

   d. On the Validation tab, set **Validate** to **Content and Value.**

12. The acknowledgement response from the assessors can be wired to a trace node for testing. In production this reply might be monitored to measure how available the assessors' systems are.

Configure Delete claim status DataDelete node

1. Right-click the Delete claim status node and in its properties, set the data source to EMERGE.

2. Open the Mappings editor from the properties panel and:

   a. Select the SOAP envelope in the Source box as in the previous section.

   b. Right-click in the Target box and select **Add RDB Table Mapping Output ...**

   c. Assuming you have completed 9.5.3, "Connect to the database from the broker workbench" on page 317, accept **Add database table schemas from workspace** → **Next** → and select the **ASSESSOR_ASSESSOR_EMERGE_CLAIMASSESSOR** table → **Finish.**

   d. Right-click in the Target box → **Set RDB Schema name** → Check the radio button **Use schema name in table definition.**

   e. Map the ClaimID from the source box to the target box to specify the claimID must match to delete the table row.

   f. Save the mapping file. See Figure 9-80.
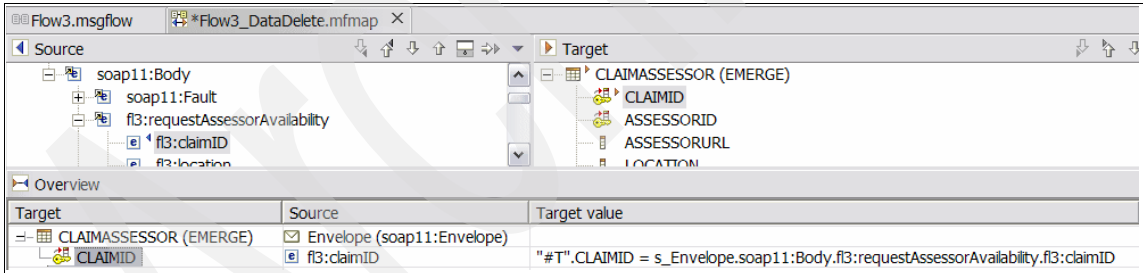
Figure 9-80   Setting the delete condition for the CLAIMASSESSOR table

Flow4a

This message flow receives a Flow4a message from an assessor, validates it, sends an acknowledgment, and invokes flow3a to perform the aggregation. See Figure 9-81 on page 337.
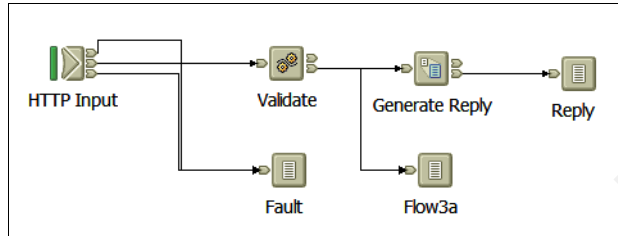
Figure 9-81   Flow4a

1. Create the following nodes:
   a. An HttpInput node to wait for the availability message.
   b. A compute node, Validate, to check the input message.
   c. A mapping node, Create Reply, to map the response into the reply message.

2. Drop the Reply, Fault and Flow3a sub-flow into the flow.

3. Configure the HttpInput node,

   a. On the Basic tab:

      i. The **URL selector** is provided by the solution architect - set it to `http://SAH414A:7080/AvailabilityReceive`.

      ii. The **Maximum Client Wait time** is set to 60 seconds for testing.

   b. On the Default tab, set the **Message** parameters identically to Figure 9-69 on page 328.

   c. On the Validation tab , set **Validate** to `Content and Value`.

4. Rename the generated ESQL for Compute node, Validate, to `Flow4a_Validate`.

5. Configure the mapping transformation to return the acknowledgement to the assessor in the same way as "Flow3" on page 328.

> **Tip:** Remember we are mapping the AvailabilityResponse to the AvailabilityReponse return elements *referenced* from the soap11:envelope message, so it is the soap11:envelope message that needs to be mapped.

Flow3a

This message flow (Figure 9-82 on page 338) receives all valid messages from Flow4a and generates a Flow3a message to be passed to the ExternalClaimAssessors process. This is the fan-in part of the aggregation.

It also uses database table 'CLAIMASSESSOR' for the following reasons:

1. The incoming message does not contain the assessor's URL, but the Flow3a message needs the assessor's URL, so we retrieve it based on the ClaimID/AssessorID key.

2. The flow4a message from the assessor is not necessarily from WebSphere MQ, so cannot be assumed to contain in the correlID the original request's msgID. Aggregation assumes that the correlID does contain this value, so this flow retrieves the original msgID from the table and propagates it to the AggregateReply node as the CorrelId.

3. An audit trail of messages is kept in the CLAIMASSESSOR database.
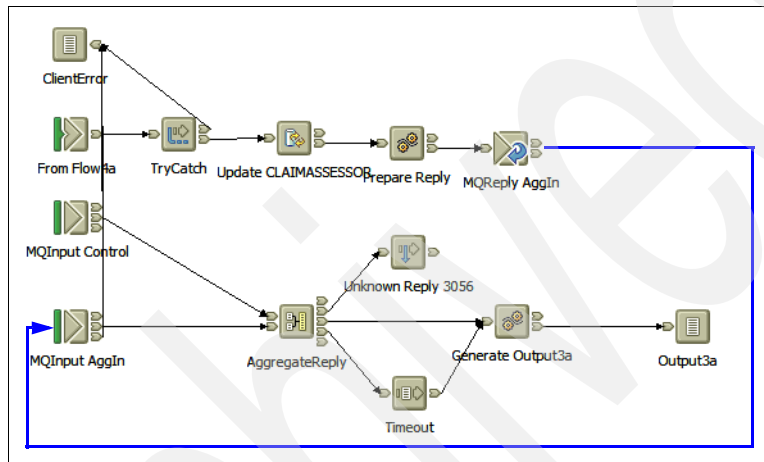
Figure 9-82   Flow 3a

1. The MQInput Control node receives the aggregation control message from Flow3.

   a. On the Basic tab, set the **Queue Name** as `FLOW3A.CONTROL`.

   b. On the Default tab, set the **Message Domain** to `XML`. No other properties are required on this tab.

   c. On the Advanced tab, set the **Transaction mode** to `No.` There are no other MQ messages to coordinate with.

   d. On the Validation tab, set **Validate** to `Content and Value.`

2. Create a TryCatch node to prevent any errors returning to the previous service flow.

3. Wire up the ClientError subflow to the Catch and Failure terminals of the MQInput nodes and to the catch terminal of the TryCatch node.

4. Create a Database update node, Update CLAIMASSESSOR. This node updates the CLAIMASSESSOR database with each assessors reply.

5. Create a Compute node, Prepare Reply, and generate a default ESQL module called Flow3a_Prepare_Reply. This is to be used to set up the WebSphere MQ message correctly for the AggregateReply node.

6. Create an MQ Reply node, MQReply AggIn, with AggIn as the queue name.

7. Create an MQ Input node, MQInput AggIn, with Aggin as the queue name. Set up the default properties to refer to the message set as before.

8. On the AggregateReply node set the **Aggregate Name** to the same as on the AggregateControl node: `proxyAssessorSystem`, Set the same timeout, 115 seconds. This is the timeout before unknown replies are discarded in case the replies arrive before the control message. Uncheck **Transaction Mode** as we are not limited to WebSphere MQ messages.

9. Create a compute node, Generate Output3a, and generate a default ESQL module called `Flow3a_Generate_Output3a` to create the Flow3a message to be sent.

10. Create a dummy Output3a flow, and wire it up.

11. Connect the Unknown and Timeout terminals of the AggregateReply node to Trace nodes configured like the trace nodes in the Fault flow, but with special error numbers to assist with debugging. In a real production flow some more manageable way of catching these conditions might be used.

12. Wire up the out terminal of the Timeout node to the Output3a flow because we *do* want to forward incomplete sets of replies.

Output3a

The Output3a flow (Figure 9-83) interfaces the proxyAssessorSystem to the LGI enterprise service bus using SOAP/Http to return the assessor availability list to the ExternalClaimAsessors process.
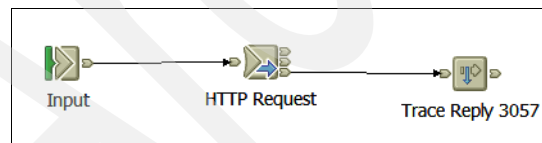
Figure 9-83   Output3a flow

1. Configure the properties of the HttpRequest node as follows:

   a. On the Basic tab:

      i. Set the **Web service URL** to the partner link for the ReceiveAssessorAvailabilityList in the ExternalClaimAssessor process: `http://SAH414B:9082/AssessorAvailabilityList.`

      ii. Set the **Request Timeout** to 60 seconds for testing.

iii. Select **Follow Http redirection** as the behavior for http redirection requests.

b. Leave the Advanced Tab and the Error Tab unchanged.
c. The Default Tab is identical to Figure 9-69 on page 328.
d. Set **Validation** to **Content and Value** on the Validation Tab.

2. For testing purposes. trace the reply to the event log as in Flow 4.

Output3aAck

No Output3aAck flow is needed for the SOAP/Http implementation of the LGI Enterprise service bus as the acknowledge flows back to the HttpRequest node in Output3a.

9.6.5 Create AssessorReport flows

Figure 9-84 shows the flows that have to be created for the AssessorReport interactions.
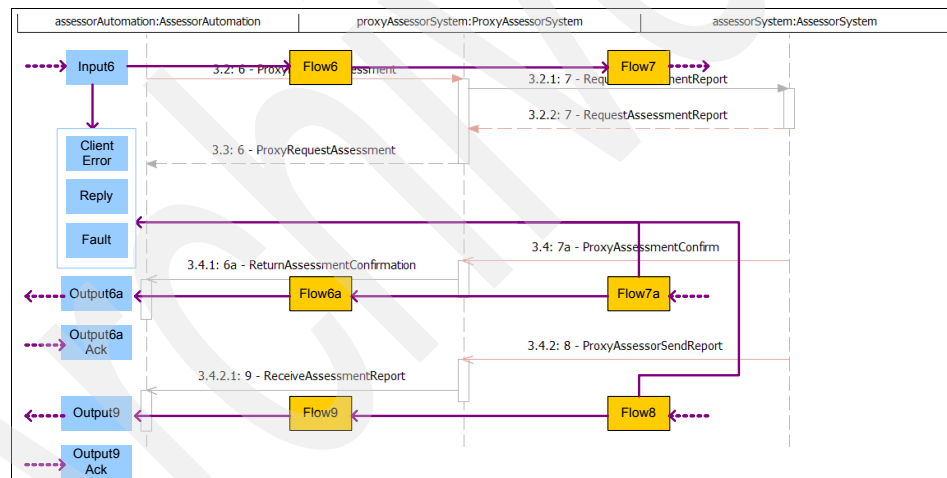
Figure 9-84 Assessor Report flows

Input6

The Input6 flow (Figure 9-85 on page 341) receives a request for an assessment report from the ExternalClaimAssessor process, passes control to Flow 6 for validation and processing, and deals with any errors that are thrown by returning a SOAP fault. It follows the same pattern as Input3.
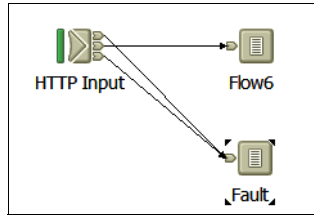
Figure 9-85   Input6 flow

1. Configure the Http Input node as follows,

    a. On the Basic tab, set the **URL Selector** to the value supplied by the
        solution architect in the AllocateAssessmentRequest(6).wsdl file -
        `http://SAH414A:9080/AllocateAssessmentRequest` and the timeout value
        to 60 seconds.
    b. On the Default tab, set the Message set properties as in Figure 9-69 on
        page 328.
    c. On the Validation tab, set **Validate** to `Content and Value`.

Flow6

Flow6 (Figure 9-86) validates the request received from flow Input6 before
sending the request for a claims assessment report being sent to an assessor.
Flow6 follows the same pattern as Flow3.
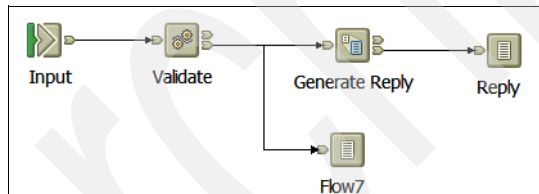
Figure 9-86   Flow6

Flow7

Flow7 (Figure 9-87 on page 342) sends the request for an assessment report to
an individual assessor. The ACTIONASSESSOR table keeps an audit record of
the requests which is updated in flow8 with the responses. The URL of the
assessor is set in the compute node, set SOAP address. The ESQL is described
in a later section.
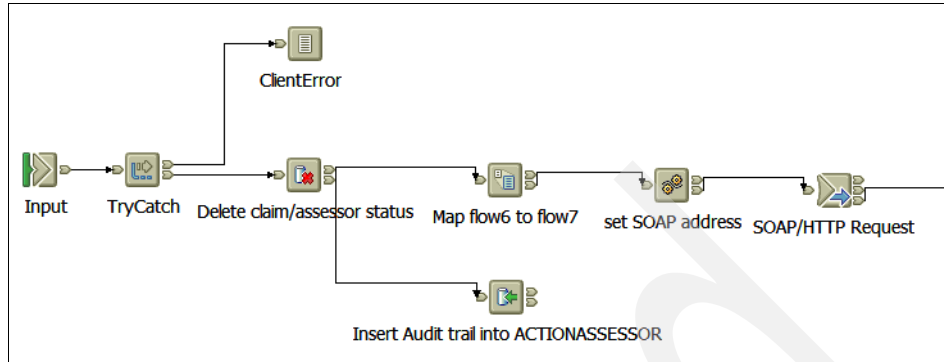
Figure 9-87　Flow7

1. Create the following nodes:

   a. A TryCatch node to trap any errors from returning to the calling flow, in this case flow6, and the associated ClientError flow to handle the errors

   b. A Database delete node to clear out any junk entries in the ACTIONASSESSOR table keyed by the combination of claimID/assessorID

   c. A Mapping node called Map flow6 to flow7

   d. A Database insert node called Insert Audit trail into ACTIONASSESSOR

   e. A compute node called set SOAP address

   f. A HTTP Request node called SOAP/Http Request

   g. A Trace node, Trace Reply 3058, to catch the reply

2. This time the Database Delete node tidies up any preceding instance of a particular claim being sent to a particular assessor.

   This instance of a Database delete node differs from the on in Flow4 "Configure Delete claim status DataDelete node" on page 336, in that there are two fields comprising the key and they need to be combined to select the correct set of rows.

   The procedure to do this is:

   a. Set up the mapping between the ActionAssessor(6) message and the ACTIONASSESSOR table as shown in Figure 9-88.

Figure 9-88　Mappings for deleting a claim for a particular assessor in flow6

b. Click on ASSESSORID and CLAIMID in the **Outline view** holding down the shift key to select both columns and right click the mouse button. See Figure 9-89.
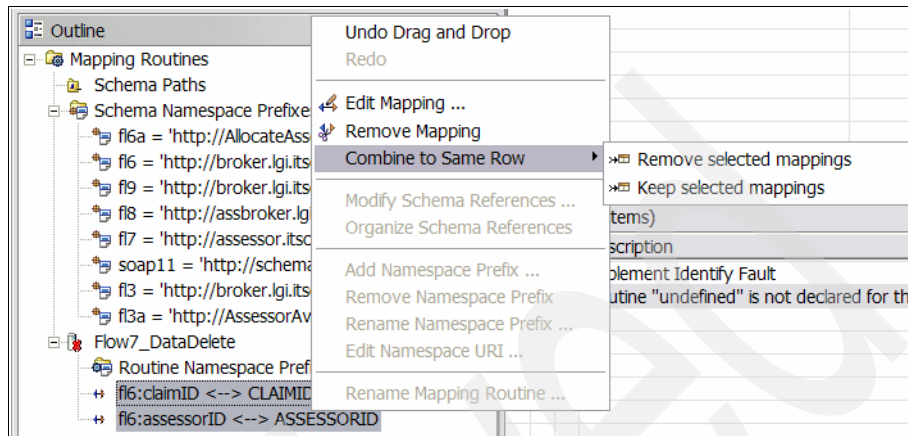
Figure 9-89   Selecting both fields for to construct a mapping expression

c. Select **Combine to same row** → **Remove selected mappings** and review the mapping condition (Figure 9-90) → **OK.**
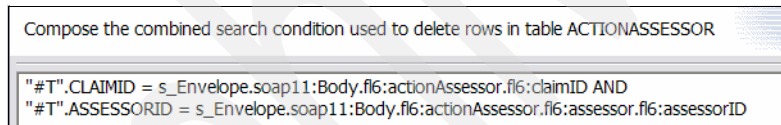
Figure 9-90   Review the mapping expression

d. You can review the Data Delete expression by selecting the combined mapping, or one of the fields and right-clicking **Edit Mapping** → **OK**.

See Figure 9-91 on page 344.
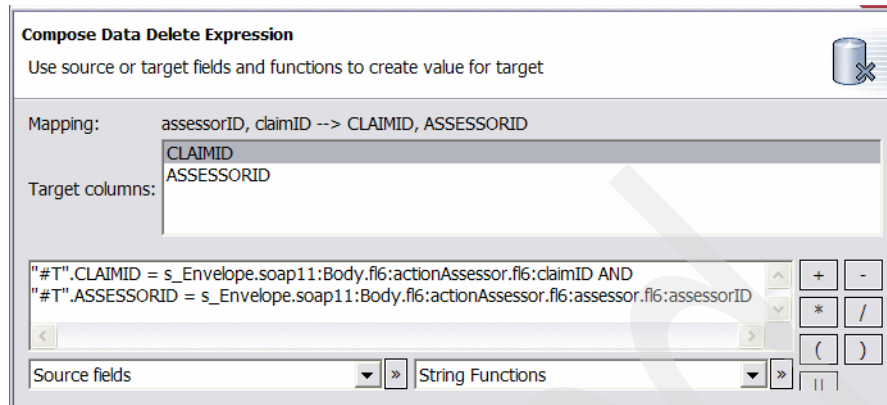
Figure 9-91   Editing a data delete expression

3. Generate default ESQL for the Compute node, "Set SOAP address", called Flow7_Set_SOAP_address.

4. Configure the properties of the HttpRequest node as follows,

   a. Basic Tab
      i. Set the **Web service URL** to `http://SAH414A:7080/UNKNOWN` as we did in Flow4.
      ii. Set the **Request Timeout** to 60 seconds for testing.
      iii. Select **Follow Http redirection** as the behavior for http redirection requests.
   b. Leave the *A*dvanced tab and the Error tab unchanged.
   c. The Default tab is shown in Figure 9-69 on page 328.
   d. Set **Validation** to **Content and Value** on the Validation tab.

5. The mappings from flow6 to flow7 are shown in Figure 9-92.

Figure 9-92   Mapping from flow6 to flow7

6. the database insert is illustrated in Figure 9-93 on page 345.

> **Important:** Remember to combine all the mappings into a single row, or else each mapping will be entered as a separate row in the table.
>
> Select all the mappings by using the mouse and shift key as before, and selecting **Combine into a single row → Remove selected mappings.**

Target	Source	Target value
ACTIONASSESSOR (EMERGE)	Envelope (soap11:Envelope)	
CLAIMID	fl7:claimID	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:claimID
ASSESSORID	fl7:assessorID	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:assessorID
LOCATION	fl7:location	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:location
REQDATE	fl7:requiredDate	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:requiredDate
MAKEOFCAR	fl7:makeOfCar	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:cardet.fl7:makeOfCar
REGISTRATION	fl7:registration	s_Envelope.soap11:Body.fl7:deliverAssessor.fl7:cardet.fl7:registration

Figure 9-93   inserting Assessment report request in ACTIONASSESSOR table

7. For testing purposes, trace the reply to the event log as in Flow4.

Flow7a

This message flow (Figure 9-94) receives a Flow7a message from the assessor chosen to produce the assessment report, validates it, sends an acknowledgment, and invokes flow6a to return the assessors acceptance or rejection to the ExternalClaimAssessors process. It's pattern is similar to Flow4a.
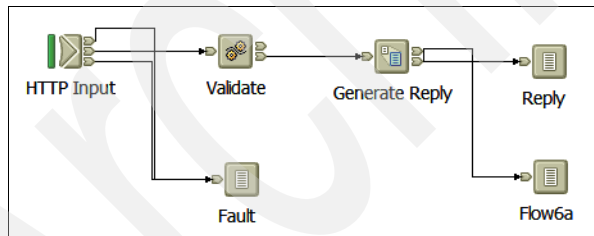
Figure 9-94   Flow7a

1. Create the following nodes:
   a. An HttpInput node to wait for the acceptance message.
   b. A compute node, Validate, to check the input message.
   c. A mapping node, Generate Reply, to map the response into the reply message.

2. Drop the Reply, Fault and Flow6a sub-flows into the flow.

3. Configure the HttpInput node:

   a. On the Basic tab:

i. The **URL selector** is provided by the solution architect - set it to `http://SAH414A:7080/DeliverAssessmentResponse.`

ii. The **Maximum Client Wait time** is set to `60` seconds for testing.

b. On the Default tab, set the **Message** parameters identically to Figure 9-69 on page 328.

c. On the Validation tab, set **Validate** to `Content and Value.`

4. Rename the generated ESQL for Compute node, "Validate" to `Flow7a_Validate.`

5. Configure the mapping for the acknowledgement as in Figure 9-95.

Target	Source	Target value
⊟─☑ Envelope (soap11:Envelope)	☑ Envelope (soap11:Envelope)	
├─🔳 soap11:Body		
├─☑ fl8:assessorResponseResponse		
├─🄴 fl8:assessorResponseReturn		
├─🄴 fl8:assessorID	🄴 fl8:assessorID	s_Envelope.soap11:Body.fl8:assessorResponse.fl8:assessorID
├─🄴 fl8:claimID	🄴 fl8:claimID	s_Envelope.soap11:Body.fl8:assessorResponse.fl8:claimID
└─🄴 fl8:timeStamp		CURRENT_TIMESTAMP

Figure 9-95   Mapping the acknowledgement message for flow7a

Flow6a

Flow6a (Figure 9-96) formats the assessor's acceptance response to be output by Output6a and stores an audit trail of the response in the ACTIONASSESSORS table.
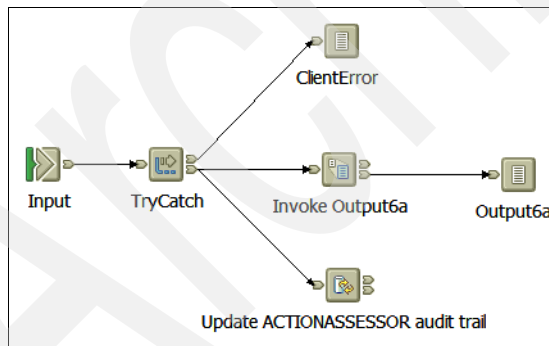
Figure 9-96   Flow 6a

1. Create the following nodes in Flow6a:

a. A TryCatch node and the ClientError subflow to isolate any faults from hereon in.

b. A Database update node to keep an audit trail.

c. A Mapping node to format the message for flow Output6a.

d. The Output6a subflow to return the results to the ExternalClaimAssessor process.

2. Create the update mappings as shown in Figure 9-97,

a. Select the message source and target ACTIONASSESSOR database table as before. D not forget to set the database schema to be used to EMERGE which is defined in the table source. See example in Figure 9-80 on page 336.

b. Map the two fields and set the one way CURRENT_TIMESTAMP mapping as shown in Figure 9-97 on page 347.

c. Combine the updates into a single row using the shift+mouse click protocol → **Remove selected rows**. You are then be presented with the Combine Data Update Mappings panel. You need to manually type in the Update when conditions (The SQL WHERE clause of course).

> **Tip:** One way of manufacturing this clause, and then pasting it in, is to create a Data Delete node and copying the delete conditions

Figure 9-97   Database Update configuration in Flow6a

3. Create the mappings for Invoke Output6a as shown in Figure 9-98 on page 347.

Figure 9-98   Mappings for flow6a

Output6a

The Output6a flow (Figure 9-99) interfaces the proxyAssessorSystem to the LGI enterprise service bus using SOAP/Http and passes the assessor availability list back to the ExternalClaimAssessor process.
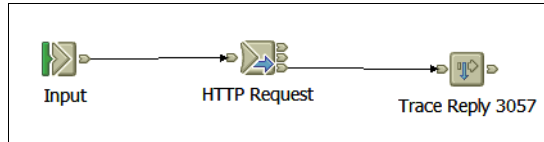
Figure 9-99   Output3a flow

1. Configure the properties of the HttpRequest node as follows:

   a. On the Basic tab:

      i. Set the **Web service URL** to the partner link for the ReceiveAssessorAvailabilityList in the ExternalClaimAssessor process: `http://SAH414B:9082/AssessorAvailabilityList.`

      ii. Set the **Request Timeout** to 60 seconds for testing.

      iii. Select **Follow Http redirection** as the behavior for http redirection requests.

   b. Leave the Advanced Tab and the Error Tab unchanged.
   c. The Default Tab is identical to Figure 9-69 on page 328.
   d. Set **Validation** to **Content and Value** on the Validation Tab.

2. For testing purposes trace the reply to the event log as in Flow 4.

Output6aAck

No Output6aAck flow is required because we are using SOAP/Http and the reply flows back to the Http Request node in the Output6a flow.

Flow8

Flow8 (Figure 9-100 on page 349) receives the assessor report from the chosen assessor and forwards it to Flow9 (Figure 9-102 on page 350). Its pattern is similar to Flow7a.
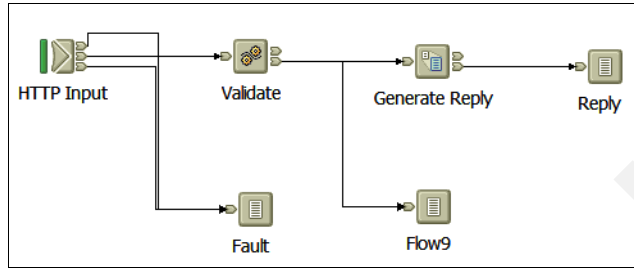
Figure 9-100   Flow8

1. Create the following nodes:
   a. An HttpInput node to wait for the acceptance message.
   b. A compute node, Validate, to check the input message.
   c. A mapping node, Generate Reply, to map the response into the reply message.

2. Drop the Reply, Fault and Flow9 sub-flows into the flow.

3. Configure the HttpInput node:

   a. Basic Tab:

      i. The **URL selector** is provided by the solution architect - set it to `http://SAH414A:7080/AssessorReport.`

      ii. The **Maximum Client Wait time** is set to `60` seconds for testing.

   b. On the Default tab, set the **Message** parameters as shown in Figure 9-69 on page 328.

   c. On the Validation tab, set **Validate** to `Content and Value.`

4. Configure the mapping node, "Generate Reply" as shown in Figure 9-101.

Figure 9-101   Configuring mapping for Flow8 acknowledgement

Flow9

Flow9 prepares the assessor report message to send to the receiving partner link in the ExternalClaimAssessor process. It passes the message to Output9 to put on the LGI service bus after updating its audit log
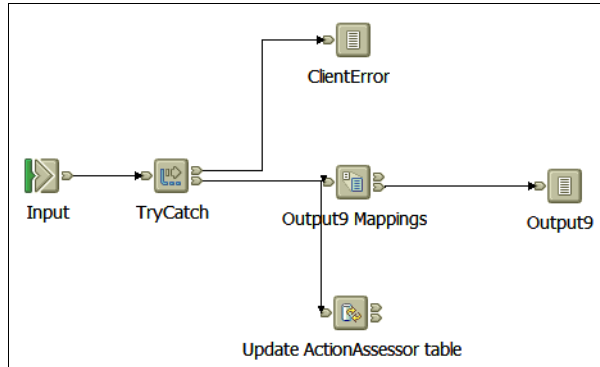
Figure 9-102   Flow9

1. Create the following nodes in Flow6a,

   a. A TryCatch node and the ClientError subflow to isolate any faults from hereon in.

   b. A Database update node, Update ActionAssessor table to keep an audit trail.

   c. A Mapping node to format the message for flow Output9.

   d. The Output9 subflow to return the results to the ExternalClaimAssessor process.

2. Create the database update mappings as shown in Figure 9-103 using a similar procedure to "Flow6a" on page 346.

Figure 9-103   Database update expression for flow9

3. Create the message mapping shown in Figure 9-104 on page 351.

Target	Source	Target value
☑ Envelope (soap11:Envelope)	☑ Envelope (soap11:Envelope)	
☑ soap11:Body		
☑ fl9:receiveAssessorReportRequest		
e claimID	e fl8:claimID	s_Envelope.soap11:Body.fl8:receiveAssessorReport.fl8:claimID
e assessorID	e fl8:assessorID	s_Envelope.soap11:Body.fl8:receiveAssessorReport.fl8:assessorID
e XMLReport	e fl8:XMLReport	s_Envelope.soap11:Body.fl8:receiveAssessorReport.fl8:XMLReport
e assCompDate	e fl8:assCompDate	s_Envelope.soap11:Body.fl8:receiveAssessorReport.fl8:assCompDate

Figure 9-104   Message mappings for Flow9

Output9

The Output9 flow (Figure 9-105) interfaces the proxyAssessorSystem to the LGI enterprise service bus using SOAP/Http and passes the assessor report back to the ExternalClaimAssessor process.
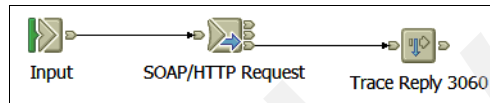
Figure 9-105   Output9 flow

1. Configure the properties of the HttpRequest node:

   a. On the Basic Tab:

      i. Set the **Web service URL** to the partner link for the ReceiveAssessorAvailabilityList in the ExternalClaimAssessor process: `http://SAH414B:9082/assessorReport.`

      ii. Set the **Request Timeout** to 60 seconds for testing.

      iii. Select **Follow Http redirection** as the behavior for http redirection requests.

   b. Leave the Advanced tab and the Error tab unchanged.
   c. The Default tab is identical to Figure 9-69 on page 328.
   d. Set **Validation** to **Content and Value** on the Validation tab.

2. For testing purposes trace the reply to the event log as in Flow 4.

Output9aAck

No Output9Ack flow is required because we are using SOAP/Http and the reply flows back to the Http Request node in the Output9 flow.

9.7 Create the ESQL code for the message flows

The final step in configuring the message flows in the broker is to write the ESQL code for the compute nodes. To browse all the ESQL modules we have created

to support the flows, open the properties of any compute node and click **Browse.** See Figure 9-106).
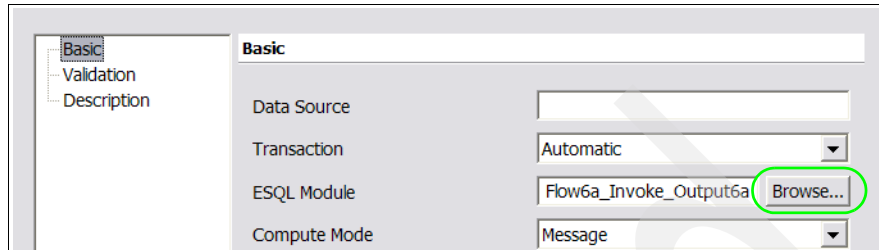
Figure 9-106   Browsing ESQL modules

Figure 9-107 shows the list of ESQL modules needed for the proxyAssessorSystem, plus, there is a declaration module, common.esql.
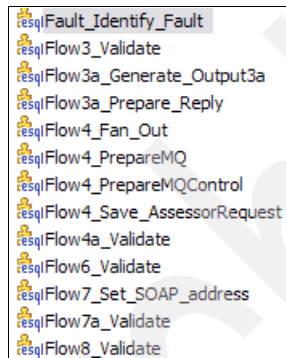
Figure 9-107   ESQL modules to be written

There are fourteen ESQL modules to be written. This can seem like quite a lot of code to create and maintain. But the code falls into only four categories:

1. Implementing the aggregation for SOAP/Http, something the broker does not support out of the box in version 5 (6 modules).

2. Fault Handling, the ESQL is an attempt to introduce a little more diagnostic information into the standard error output. (6 modules).

3. Constructing the SOAP address to send assessment request to the chosen assessor.

4. Improving the readability of the code by defining namespace prefixes in the common.esql module.

In contrast Figure 9-108 on page 353 lists the fifteen mapping nodes that handle most of the data mediation and database manipulation and do not require esql coding.
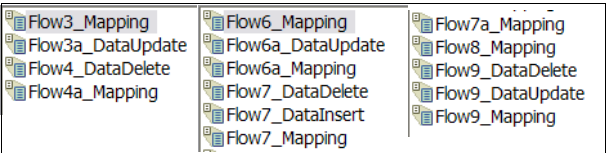
Figure 9-108   Mapping modules

9.7.1 ESQL functions to support Aggregation

Table 9-10 lists the seven ESQL modules needed for specific message flows.

Table 9-10   Message flow specific ESQL modules

ESQL Module	Description
Flow4_PrepareMQ	Convert the HTTP input message to WebSphere MQ
Flow4_Fan_Out	Construct the individual assessor availability request messages, create and save the Reply Identifier and set each assessors SOAP/Http address
Flow4_PrepareMQControl	Create the WebSphere MQ control message to send to the Aggregate Reply node
Flow4_Save_Assessor_Request	Save the request message to the assessor in the CLAIMSASSESSOR database.
Flow3a_Prepare_Reply	Correlate the assessors' availability with the Reply Identifier and insert into the LocalEnvironment folder
Flow3a_Generate_Output3a	Create aggregated availability response message to send to the ExternalClaimAssessors process
Flow7_Set_SOAP_Address	Set the assessors SOAP/Http address to send the assessment report request to.

Flow4_PrepareMQ

Example 9-9 on page 354 copies the input message into the output folder, removing the HTTP headers and replacing them by a new MQMD.

Example 9-9 `Flow4_PrepareMQ`

```
CREATE COMPUTE MODULE Flow4_PrepareMQ
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      SET OutputRoot = InputRoot;
      SET OutputRoot.HTTPInputHeader = null;
      SET OutputRoot.HTTPResponseHeader = null;
      CREATE NEXTSIBLING OF OutputRoot.Properties domain 'MQMD';
      SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID; -- create MQMD
      SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
      SET OutputRoot.MQMD.Format = '        ';
      SET OutputRoot.MQMD.MsgType = MQMT_DATAGRAM;
      RETURN TRUE;
   END;
```

Flow4_Fan_Out

Example 9-10 on page 355 is the ESQL for the Flow4_Fan_out module. There are eight parts to the code.

Nearly all the code (with the exception of the data in the SQL Insert statement and the LocalEnvironment references) is generated using Autocomplete - so the code is less difficult to write than it appears at first sight.

1. Declarations of local variables.

2. While loop to step through each assessor in the list of assessors.

3. Copying the InputLocalEnvironment to the OutputLocalEnvironment.

4. Setting the SOAP/Http destination in the LocalEnvironment for the HTTPRequest node to use dynamically.

5. Copying the data portion of the assessor request message from the input message to the output message.

6. Propagating each assessor message and its LocalEnvironment down the message flow.

Example 9-10   COMPUTE MODULE Flow4_Fan_Out

```
CREATE COMPUTE MODULE Flow4_Fan_Out
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      DECLARE numberOfAssessors INTEGER CARDINALITY (InputRoot.MRM.soap11:Body.
            fl3:requestAssessorAvailability.fl3:assessorList.fl3:assessors[]);
      DECLARE assessorCount INTEGER 0;
      WHILE assessorCount < numberOfAssessors DO
-- These statements must be in the loop because propagate clears OutputRoot
         CALL CopyMessageHeaders();
         SET OutputLocalEnvironment = InputLocalEnvironment;
         SET OutputRoot.MQMD.MsgType = MQMT_REQUEST;
         SET OutputRoot.MQMD.ReplyToQ = 'AggIn';
         SET OutputRoot.MQMD.Report = MQRO_COPY_MSG_ID_TO_CORREL_ID;
         SET assessorCount = assessorCount + 1;
-- Set the SOAP/Http address for the RequestHttp node to use as a destinations
         SET OutputLocalEnvironment.Destination.HTTP.RequestURL = InputRoot.MRM.soap11:Body.
fl3:requestAssessorAvailability.fl3:assessorList.fl3:assessors[assessorCount].fl3:assessorURL;
-- Copy data portion of message from list (f17 is a synonym for f14 - same namespace)
-- Output fields must be in order as the element is a sequence
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:claimID =
         InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:claimID;
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:assessorID =
                  InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:assessorList.
                  fl3:assessors[assessorCount].fl3:assessorID;
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.
                  fl7:makeOfCar =
                     InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:makeOfCar;
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.
                  fl7:registration = 'JB 007'; -- Fixup as the registration wasn't provided
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:location =
         InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:location;
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:reqDate =
         InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:requiredDate;
         SET OutputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:responseTime =
         InputRoot.MRM.soap11:Body.fl3:requestAssessorAvailability.fl3:responseTime;
         PROPAGATE;
      END WHILE;
-- All messages are explicitly propagated, so don't propagate a null message
      RETURN FALSE;
   END;
```

Flow4_Control_Message

The Flow4_Control_Message compute node propagates the message output on the Control terminal of the Aggregate control node to the MQOutput node that puts it on the queue destined for the AggregateReply node.

All we need to do for this module is create an WebSphere MQ message and pass it the message propagated from the Aggregate Control node as an unstructured XML message.

Example 9-11   COMPUTE MODULE Flow4_PrepareMQControl

```
CREATE COMPUTE MODULE Flow4_PrepareMQControl
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID;
        SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
        SET OutputRoot.XML = InputRoot.XML;
        RETURN TRUE;
    END;
END MODULE;
```

Flow4_Save_AssessorRequest

We save the assessor request into the CLAIMASSESSOR database after generating the request message so that we have the actual written value of the MsgID stored by the AggregateRequest node. Note we saved the Assessor URL in the Local Environment.

Example 9-12   Flow4_Save_AssessorRequest

```
CREATE COMPUTE MODULE Flow4_Save_AssessorRequest
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyEntireMessage();
        SET OutputLocalEnvironment = InputLocalEnvironment;
-- Save message in the CLAIMASSESSOR table
        INSERT INTO Database.EMERGE.CLAIMASSESSOR (claimID, assessorID, assessorURL,location,
                    reqdate, makeofcar, registration, replytoq, replytoqmgr, correlid) VALUES (
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:claimID,
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:assessorID,
        InputLocalEnvironment.Destination.HTTP.RequestURL,
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:location,
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:reqDate,
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.fl7:makeOfCar,
        InputRoot.MRM.soap11:Body.fl7:requestAssessorAvailability.fl7:cardet.fl7:registration,
        'NoReplytoQ',
        'NoReplytoQmgr',
-- Storing the correlation token from the generated message id
        InputLocalEnvironment.WrittenDestination.MQ.DestinationData.msgId);
        RETURN TRUE;
    END;
```

Flow3a_Prepare_Reply

The Flow3a_Prepare_Reply node receives availability responses from the assessors. Its function is to extract the Reply Identifier for the Aggregate Reply node and create an WebSphere MQ request message which will be sent to the MQ Reply node to return a real WebSphere MQ reply message with the correct correlation information for the AggregateReply node. Example 9-13 is the SQL to do that.

The ESQL clears out any HTTP information, creates an MQMD, and configures it as a request message with the original MsgId.

Example 9-13   COMPUTE MODULE Flow3a_Prepare_Reply

```
CREATE COMPUTE MODULE Flow3a_Prepare_Reply
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      SET OutputRoot = InputRoot;
      SET OutputRoot.HTTPInputHeader = null;
      SET OutputRoot.HTTPResponseHeader = null;
-- Create MQ request message - it will be converted to a reply message by MQReply
      CREATE NEXTSIBLING OF OutputRoot.Properties domain 'MQMD';
      SET OutputRoot.MQMD.StrucId = MQMD_STRUC_ID; -- create MQMD
      SET OutputRoot.MQMD.Version = MQMD_CURRENT_VERSION;
      SET OutputRoot.MQMD.Format = '         ';
      SET OutputRoot.MQMD.Report =  MQRO_COPY_MSG_ID_TO_CORREL_ID;
      SET OutputRoot.MQMD.MsgType = MQMT_REPLY;
-- Same msgid as request message id which was stored in the ClaimAssessor table.  MQReply will
copy this to the correlid
      SET OutputRoot.MQMD.MsgId =
      THE (SELECT ITEM A.correlid FROM Database.EMERGE.CLAIMASSESSOR AS A
      WHERE A.assessorID = InputRoot.MRM.soap11:Body.fl8:assessorAvailability.fl8:assessorID
      AND A.claimID = InputRoot.MRM.soap11:Body.fl8:assessorAvailability.fl8:claimID);
      SET OutputRoot.MQMD.ReplyToQ = 'AggIn';
      RETURN TRUE;
```

Flow3a_Generate_Output3a

The Flow3a_Generate_Output3a module builds the Flow3A result message including the list of assessors to return to the ExternalClaimAssessor process.

The composite message is stored in the ComIbmAggregateReplyBody.Flow4 array. For some reason, the MessageSet must be reset in the Properties. The value is the same as we have been setting in all the input folders.

Example 9-14   COMPUTE MODULE Flow3a_Generate_Output3a

```
CREATE COMPUTE MODULE Flow3a_Generate_Output3a
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
-- How many replies have we got? The Flow4 folder is defined in the Aggregate Request node?
        DECLARE noofreplies INTEGER CARDINALITY(InputRoot.ComIbmAggregateReplyBody.Flow4[]);
        DECLARE replyno INTEGER 1;
        DECLARE assessorID INTEGER;
        SET OutputRoot.Properties = InputRoot.ComIbmAggregateReplyBody.Flow4.Properties;
        SET OutputRoot.Properties.MessageSet = 'PIJOMIKOO2OO1';
        IF noofreplies > O THEN
            SET OutputRoot.MRM.soap11:Body.fl3a:AvailableAssessorsList.claimID =

InputRoot.ComIbmAggregateReplyBody.Flow4[replyno].MRM.soap11:Body.fl8:assessorAvailability.fl8:claimID;
            WHILE noofreplies >= replyno DO
                SET assessorID = InputRoot.ComIbmAggregateReplyBody.Flow4[replyno].
                    MRM.soap11:Body.fl8:assessorAvailability.fl8:assessorID;
                SET
OutputRoot.MRM.soap11:Body.fl3a:AvailableAssessorsList.resultAssessorCollection[replyno].
                    assessorEstimations.assessorID = assessorID;
-- The assessorURL is not in the reply message, so we have to get it from the database
                SET
OutputRoot.MRM.soap11:Body.fl3a:AvailableAssessorsList.resultAssessorCollection[replyno].
                    assessorEstimations.assessorURL = THE (SELECT ITEM A.assessorURL FROM
                    Database.EMERGE.CLAIMASSESSOR AS A WHERE A.assessorID = assessorID);
                SET
OutputRoot.MRM.soap11:Body.fl3a:AvailableAssessorsList.resultAssessorCollection[replyno].
                    assessorEstimations.preCost = InputRoot.ComIbmAggregateReplyBody.Flow4[replyno].
                    MRM.soap11:Body.fl8:assessorAvailability.fl8:predCost;
                SET
OutputRoot.MRM.soap11:Body.fl3a:AvailableAssessorsList.resultAssessorCollection[replyno].
                    assessorEstimations.preDate = InputRoot.ComIbmAggregateReplyBody.Flow4[replyno].
                    MRM.soap11:Body.fl8:assessorAvailability.fl8:predDate;
                SET replyno = replyno + 1;
            END WHILE;
            RETURN TRUE;
        ELSE
-- Don't propagate a message if there are no replies
            RETURN FALSE;
        END IF;
    END;
```

9.7.2 Setting the SOAP/Http destination dynamically

Example 9-15 on page 359, Flow7_Set_SOAP_address, sets the SOAP/Http
destination to be used by the HTTPRequest node to send the request for an
assessment to the chosen assessor. The assessorURL is in the Flow6 input
message, but it is not copied over to the Flow7 message sent to the assessor, so
the ESQL gets it from the Flow6 input message.

Example 9-15   COMPUTE MODULE Flow7_Set_SOAP_address

```
CREATE COMPUTE MODULE Flow7_Set_SOAP_address
    CREATE FUNCTION Main() RETURNS BOOLEAN
    BEGIN
        CALL CopyEntireMessage();
        -- We need to set the URL of the AllocateAssessmentRequest. Only the
        URL of the AssessorAvailabilityRequest is defined
        -- The quick fixup is to have a naming convention...
        SET OutputLocalEnvironment.Destination.HTTP.RequestURL =
            REPLACE(InputRoot.MRM.soap11:Body.fl6:actionAssessor.fl6:assessor.
                    fl6:assessorURL,'Availability','DeliverAssessment');
        RETURN TRUE;
    END;
```

9.7.3 Common namespace prefix declarations

Table 9-11 lists the namespace module.

Table 9-11   Common namespace module

ESQL Module	Description
common	Namespace and prefix declarations

It is good practice to provide shortened namespace prefixes for the namespaces used in the ESQL code. It shortens statements and makes the code more readable. The namespaces will be found in the Assessor Messageset we created earlier. If there are missing or duplicate prefixes they can be added or changed now. The same prefixes should be given to the same namespaces. We were careful to edit the schema files to all contain different prefixes before they were converted into message sets. The results can be seen in Figure 9-109 on page 360.

In the **Resource Navigator** → **Assessor Messageset** → **Assessor** → Click **messageSet.mset** and click the **XML1** tab in the message set editor.
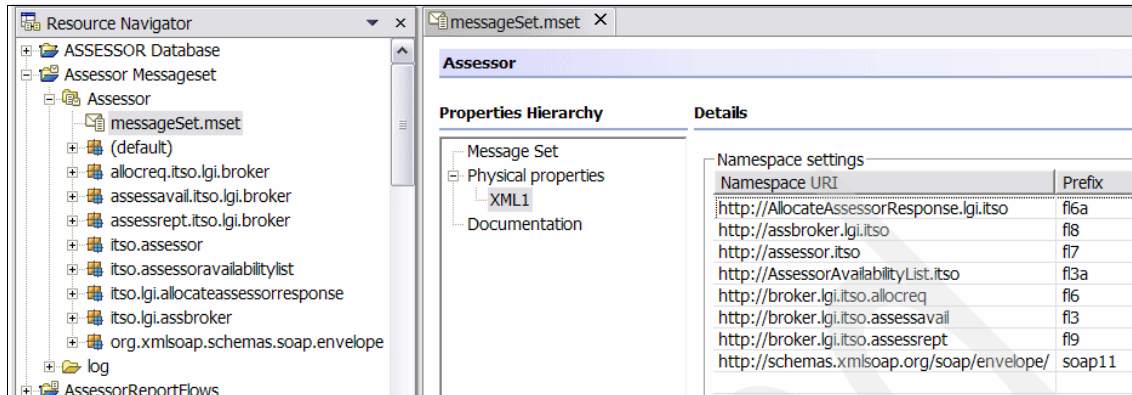
Figure 9-109 Adding prefix to namespace declaration

> **Note:** Where there were schemas with the same target namespace the
> duplicate namespaces and prefixes have been discarded by the message
> broker.

Add the namespaces and prefixes from each of the message sets to the
common.eqsl module as shown in Example 9-16. Note where we have duplicate
namespaces we can't declare duplicate prefixes.

Example 9-16 Declaring namespaces in common.esql

```
BROKER SCHEMA   proxyAssessorSystem
DECLARE soap11 NAMESPACE 'http://schemas.xmlsoap.org/soap/envelope/';
DECLARE fl3    NAMESPACE 'http://broker.lgi.itso.assessavail';
DECLARE fl3a   NAMESPACE 'http://AssessorAvailabilityList.itso';
DECLARE fl7    NAMESPACE 'http://assessor.itso';
DECLARE fl8    NAMESPACE 'http://assbroker.lgi.itso';
DECLARE fl6    NAMESPACE 'http://broker.lgi.itso.allocreq';
DECLARE fl6a   NAMESPACE 'http://AllocateAssessorResponse.lgi.itso';
-- DECLARE fl4a NAMESPACE 'http://assbroker.lgi.itso';
-- DECLARE fl7a NAMESPACE 'http://assbroker.lgi.itso';
-- DECLARE f4   NAMESPACE 'http://assbroker.itso';
-DECLARE fl9    NAMESPACE 'http://broker.lgi.itso.assessrept';
```

> **Tip:** When changing namespace prefixes, or editing the common.esql file,
> warning messages might appear inexplicably in the task list about unresolved
> message field references. If rebuilding all the workspace does not clear the
> warnings, closing the workspace, reopening it and rebuilding it entirely again
> will clear incorrect warnings.

9.7.4 ESQL Error handling code

Table 9-12 lists the ESQL validate modules to be written and the common fault handler. The validation modules differ only very slightly from one another.

Table 9-12   Validation ESQL modules

ESQL Module	Description
Flow3_Validate	
Flow4a_Validate	
Flow6_Validate	"flow specific" validation of input message
Flow7a_Validate	
Flow8_Validate	
fault_identify_fault	Common SOAP fault handler

Flow3_Validate shows a particular example of one of the routines.

Flow3_Validate

The validation routines all follow the same general pattern,

1. Set up the parameters to call the ValidateMessage routine in the Broker's global data folder "Environment":

   - SOAP message named expected in Environment.Message.

   - Flag to indicate that the fault exception message is set in this module rather than by the generic SOAP fault handler.

   - References to the folders passed to ValidateMessage.

2. Call ValidateMessage and check the return flag.

   - Propagate the Inputroot to Outputroot if validation succeeds.

   - Throw an exception with specific fault data that will be caught by the input node at the start of the flow and passed to the generic fault handler if the validation fails.

     We use a generic SOAP fault message to store the exception as not all the WSDLs we are working with have a Fault Interface. The fault should only get returned if the Web service has a fault interface.

Example 9-17 on page 362 shows the code for validating flow3.

Example 9-17 Flow3_Validate ESQL

```
CREATE COMPUTE MODULE Flow3_Validate
   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      DECLARE xInputRoot REFERENCE TO InputRoot;
      SET Environment.Message = 'requestAssessorAvailability';
-- The following statement registers the fact that this web service generates its
-- own exception message rather than the default SOAP Fault
      SET Environment.SOAP.Fault.FaultOption = 'CustomizedFault';
      DECLARE xEnvironment REFERENCE TO Environment;
-- Validate the message
      CALL ValidateMessage(xInputRoot, xEnvironment);
      IF Environment.SOAP.Fault.FaultCode = ' ' THEN
         SET OutputRoot = InputRoot;
      ELSE
-- This web service generates its own fault message....  here it is and the exception path is
followed
         SET Environment.soap11:Body.Fault.faultstring =
         'Flow3 input message validation failed...  claimID ' ||
         CAST(InputBody.soap11:Body.fl3:requestAssessorAvailability.fl3:claimID AS CHARACTER);
         THROW USER EXCEPTION VALUES ('Flow3 Input validation failed');
      END IF;
      RETURN TRUE;
   END;
```

Remainder of the validation modules

Copy the code from Flow3_Validate, and make the changes shown in Table 9-13 to each of the copies.

The parts of the code marked in **bold red** need to be changed for each Validate compute node ESQL implementation.

Table 9-13 Variable inserts into Validate routines

Flow	Message	ClaimID field
Flow3	requestAssessorAvailability	fl3:requestAssessorAvailability.fl3:claimID
Flow4a	assessorAvailability	fl4a:assessorAvailability.fl4a:claimID
Flow6	actionAssessor	fl6:actionAssessor.fl6:claimID
Flow7a	assessorResponse	fl7a:assessorResponse.fl7a.claimID
Flow8	receiveAssessorReport	fl8:receiveAssessorReport.fl8:claimID

Fault_Identify_Fault

The purpose of the Fault_Identify_Fault module (Example 9-18) is to format a compliant SOAP fault packet containing useful diagnostic data. The Main module is simply an escape if the fault packet has already been created. The output is sent to the HttpReply node for returning to the SOAP client, otherwise FaultProc is called to create the fault packet.

Example 9-18   COMPUTE MODULE Fault_Identify_Fault - Main Module

```
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
   IF Environment.SOAP.Fault.FaultCode = 'FaultReceived' THEN
      -- We have received a fault from another web service....  copy details to output
      SET OutputRoot = InputRoot;
   ELSE
      CALL FaultProc();
   END IF;
   RETURN TRUE;
END;
```

The faultproc routine in Example 9-19 on page 364 has eight sections and tailors the detail of the fault message depending on what information is available about the fault.

Example 9-19   COMPUTE MODULE Fault_Identify_Fault - Faultproc

```
CREATE PROCEDURE FaultProc()
BEGIN
-- 1. The web service requires a generic SOAP fault message
    CALL CopyMessageHeaders();
    SET OutputRoot.HTTPInputHeader = null;
    SET OutputRoot.HTTPResponseHeader = null;
-- 2. supply suitable values for unknown fault in Service Configuration Data
    IF Environment.SOAP.Fault.FaultCode IS NULL OR
        Environment.SOAP.Fault.FaultCode = ' '
        THEN
        SET Environment.SOAP.Fault.FaultActor ='proxyAssessorSystem';
        SET Environment.SOAP.Fault.FaultCode = 'Server';
        SET Environment.SOAP.Fault.FaultString = 'Server error in SOAP Web service';
    END IF;
-- 3. Build an output SOAP fault message (MRM)
    SET OutputRoot.Properties.MessageSet = 'Assessor';
    SET OutputRoot.Properties.MessageType = 'Envelope';
    SET OutputRoot.Properties.MessageFormat = 'XML1';
-- 4. Output will be MRM Message, Add standard SOAP Envelope
-- Explicitly add the namespace to fault code value
-- Put the original message body in the fault (if we can.... )
-- 5. The web service requires a customized fault message
    IF Environment.SOAP.Fault.FaultOption = 'CustomizedFault' THEN
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultcode = 'soap11'||':'||
            Environment.SOAP.Fault.FaultCode;
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultstring = Environment.SOAP.Fault.FaultString;
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultactor = Environment.SOAP.Fault.FaultActor;
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.detail = Environment.soap11:Body;
    ELSE
-- 6. The web service requires a default SOAP fault message
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultcode = 'soap11'||':'||
            Environment.SOAP.Fault.FaultCode;
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultstring = Environment.SOAP.Fault.FaultString;
        SET OutputRoot.MRM.soap11:Body.soap11:Fault.faultactor = Environment.SOAP.Fault.FaultActor;
        IF InputExceptionList.ParserException.ParserException.ParserException.Number = 5117 THEN
            SET OutputRoot.MRM.soap11:Body.soap11:Fault.detail.OriginalBody =
                'Input message is not valid XML';
        ELSE
            SET OutputRoot.MRM.soap11:Body.soap11:Fault.detail.OriginalBody = InputRoot.*[<];
        END IF;
-- 7. Check if the exception list is the result of an ESQL THROW...
        IF InputExceptionList.RecoverableException.RecoverableException.UserException.Number = 2951 THEN
-- 8. and if not, output the exception list in the SOAP fault message
        ELSE
            SET OutputRoot.MRM.soap11:Body.soap11:Fault.detail.ExceptionList = InputExceptionList;
        END IF;
    END IF;
END;
```

9.8 Deploy message set and flows

Our next task would be to deploy the message sets and flows ready to test them, but first we need to implement the URL: http://SAH414A:7080/UNKNOWN which was defined in flows 4 and 7 as the default assessor URL should the flow fail to set an assessor URL dynamically. This is one of these *Should never happen* situations, and so we will define a trivial broker flow to handle it.

9.8.1 Create the UNKNOWN flow

Within the CommonSOAPHttpFlows message flow project, create a new message flow called `UnknownAssessor`. See Figure 9-110.
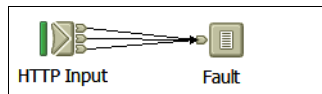
Figure 9-110   Unknown flow

Drop an HttpInput node and the fault subflow into the flow, and define the HttpInput node properties as follows,

1. Basic Tab:
    a. Set the URL Selector to `http://SAH414A:7080/UNKNOWN`.
    b. Set the client wait time to 30 seconds.
2. Default Tab:
    a. Message domain: MRM.
    b. Message Set: Assessor.
    c. Message Type: Envelope.
    d. Message Format: XML1.
3. Validation Tab:
    a. Validate: Content and Value.
    b. Failure Action: Exception.

9.8.2 Create a Broker Archive

To deploy the flows and message set we need to define a Broker Archive (.bar file) and specify which flows and message sets comprise it. For simplicity we define a single .bar file containing everything we need and deploy it to a single execution group on a single broker. If, for example, we wanted to deploy the Assessor Availability flows separately from the Assess Report flows we would define two broker archives.

1. Switch to the Broker Administration perspective.
2. Right-click **Broker Archives** → **New** → **Message Broker Archive** and give the new file the name `Assessor` → **Finish.**

3. In the .bar editor (the **Content** panel) Click the green **Add** icon and select the flows and message set as shown in Figure 9-111. → **Finish**.

4. Check the "Details" in the dialog for errors. → **OK.**
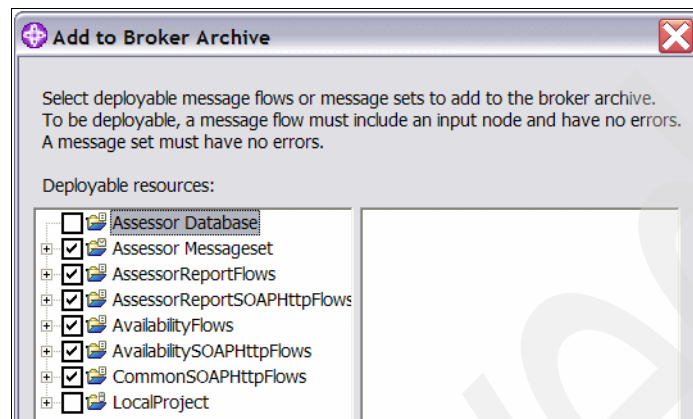
Figure 9-111   Creating the Assessor.bar file

The resulting archive looks like Figure 9-112. Note only flows with Input nodes are listed.

Figure 9-112   Assessor.bar file

5. Save the Assessor.bar file.

Tip: You can open the .bar file with a .zip tool and inspect the contents. One of the useful tricks that a message broker expert will tell you about is if you are having problems with SQL generated by the Database nodes in the message flows, then inspect the final SQL in the .bar file rather than the .xmi format in the development perspective of the workbench. You can see what is actually being deployed. You might also find the SQL easier to understand because some symbolic values have been resolved. See Example 9-20 on page 367.

Example 9-20   Sample SQL from .bar file

```
CREATE PROCEDURE Flow3a_DataUpdate (IN s_Envelope REFERENCE
{&apos;http://schemas.xmlsoap.org/soap/envelope/&apos;}:Envelope)
BEGIN
--$IBM_WBIMB_XMIID=UpdateStatement_1
UPDATE Database.EMERGE.CLAIMASSESSOR AS &quot;T#&quot;;
--$IBM_WBIMB_XMIID=UpdateStatement_1#assignments
SET PREDCOST = s_Envelope.soap11:Body.fl8:assessorAvailability.fl8:predCost,
PREDDATE = s_Envelope.soap11:Body.fl8:assessorAvailability.fl8:predDate
WHERE &quot;T#&quot;.CLAIMID =
s_Envelope.soap11:Body.fl8:assessorAvailability.fl8:claimID AND
&quot;T#&quot;.ASSESSORID =
s_Envelope.soap11:Body.fl8:assessorAvailability.fl8:assessorID ;
END;
```

9.8.3 Deploying Assessor.bar to the broker

If you have not already done so, you need to start the server running the message broker. See 7.2.5 "Install and configure the Message Broker" on page 230, and reconnect to the message broker from the toolkit. See especially "Toolkit configuration" on page 234.

1. In the Broker Administration perspective open the **Domain** view and right click **WBRK_BROKER** → **New** → **Execution Group** and call it `Assessor` → **Finish.** See Figure 9-113.
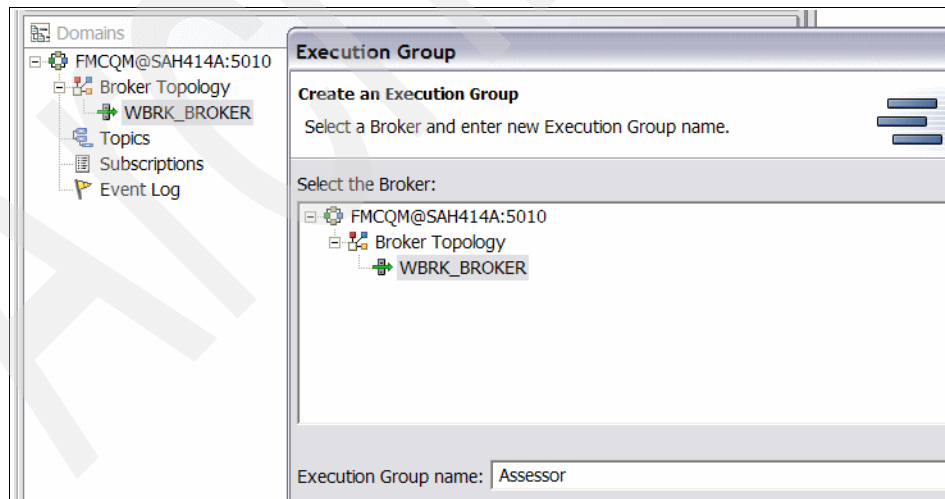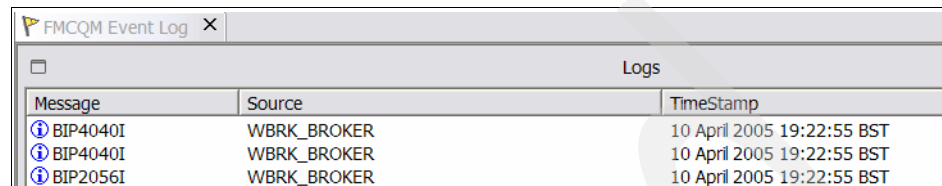
Figure 9-113   Creating the Assessor execution group

2. Drag and drop the Assessor.bar file onto the new execution group. Check the details in the Pop-up box → **OK.**

3. Double-click the Event log in the Domains view. Check the FMCQM event log that opens (Figure 9-114). Clicking the messages shows their contents. There should be no errors if you have followed the instructions accurately.

Figure 9-114   Checking event log

4. Refresh the execution group in the Domains view. The result is shown in Figure 9-115.
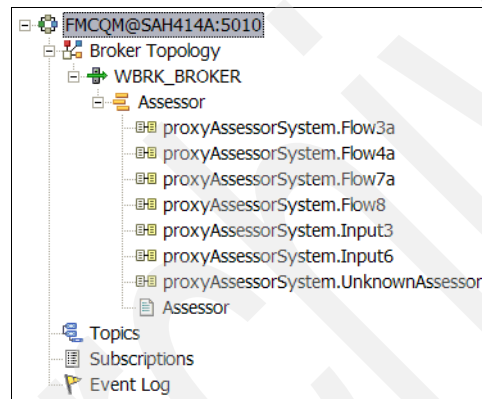
Figure 9-115   Assessor execution group after deployment

9.9 Unit testing the deployed flows

Testing the deployed flows requires:

1. A Test tool to submit and receive sample SOAP messages

2. A scaffolded assessor and claim system, or use of the sample assessor application running on WebSphere Application Server

3. Knowledge of how to trace and debug WebSphere Business Integration Message Broker flows

9.9.1 Test tools

There are a number of SOAP client test tools you could use. The most accessible and versatile we have found is the Web services Explorer packaged with WebSphere Studio Application Development Integration Edition and Rational Software Architect. This tool lets you load a WSDL file, and modify portions of it, such as a Web service address. It has both a Form view and a Trace view to see the contents of request and reply messages. Both these workbenches also have a TCP monitor tool if you want to see what is actually being transmitted.
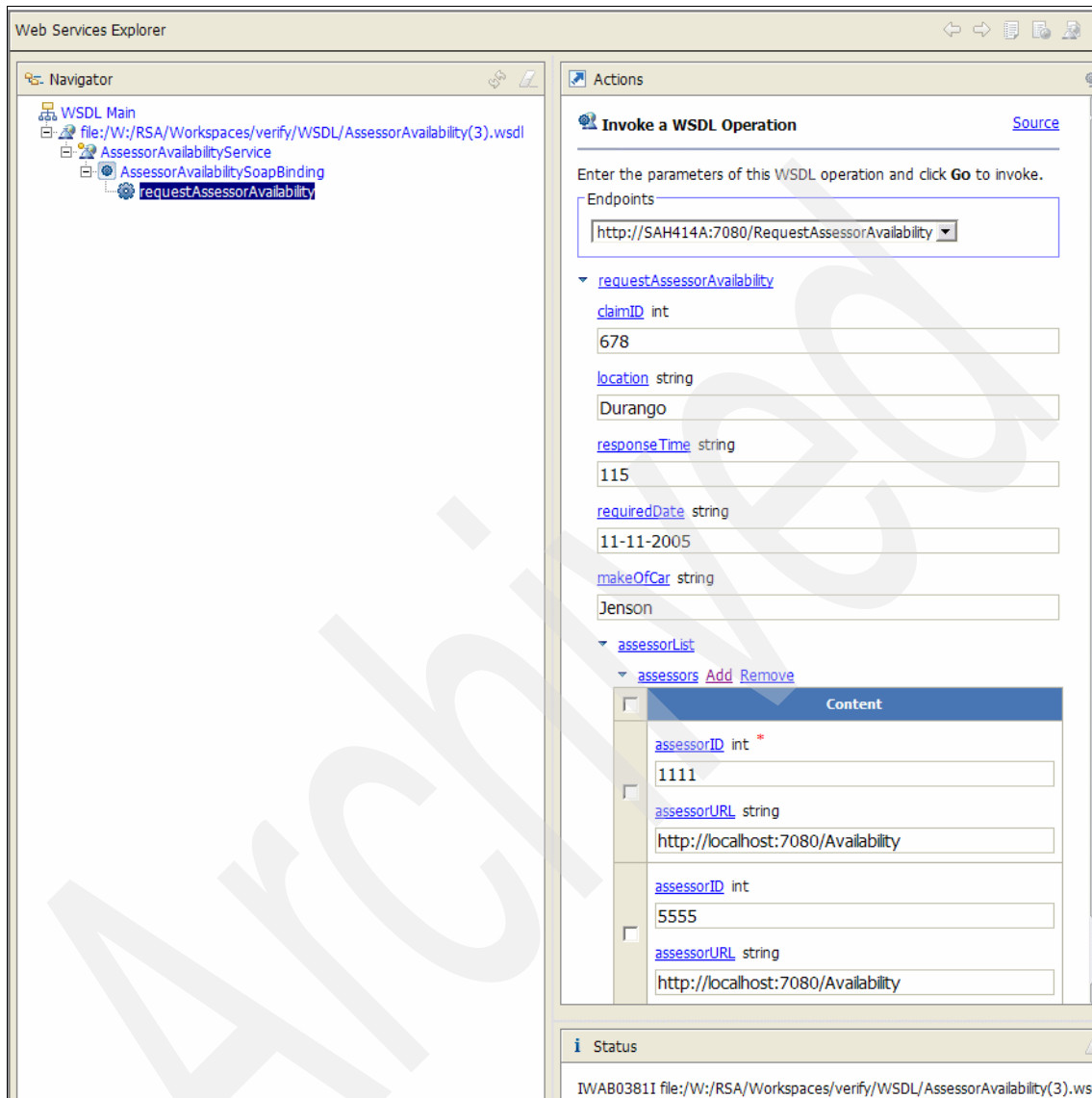
Figure 9-116   Using Web services explorer to test WebSphere Business Integration Message Broker

9.9.2 Scaffolded Assessor and Claim system

It is very easy to scaffold an Assessor and a Claim system to test the proxyAssessor system. One of the difficulties with the Assessor system is it behaves asynchronously, returning two messages in response to the selected assessor request. It is difficult to program definite or indefinite delays into EJBs.

An alternative is to create an Assessor message flow in the message broker, and simulate the delay by putting WebSphere MQ messages onto queues which are Get Inhibited, and releasing the messages only when required.

Figure 9-117 shows the basic assessor availability scaffolding. The flow sends an HTTP response back, and then sends the assessment availability request back. Mapping nodes make it easy to assemble the right contents for the messages. Trace nodes help to debug and verify the solution.
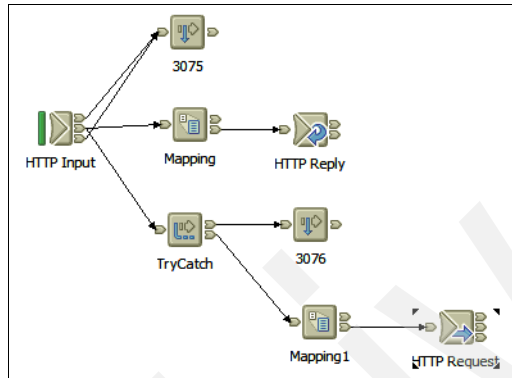
Figure 9-117   Main Assessor Availability scaffolded flow

Figure 9-118 shows the main Assessor Report scaffolding. The structure is pretty much the same, but this time the response is followed by two flows to send the acknowledgement and then the report itself. The requests are put onto WebSphere MQ queues. Two other flows take the requests off the queues as soon as the Get Inhibit flag is reset using the MQ Explorer tool, and then generate HTTP requests back to the proxyAssessorSystem.
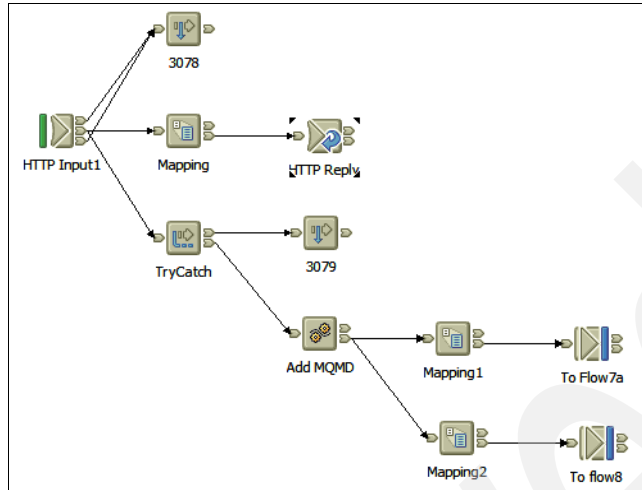
Figure 9-118   Main Assessor report scaffolded flow

9.9.3 Tracing and debugging flows

There are three main techniques of debugging WebSphere Business Integration Message Broker flows: using trace, using trace nodes and using the workbench debugger.

One of the best features of WebSphere Business Integration Message Broker is its debugging capability. The tools give visibility to all the data that is flowing, node to node, and depending on your preferences, you can use the trace facility to get a complete printout of the behavior of the flows, use trace nodes to snapshot only part of the activity, or use the debugger to step though sections of the flow or ESQL and interactively modify its execution.

Tracing

Tracing is controlled from the command line and works at the granularity of an execution group. You can change the granularity of tracing to individual flows by setting options in the workbench domains view.

These five scripts are handy to make new trace runs quickly.

ClearAtrace (Example 9-21 on page 373) and GetAtrace (Example 9-22 on page 373) handle the interfaces to the broker. They leave the level of tracing as set, for example using the interface in the workbench. ClearTraces and GetTraces (Example 9-23 and Example 9-24) contain the lists of execution groups being studied, and Trace (Example 9-25 on page 373) pulls it altogether in one command which you run each time you want to start tracing.

Example 9-21   ClearAtrace: Clears the trace log for an execution group

```
@rem clears the trace log
@SETLOCAL
@rem first argument is the broker name and the second, execution group name
@mqsichangetrace %1 -u -e %2 -r
@echo Tracing options for execution group %2 running on broker %1
@mqsireporttrace %1 -u -e %2
@ENDLOCAL
```

Example 9-22   GetAtrace: Retrieves the formatted trace for an execution group

```
@rem Retrieves the trace log
@SETLOCAL
@rem first argument is the broker name and the second, execution group name
@mqsireadlog %1 -u -e %2 -o %2.xml
@mqsiformatlog -i %2.xml -o %2.txt
@start notepad %2.txt
@ENDLOCAL
```

Example 9-23   ClearTraces: Clear all the execution groups you are interested in

```
@SETLOCAL
@Call ClearAtrace WBRK_BROKER LGIAvailability
@Call ClearAtrace WBRK_BROKER LGIReport
@Call ClearAtrace WBRK_BROKER Assessor
@ENDLOCAL
```

Example 9-24   GetTraces Retrieve all the traces

```
@SETLOCAL
@Call GetAtrace WBRK_BROKER LGIAvailability
@Call GetAtrace WBRK_BROKER LGIReport
@Call GetAtrace WBRK_BROKER Assessor
@Call ClearTraces
@ENDLOCAL
```

Example 9-25   Trace: Retrieve all the traces and clear them

```
@SETLOCAL
@Call GetTraces
@Call ClearTraces
@ENDLOCAL
```

Trace Nodes

Trace nodes ▸ ⬚ ▸ can be inserted anywhere in a flow, or wired in parallel with another output connector.
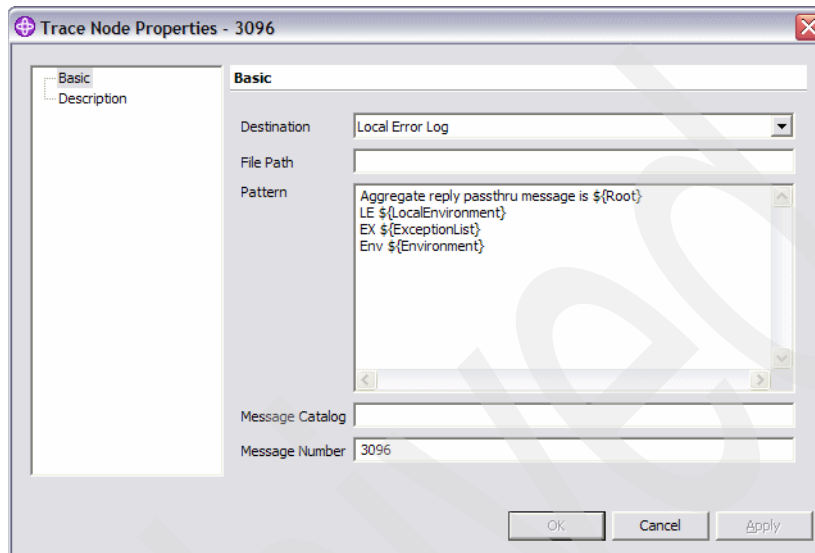
Figure 9-119   Typical trace node

Figure 9-119 is a typical trace node. It is configured to output an event to the Windows event log, which will appear as an Error 3096 and will contain the contents of the message tree, the local environment, the exception list and the Environment. It doesn't matter if any of these are null at run time. Use SQL expressions to tailor the output, but for debugging, dumping the lot is good enough.

The Message number is picked from the predefined catalog from the reserved range, 3051-3099 which have a predefined message. Additional catalogues and messages may be defined, but for debugging the predefined message catalog is good enough.

The usual procedure is to bring up the Windows Management Console to view the Application event log. Clear the log before a tracing run, and then if by selecting have chosen message numbers carefully the progress of the run is easily tracked as trace events appear in the log.

Debugging

The last technique to describe is traditional step by step debugging.

Open the flow debug perspective, which appears as a red bug . The dialog box takes you through attaching to a broker, then to one or more execution groups, and setting break points. Note that the button to press to step through ESQL is different from the button to press to step through a message flow, and easily missed. See Figure 9-120.
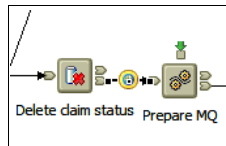
Figure 9-120   Breakpoint in message flow with option to trace into ESQL

The variables windows displays the value of variables when you are stepping through ESQL in a compute node. If there is no ESQL in your flow, then the debugger is not going to be so useful. The variables windows has a special folder called Debug Message that shows all the data available in the flow. See Figure 9-121.
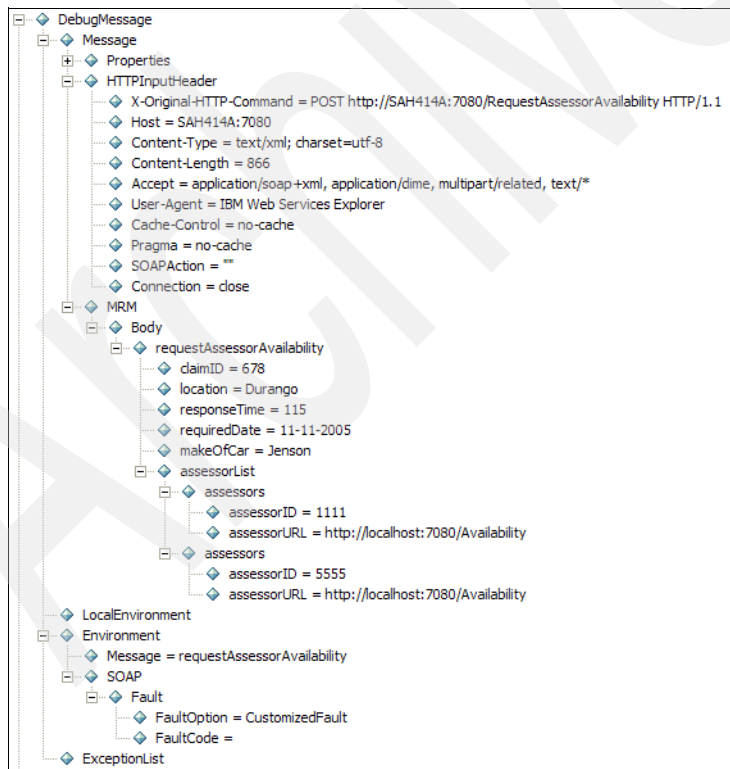
Figure 9-121   Debug message

10

Build the Request External Reports process

In this chapter, we describe how to develop and deploy a business process which is exported from WebSphere Business Integration Modeler in the format of Business Process Execution Language for Web service (BPEL4WS). We build our business process using the output of WebSphere Business Integration Modeler so that we can take over from the business analyst. Because the business process exported from Modeler still lacks implementable services and settings information, needed in running the business process in a real system, we have to modify BPEL so that the business process engine can execute the business process.

This chapter includes following sections:

10.1 Overview

The IT process specialist uses WebSphere Studio Application Developer Integration Edition V5.1.1 to deploy the business process defined by the business analyst in WebSphere Business Integration Modeler on WebSphere Business Integration Server Foundation V5.1. Figure 10-1 shows the central role the IT process specialist plays in implementing the solution.

BPEL is exported from WebSphere Business Integration Modeler by the IT process specialist rather than by the business analyst. Technical errors in the BPEL have to be fixed before the process can be exported and the IT specialist has the right skills to fix the errors. It is also an opportunity for the IT specialist to get an overview the process in the Modeler. The export of BPEL is described in 4.5.3, "Export RequestExternalReports as a BPEL4WS process" on page 143.

The BPEL process uses partner links to interact with most of the external services. All the partner links required by the IT process specialist have been defined in .WSDL files by the solution architect using Rational Software Architect. These have been packaged in a single .zip file to be easily managed. The .zip file needs to be imported into a package in WebSphere Studio Application Development Integration Edition.

Any additional WSDL interfaces that are required can be defined in WebSphere Studio Application Development Integration Edition or your favorite WSDL editor.
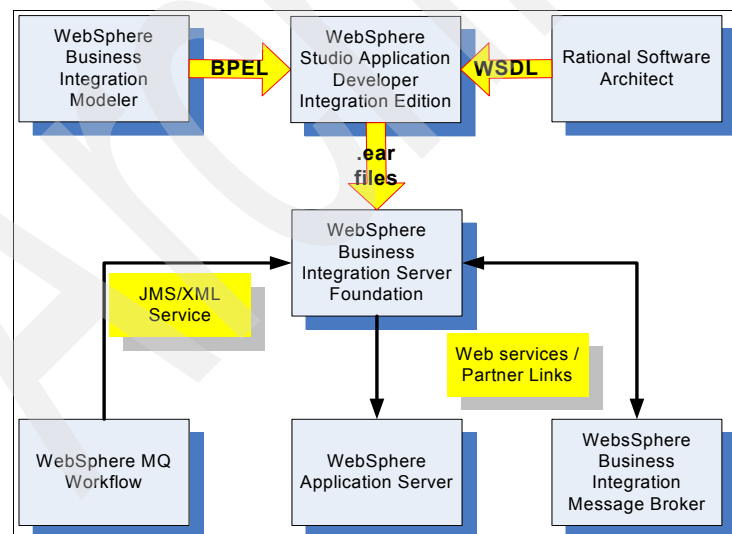
Figure 10-1   Building the business process flow

There are three kinds of services the IT process specialist integrates:

1. There are the straightforward Web services invoked by the business process. They happen all to be implemented as EJBs and reside on a separate WebSphere Application Server.

   These are described in Chapter 8, "Test and deploy the application components" on page 243.

2. There are services connected to the ESB, implemented by WebSphere Business Integration Message Broker. The requests and replies are implemented as separate services, with the Process Choreographer acting as a service to receive the responses from the ESB.

   These are described in Chapter 9, "Build the Enterprise Service Bus" on page 261.

3. The other connection that has to be built is between WebSphere MQ Workflow and WebSphere Business Integration Server Foundation. The Workflow engine has a special integration mechanism that makes use of a proxy EJB running on the same process engine as the business process. Invoking the RequestExternalReports process is then a simple partner link in which our process is invoked by this EJB on behalf of the Workflow process. This connection is the job of the Workflow IT specialist.

   This is described in Chapter 11, "Modify the Claim Investigation process" on page 453.

When the IT process specialist has completed the definition of the BPEL process, it can be tested, either by importing the application components into the WebSphere Studio Application Development Integration Edition test environment or by linking the test versions of the services running on their native platforms. After the process is debugged and tested, then it is deployed to WebSphere Business Integration Server Foundation.

10.2 Import WSDL and BPEL into the IDE

The WSDL used in the External Claim Assessor solution has been defined by the solution architect and is part of the PSM (Product Specific Model) contract between the solution architect and the IT specialists. The BPEL has been defined by the business analyst and is part of the CIM (Computer Independent Model) between the business analyst and the IT roles. This section describes how the IT specialist transfers the BPEL and WSDL files into WebSphere Studio Application Development Integration Edition.

10.2.1 Import WSDL from Rational Software Architect

The WSDL definitions will be used to create the partner links, operations and variables in the BPEL model. The solution architect exported the final WSDL definitions in a .zip file (see "Making materials available" on page 215.) We will now import the .zip file into WebSphere Studio Application Development Integration Edition and create the partner links we need.

1. Launch WebSphere Studio Application Development Integration Edition and open **Business Integration** perspective. Select **File** → **New** → **Service Project**. The New Project wizard opens. See Figure 10-2.
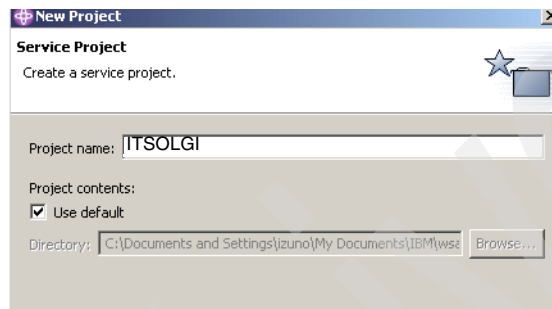
Figure 10-2   Create a new service project

2. Type the unique name for the new project (ITSOLGI) and click **Finish**. A new service project will be created under Service Projects folder in Services view.

3. Before copying the WSDL file into the Service Project which includes our BPEL process, create a new Java package to organize the imported WSDL files.

   Right-click the Service Project **ITSOLGI** and select **New** → **Package** to add new package. Type services as the name of the package → **OK**. You can find the new package added under the Service project. See Figure 10-3 on page 381.

   Tip: If you choose to load or reference WSDLs in a different project, then you need to alter the properties of the LGIITSO service project to reference it.
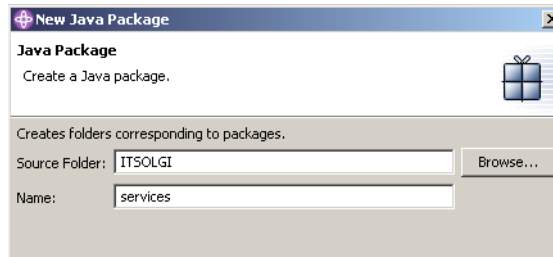
Figure 10-3   Create new Java package

4. Import the relevant WSDL files.

   In **Package explorer**, right-click **Import** → Zip file → browse to find the WSDL files packaged by the solution architect (or use .\SG24-6636\RSA\ClaimsInvestigation WSDL files.wsdl → Select the WSDLs which are used by the Assessor Automation service. See Table 10-1 on page 391 → **Finish.**

Important tip: The WSDL files should be imported directly into the Package folder, and not into a subfolder. By default, the folder tree is copied from Rational Software Architect and the WSDLs are in a subdirectory. This does not cause any problems building the BPEL process as long as the folder names are valid package names. However when you deploy the process, the WSDLs are not copied from a subdirectory into the deployment .ear file. It is a good practice to move the WSDLs directly into the package folder from the start.

If you put your WSDLs in the wrong subdirectory, how do you move them?

► If you discover the mistake before you have started developing and building the process, simply move the WSDLs using the navigator.

► If you have started building the process, or even have got to the deployment step before discovering the problem, then moving the WSDLs will refactor many of the references to the WSDLs, but it is not 100% complete. After refactoring, use the search button and search for the subpath where the WSDLs were previously located and fix up the references. It is easiest to open the affected files using a simple text editor. One of the files that will not be refactored is <processname>.bpelex. There will be one reference to change for each partner link in this file. When typing in the new path to the WSDL files, *ensure your path starts with a forward slash* (/). Not using the slash causes the build process to abend.

10.2.2 Import BPEL from WebSphere Business Integration Modeler

The first question the IT specialist must consider, before we describe the mechanics of importing BPEL from WebSphere Business Integration Modeler into WebSphere Studio Application Development Integration Edition, is whether to

► Continue to refine the CIM BPEL process in the Modeler and then transfer it to WebSphere Studio Application Development Integration Edition.

► Take the model as-is from the analyst when it is ready for export.

The IT specialist can choose to edit the Request External Reports business process using either tool.

Choosing when to import BPEL into the IDE

Modeler shows the business process using Business Process Modeling Notation (BPMN)[1]. In contrast, WebSphere Studio Application Development Integration Edition has three alternative ways of editing the business process suited to different styles of process modeling. The Modeler style is chosen to suit the business analyst and to be consistent across different modeling tools. Because the style is not tied to a particular process technology,it is easier to compare business processes modeled with different tools. It is also easier to see the logic of the overall flow. For complex flows, it might be worthwhile continuing to refine the logic of the flow in Modeler before transferring to WebSphere Studio Application Development Integration Edition.

The editing styles in WebSphere Studio Application Development Integration Edition are designed to make mapping the flow to the target technology as easy as possible. In particular two of the styles, flow-based BPEL process and sequence-based BPEL process, are optimized to edit BPEL processes. Which to choose is a matter of your personal taste. At some point it, will be necessary to edit the BPEL flow using WebSphere Studio Application Development Integration Edition to fine tune the how the flow is executed.

The question are:

► At what point should the IT specialist switch from using the Modeler to using WebSphere Studio Application Development Integration Edition?

► Taking into account the strengths of the tools, when should we export BPEL from Modeler?

► How far we should edit BPEL in Modeler?

---

[1] See Stephen White, "Introduction to BPMN", found at
http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf

In the case of the External Claim Assessor scenario, the choice is straightforward. That choice is based on the decision we made that the solution architect is responsible for interfaces and data modeling. The business analyst only defines the business process activity model and not its interfaces. The solution architect uses Rational Software Architect to model the interfaces. The architect has chosen to use Rational Software Architect rather than WebSphere Business Integration Modeler because of the requirement to incorporate interfaces directly into the UML architecture model. The process specialist needs to work in WebSphere Studio Application Development Integration Edition from the start because that person needs to integrate the BPEL process with its WSDL interfaces. Modeler does not have the capability to import WSDL interface definitions.

In other model-driven development projects, the choices are different. The data modeling might be done in WebSphere Business Integration Modeler, or in some specific data-modeling tool. As discussed in 3.2.8, "Tool chains" on page 70, before commencing a project in earnest it is really important to work out who is responsible for what tasks, which tools they use, and how they integrate the artifacts from the different tools.

In our case we were circumscribed in our choices because WebSphere Business Integration Modeler 5.1.1 does not have the capability to import WSDLs. But in a project where the WSDLs are created in the Modeler along with the BPEL there is more scope for the IT specialist to continue with a technical refinement of the process model in WebSphere Business Integration Modeler before exporting it to WebSphere Studio Application Development Integration Edition.

The the External Claim Assessor scenario our the best option was to export the simple BPEL process definition from WebSphere Business Integration Modeler and to ask the IT specialists to use the WSDL definitions from Rational Software Architect to complete the process definition and implementation as described in the rest of this chapter.

Export BPEL from Modeler

The business analyst's business process model should be saved as part of the CIM contract between the business and IT roles in the project.

The IT specialist is then responsible for taking the model through the rest of its refinement.

Inspection of the BPEL validity in WBI Modeler

When WebSphere Business Integration Modeler is switched to BPEL mode any warnings or errors in the BPEL are displayed. All the errors have to be fixed before the BPEL can be exported; the warnings are not a problem. "Validate

process model" on page 143 explains how to correct the BPEL errors in the RequestExternalReport process.

When the process model is error free, it can be exported and its definitions imported into WebSphere Studio Application Development Integration Edition. There is a choice of catalogs to import:

► Data catalog,
► Process catalog,
► Resource catalog,
► Organization catalog or its contents, or
► A single process, single business item, a resource or an organization unit.

If a complete process is exported, then all of the business items which the process and its subprocesses use, and all of the individual resources and organization units which are responsible for execution of the process will be exported together. Business items, resource definitions, organization definitions and location definitions are exported in XML schema (XSD) file format.

 "Export process" on page 150, explains how to export the RequestClaimsAssessor process from WebSphere Business Integration Modeler and what options to select.

Import BPEL into the IDE
To import the BPEL into the IDE, follow these steps:

1. Right-click the ITSOLGI service project → **Import** to open the import wizard → **File system** → **Next**.

2. Specify the source folder where you exported the BPEL to, from Modeler. To work from the materials supplied with the red book import the files from the .\SG24-6636\Modeler directory shown in Figure 10-4. Make sure that the services project you created in 10.2.1, "Import WSDL from Rational Software Architect" on page 380 is selected in **Into folder** section. The business process and business items are imported into a workspace. Make sure you select the root information model as well as the BPEL and WSDL files for import.

   See Figure 10-4 on page 385.

10.3 Integrate the process with its services
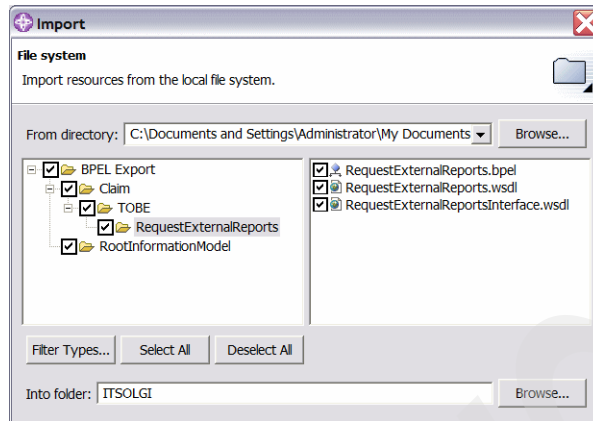
Figure 10-4   Imported process files

3. Open the business process editor by double-clicking **Service Projects** →
   (ITSOLGI) → **Claim.TOBE.RequestExternalReports** →
   RequestExternalReports.bpel in the Services view.

There might be errors. Do not worry about them for now. We fix these errors in
the following pages.

In the center of the editor, you can see the business process. This process
inherits the process definition from Modeler. The process in Modeler flows from
right to left, but the process in WebSphere Studio Application Development
Integration Edition flows from the top down. On the right part of the process
(Figure 10-5 on page 386) editor, you can see the partner links that are the
interface of Web services invoked from the process activities. Those services are
now abstract. Modeler generates the WSDL files, but those service definitions
have no implementation. In our case, we are going to replace the definitions
entirely, because they are defined by the solution architect in Rational Software
Architect.
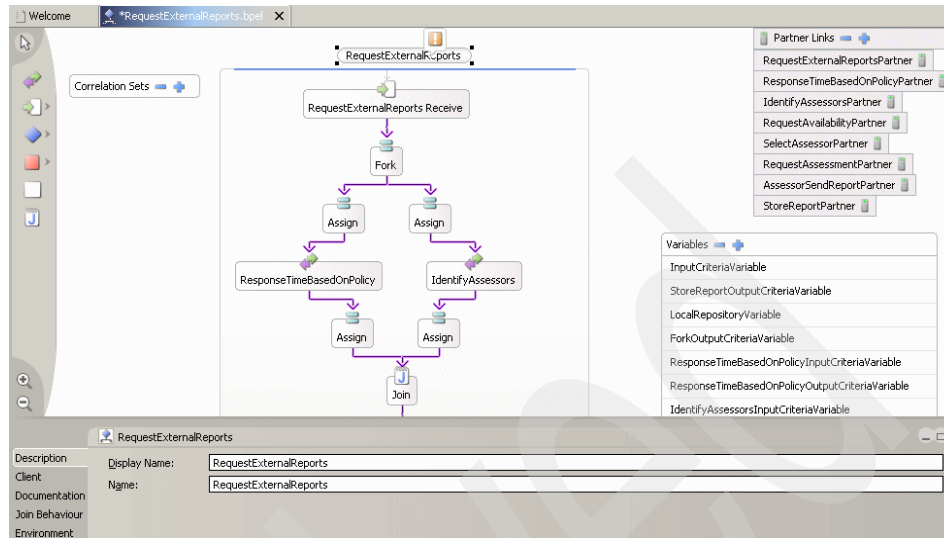
Figure 10-5   BPEL editor

In the left part of the editor there is palette to edit the process. Add new activities to the process by selecting an icon from palette and dropping it onto the editor. Specific information about each element in the process is displayed in the detail area in the bottom of the editor. The contents of this area change, depending on which process element you select.

The Modeler process editor and the WebSphere Studio Application Development Integration Edition process editor are displayed side-by-side in Figure 10-6. Modeler is on the left and WebSphere Studio Application Development Integration Edition on the right.
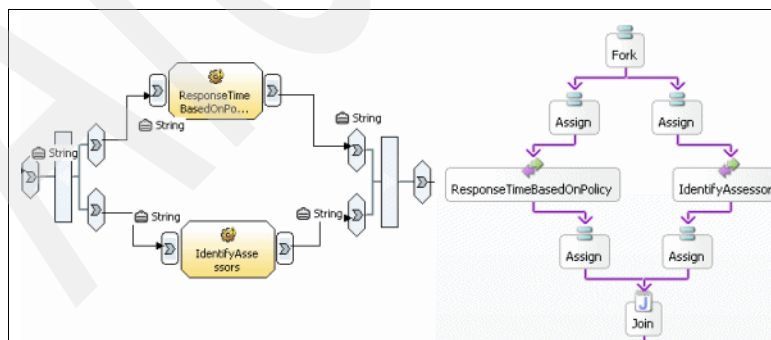
Figure 10-6   Imported process

Each element of Modeler and the definition of BPEL and WSDL correspond as follows:

- ► Local tasks in Modeler become *Invoke activities*.

- ► If a staff resource is assigned to local task, it will be staff activity in WebSphere Studio Application Development Integration Edition.

  – The manual task named ManualChooseAssesor becomes a staff activity.

- ► Forks and merges become *Assign activities* or *Empty activities*.

- ► *Assign activities* are inserted between each activity. Assign activities map the output data structure of the preceding activity to the input data of the following activity.

10.4 Integrate the process with its services

In this section we take the BPEL flow imported from the WebSphere Business Integration Modeler and transform it into an executable BPEL flow. There are some pieces of Java to be written, but mainly the procedure is driven by configuration wizards or performed with the graphical process editor. There are nine steps required to transform the BPEL process into executable form.

1. Section 10.4.1, "Correct the list of partner links in the model" on page 388 replaces the abstract partner links in the business process model provided by the business analyst with the partner links that call the real services specified by the solution architect.

2. Section 10.4.2, "Integrate the partner links with the process" on page 390 connects the new partner links with the corresponding activities in the flow, and, adds additional activities to the flow to extend the BPEL model to match the solution architecture.

3. Section 10.4.3, "Configure the partner links" on page 394 checks that each partner link is correctly configured according to its role in the activity (called or calling) and that the correct Port type (or interface) is used in the link.

4. Section 10.4.4, "Configure the activities" on page 396 checks that global input and output variables are assigned to each activity and that the operations required by each activity are correctly selected.

5. Section 10.4.5, "Configure the types of input and output variables" on page 398 checks that for each input and output variable it is correctly typed by the its corresponding input or output message used in each activity.

6. Section 10.4.6, "Map data between input and output variables" on page 400 maps the data to and from each partner link making any necessary adjustments to the data to match the requirements of the partner link services.

7. Section 10.4.7, "Configuring the flow to wait for responses from the assessors" on page 416 adds a correlation set to the flow to enable it to wait for asynchronous messages from the assessors and configures the three activities that wait for an assessor response to use the correlation set.

8. Section 10.5.1, "Checking the results from RequestAvailability" on page 418 tests the results of requesting the availability of the assessors using a Java Snippet and configures two conditional links to invoke either a manual or automatic activity to select the assessor to be asked to perform the assessment.

9. Section 10.5.2, "Create a While activity to test for a committed assessment" on page 423 adds a while activity to check that the selected assessor does agree to perform the assessment.

10.4.1 Correct the list of partner links in the model

If you count the partner links on the right-hand side of the editor generated from the RequestExternalReports model imported from WebSphere Business Integration Modeler and compare the result with the number of WSDL interfaces required by the RequestExternalReports collaboration imported from Rational Software Architect, you can see there is a mismatch. This mismatch is due to the business process model not detailing the additional interactions that are required to implement the asynchronous flows.

Table 10-1 on page 391 shows the WSDL files for the interfaces where the RequestExternalReports process plays either a client or server role, and the corresponding partner links. Entries in blue are present in the BPEL model imported from WebSphere Business Integration Modeler and entries in red are the missing ones, which will need to be added, along with some means to route the requests from the partner links back into the process flow. The black rows are interactions that do not involve the Automated Assessor process component.

The following procedure corrects the list of partner links.

1. Remove the partner links imported with the BPEL process from the Automated Assessors model:

   With the requestExternalReports.bpel file open in the editor window go to the outline view and select all the partner links and delete them (Figure 10-7 on page 389).
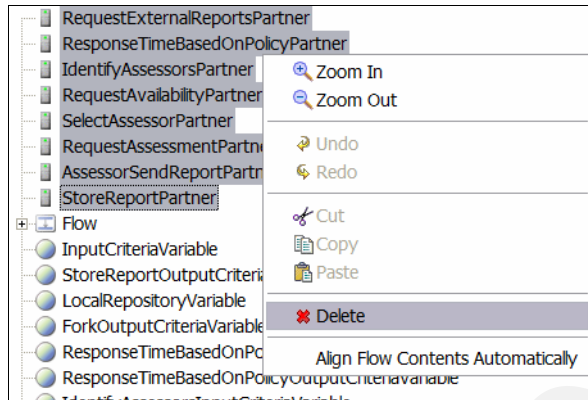
Figure 10-7   Deleting all the partner links in one go

2. Create new partner links from the architect's WSDL files:

   Select each of the WSDL files in turn and drag and drop onto the
   RequestExternalReports.bpel editor canvas. WebSphere Studio Application
   Development Integration Edition prompts you to confirm the selection of a
   service's port and port type as in Figure 10-8.
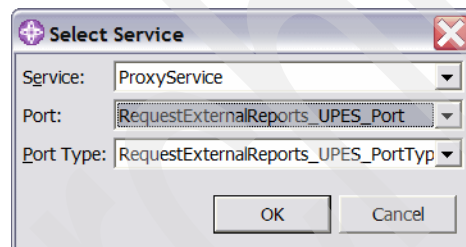
Figure 10-8   Confirm Port and Port Type

Add the partner links in the same order as the flows and your editor canvas
should look similar to Figure 10-9 on page 390.

Figure 10-9   Partner links based on interfaces from Rational Software Architect
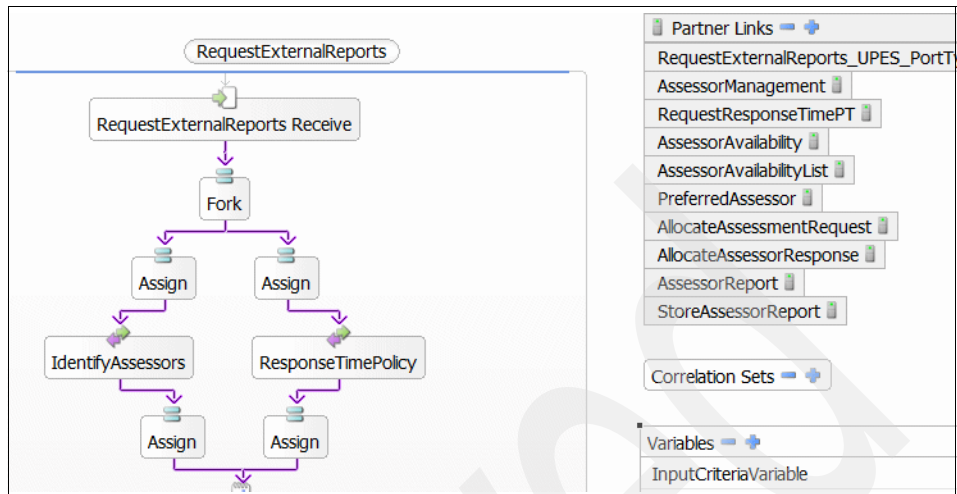
10.4.2 Integrate the partner links with the process

We now have imported the RequestExternalReports process from WebSphere Business Integration Modeler as a BPEL process and the interface definitions from Rational Software Architect as partner links. The next task is to associate the partner links with the process adding new activities in the flow to correspond with the new partner links we have added.

Connect activities in the flow with partner links

1. The first task is to connect each partner link with its corresponding activity:

   Pick an activity in the flow and select the partner link action from the palette that hovers above the mouse pointer as in Figure 10-10. Alternatively, Right-click the activity and select **Set Partner Link** from the menu.

Figure 10-10   Creating the connection between an activity and a partner link
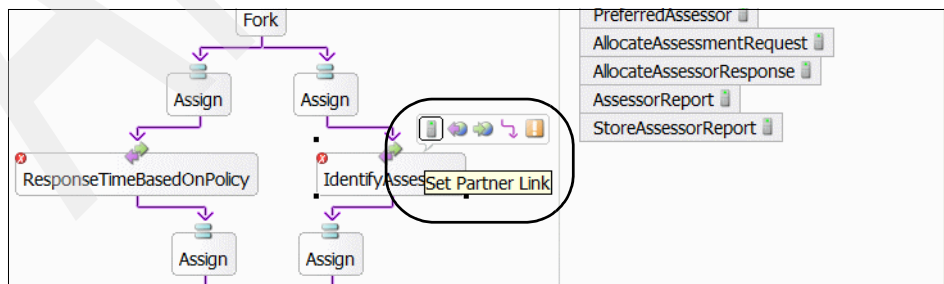
Drag the rubber-band line to the appropriate partner link. Use Table 10-1 on page 391 to guide you in the matching of connections.

An alternative way to make the connection is to select an activity and then open the implementation tab in the property editor below the graphical editor canvas. Then select the desired partner link, as in Figure 10-11. This is a method that you will need if the partner link and activity icons are too far apart to connect on the same screen.
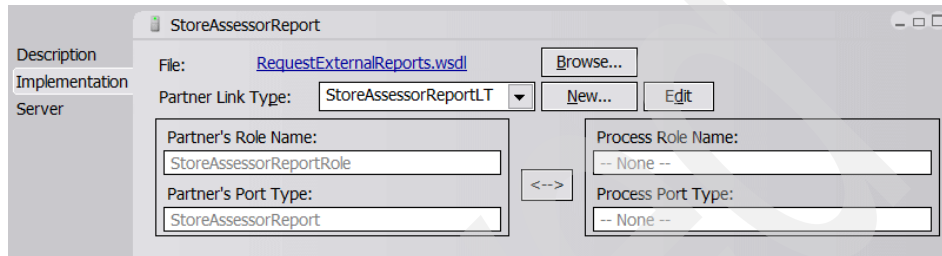
Figure 10-11   Setting partner link type using the properties editor

2. Rename each partner link to match the names in Table 10-1 (this is only for clarity, it does not alter the behavior of the process).

This leaves two partner links unattached to any activity (AssessorAvailabilityListPartner and AllocateAssessorResponsePartner). The third new partner link, AssessorReportPartner, is connected to the AssessorSendReport activity that is defined in the flow - but is the wrong type of activity. We fix these problems next.

Table 10-1   Assessor Automation process partner links and roles

Flow	Component (Interface owner)	Assessor Automation Activity & Role	Partner link name	WSDL file name
1	Assessor Automation	RequestExternal-Reports Receive *Server*	RequestExternal-ReportsProcess	ExternalClaimsAssessorInterface.wsdl
2	Assessor Management	IdentifyAssessors *Client*	AssessorManagement	AssessorManagement(2).wsdl
2a	Business Rules Engine	ResponseTime-BasedOnPolicy *Client*	RequestResponseTimePT	RequestResponseTimePT(2a).wsdl
3	Proxy Assessor System	RequestAvailability *Client*	AssessorAvailability	AssessorAvailability(3).wsdl

Flow	Component (Interface owner)	Assessor Automation Activity & Role	Partner link name	WSDL file name
4	Assessor	None	None	Availability(4).wsdl
4a	Proxy Assessor System	None	None	AssessorAvailabilityPT(4a).wsdl
3a[a]	Assessor Automation	AssessorAvailability Receive *Server*	AssessorAvailability List	AssessorAvailablityList(3a).wsdl
5	Business Rules Engine	SelectAssessor *Client*	SelectAssessors	PreferredAssessor(5).wsdl
6	Proxy Assessor System	Request-Assessment *Client*	Allocate AssessmentRequest	AllocateAssessmentReport(6).wsdl
7	Assessor	None	None	DeliverAssessment(7).wsdl
7a	Proxy Assessor System	None	None	DeliverAssessmentResponse(7a).wsdl
6a	Assessor Automation	AllocateAssessor-Response *Server*	AllocateAssessor-Response	AllocateAssessorReponse(6a).wsdl
8	Proxy Assessor System	None	None	AssessorReport(8).wsdl
9	Assessor Automation	AssessorSend-Report *Server*	AssessorReport	AssesorReport(9)
10	Claim System	StoreReport *Client*	StoreReportPartner	StoreAssessmentReport(10).wsdl

a. Red rows need to be added to the assessor automation process

Create new activities to receive messages from the assessors

First fix the AssessorSendReport activity. It is currently an Invoke activity. We want to change it to a Receive activity to receive the claims assessment report.

1. Right-click the **AssessorSendReport** activity → **Change Type** → **Receive.**

Next, add two new receive activities named as shown in column 3 in the red in Table 10-1, AssessorAvailabilityReceive and AllocateAssessorResponse.

2. Select receive activity from the palette (Figure 10-12)and drop it into the process. Type in `AssessorAvailabilityReceive` as the activity name.

Figure 10-12   Select receive activity from palette

3. Select the connector between **RequestAvailability** invoke activity and **Any assessor?** empty activity and delete it.See Figure 10-13,

Figure 10-13   Select connector

4. Connect **RequestAvailability** and **AssessorAvailabilityReceive** activities; Right-click **RequestAvailability** and select **Set Link between Flow activities** menu.

5. Connect **AvailabilityReceive** and **Any assessor?** activities. To adjust the arrangement of flow contents, **right click** the flow area and select **Align Flow Contents Automatically** on the menu.

6. **Connect** the activity with the **AssessAvailabilityListPartner** partner link. The result should look like Figure 10-14 on page 394. Note the arrow between the partner link and the activity is now black and pointing to the activity, indicating the direction of the message.
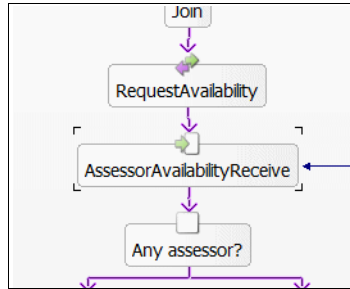
Figure 10-14   Inserted the new AssessorAvailabilityReceive activity into the flow

7. Repeat this process to insert the AllocateAssessorResponse activity in the flow and connect it to its partner link. The result is illustrated in Figure 10-15.
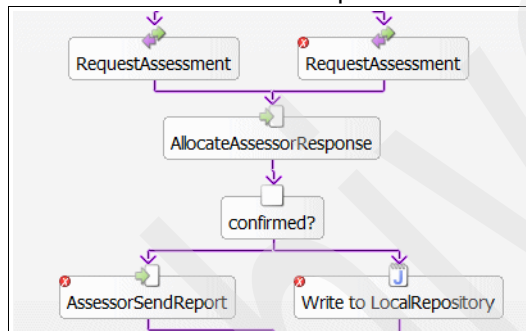
Figure 10-15   inserting the AllocateAssessorReponse activity into the flow

8. At this point it is worth saving the workspace to a zip file.

    Select the **ITSOLGI** service project in the services navigator and **right click** → **Export** → **Project Interchange** → **Check ITSOLGI** and type a file name to export to → **Finish.**

10.4.3 Configure the partner links

The next task is to assign the process and the partner the correct roles for the partner link and ensure the correct port types and operations have been selected and input and output variables have been allocated to pass the input and output messages to and from the service. Figure 10-16 on page 395 shows the relationships between the BPEL process, the Web service interface, and an EJB implementation.
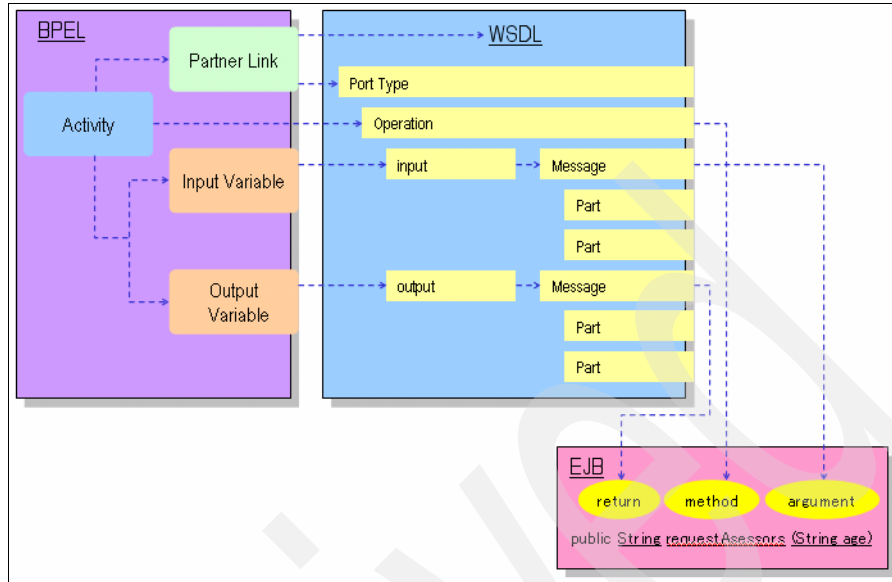
Figure 10-16   WSDL describes EJB interface invoked from BPEL process

Roles

In assigning the correct number of roles and partner links to the process, think of the relationship between the process and the partner as a conversation. If there is one conversation then there is one role to allocate. This is the case for a request/reply relationship which maps to a request/reply SOAP message. If on the other hand, there are two conversations, such as when one exchanges question and answer with voicemail, then there are two roles corresponding to two one-way SOAP messages that are exchanged asynchronously. It is also possible to have a one-way conversation, which is similar to leaving a message on voicemail without expecting a response.

In the case of the External Claim Assessor solution, we have some request/reply conversations and some one-way conversations. We have no conversations with two roles. This might seem surprising because we have three asynchronous messages. But in each case, they start a new conversation following a previous request/reply. To continue the telephone analogy, it is similar to calling someone for information and them agreeing to leave you a message later when they have researched the information for you.

Having decided that each partner link type has one role to allocate, the next question to decide is whether the role belongs to the process or the partner. The simple way of looking at whose role it is, is to ask *who is performing the service?*

If the service is performed by the process, then it is the process that owns the role. If the service is performed by the partner, then the partner owns the role.

The role played by the process is listed in Table 10-1 on page 391. Where the Assessor Automation process is the server then it owns the role. Where it is the client, the partner owns the role.

Configuring the links

It is easiest to do this using the **Outline** view. Click down through the partner links at the top of the Outline.

1. For each of the partner links that correspond to the process being the server make sure the partners role name is **-- None --** by clicking the double arrow in the implementation tab of the properties editor. See Figure 10-17. There are four partner links to assign a process role name.
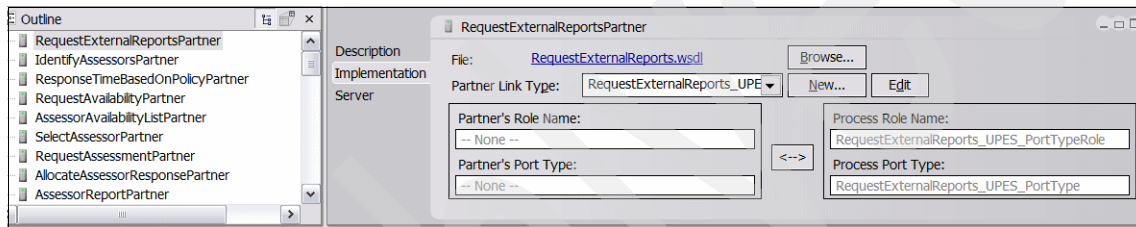
Figure 10-17 configuring the RequestExternalReports

2. Check the partner link type, role name and port type for the other partner links as you scroll through the outline view.

10.4.4 Configure the activities

Expand the Flow category in the outline, and click each activity in turn.

For each activity, on the implementation tab, check for the correct port type, operation, selected input and output variables, and create any missing variables.

1. For the RequestExternalReports Receive activity, the check box **Create a new Process instance if one does not already exist** should be checked. This is the first activity in the process and it creates a new process instance each time a new claim is processed.

2. In the AllocateAssessorResponse Receive activity we have just created, we need to create a new request variable, AssessorConfirmationResponse.

   In this case the Create a new Process ... check box is left unchecked because the Automated Assessor process has been blocked waiting for the response. It is resumed when the response is received (see Figure 10-18 on page 397).

Figure 10-18   Creating the Request variable in AllocateAssessorResponse

3. Repeat this for the AssessorAvailabilityReceive activity.

4. Add an output variable `OutputCriteraVariable` for the RequestExternalReports Reply activity.

5. The activity configurations should match those listed in Table 10-2. There are a number of variables missing, so go through the table carefully.

Table 10-2   Setting the partner links, port types, operations and variable names

Flow	**Assessor Automation Activity name**	**Port Type name**	**Request variable**
	Partner Link name	**Operation names**	**Response Variables**
1a	RequestExternalReports Receive	RequestExternalReport Process	InputCriteriaVariable
	RequestExternalReports Process	RequestExternalReport	OutputCriteriaVariable
2	IdentifyAssessors	AssessorManagement	IdentifyAssessorsInputCriteria Variable
	AssessorManagement	requestListAssessors	IdentifyAssessorsOutput CriteriaVariable
2a	ResponseTimePolicy	RequestResponseTime PT	ResponseTimeBasedOnPolicy InputCriterionVariable
	RequestResponseTime PT	requestResponseTime	ResponseTimeBasedOnPolicy OutputCriterionVariable
3	RequestAvailability	AssessorAvailability	RequestAvailabilityInputCriteria Variable
	AssessorAvailablity	requestAssessor Availability	RequestAvailabilityOutputCriteria Variable

Flow	**Assessor Automation Activity name** / **Partner Link name**	**Port Type name** / **Operation names**	**Request variable** / **Response Variables**
3a	AssessorAvailabilityReceive	AssessorAvailabilityList	ListOfAvailableAssessors
	AssessorAvailabilityList	availableAssessorsList	
5	SelectAssessors	PreferredAssessor	SelectAssessorInputCriteria Variable
	PreferredAssessor	selectAssessor	SelectAssessorOutputCriteria Variable
6	RequestAssessment	AllocateAssessment Request	RequestAssessmentInputCriteria Variable
	AllocateAssessment Request	actionAssessor	RequestAssessmentOutputCriteria Variable
6a	AllocateAssessorResponse	AllocateAssessor Response	AssessorConfirmationResponse
	AllocateAssessorResponse	assessorConfirmation Request	
9	AssessorSendReport	AssessorReport	AssessorSendReportOutput CriteriaVariable
	AssessorReport	receiveAssessorReport Request	
10	StoreReport	StoreAssessorReport	StoreReportInputCriteriaVariable
	StoreAssessorReport	storeAssessorReportURL	StoreReportOutputCriteriaVariable

a. This partner link is split into separate send and receive activities at the flow beginning and end.

10.4.5 Configure the types of input and output variables

The next step is to associate message definitions with the request and response variables for the activities. Table 10-3 on page 399 lists the messages to be associated with the variables. To associate the messages with the variables, select the variables in the Outline view and go to the message tab in the properties editor. Browse to the correct WSDL file and select the proper message as shown in Figure 10-19 on page 399.
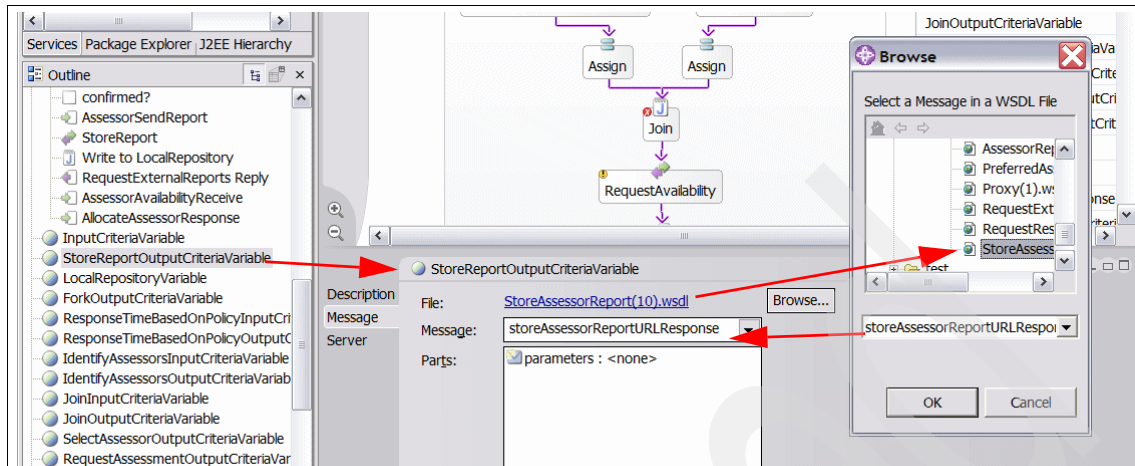
Figure 10-19   Associating messages with variables

Table 10-3   Variables and associated messages

Flow	WSDL file	Request and Response Variables	Message
1	ExternalClaimAssessors.wsdl	InputCriteriaVariable	RequestExternalReportsRequest
		OutputCriteriaVariable	requestAssessorResponseMessage
2	AssessorManagement(2).wsdl	IdentifyAssessorsInputCriteriaVariable	requestListAssessorsRequest
		IdentifyAssessorsOutputCriteriaVariable	requestListAssessorsResponse
2a	RequestResponseTimePT(2a).wsdl	ResponseTimeBasedOnPolicyInputCriterionVariable	requestResponseTimeRequest
		ResponseTimeBasedOnPolicyOutputCriterionVariable	requestResponseTimeResponse
3	AssessorAvailability(3).wsdl	RequestAvailabilityInputCriteriaVariable	requestAssessorAvailabilityRequest
		RequestAvailabilityOutputCriteriaVariable	requestAssessorAvailabilityResponse
3a	AssessorAvailablityList(3a).wsdl	ListOfAvailableAssessors	AvailableAssessorsList

Flow	WSDL file	Request and Response Variables	Message
5	PreferredAssessor (5).wsdl	SelectAssessorInputCriteria Variable	selectAssessorRequest
		SelectAssessorOutputCriteria Variable	selectAssessorResponse
6	AllocateAssessme ntReport(6).wsdl	RequestAssessmentInput CriteriaVariable	actionAssessorRequest
		RequestAssessmentOutput CriteriaVariable	actionAssessorResponse
6a	AllocateAssessorR eponse(6a).wsdl	AssessorConfirmationResponse	AssessorConfirmationRequest
9	AssesorReport(9)	AssessorSendReportOutput CriteriaVariable	receiveAssessorReportRequest
10	StoreAssessmentR eport(10).wsdl	StoreReportInputCriteriaVariable	storeAssessorReportURLRequest
		StoreReportOutputCriteriaVariab le	storeAssessorReportURLResponse

10.4.6 Map data between input and output variables

All the input and output variables are now typed to match the data in input and output messages. The next step is to map data variables received into one activity to the input variables required for the associated partner link, and then map the output results from the partner link to the input variable of the next activity. This is illustrated in Figure 10-20 on page 401 which shows control and data flows and the metadata used to assign fields in one message to fields in the next.

The mapping activity can be one of a

► Java snippet
► Transformer activity
► Assign activity

They each have pros and cons. The assign activity is the simplest way to perform straight mappings from fields in one message to equivalent fields in another structure, when the only thing that is transformed is the path to the fields. The transformer activity does not require any coding, and can handle more complex transformations. This activity includes parsing strings, which can be trickier to code. A java snippet can be a simple piece of code doing no more than an assign activity, or it can perform complex transformations.
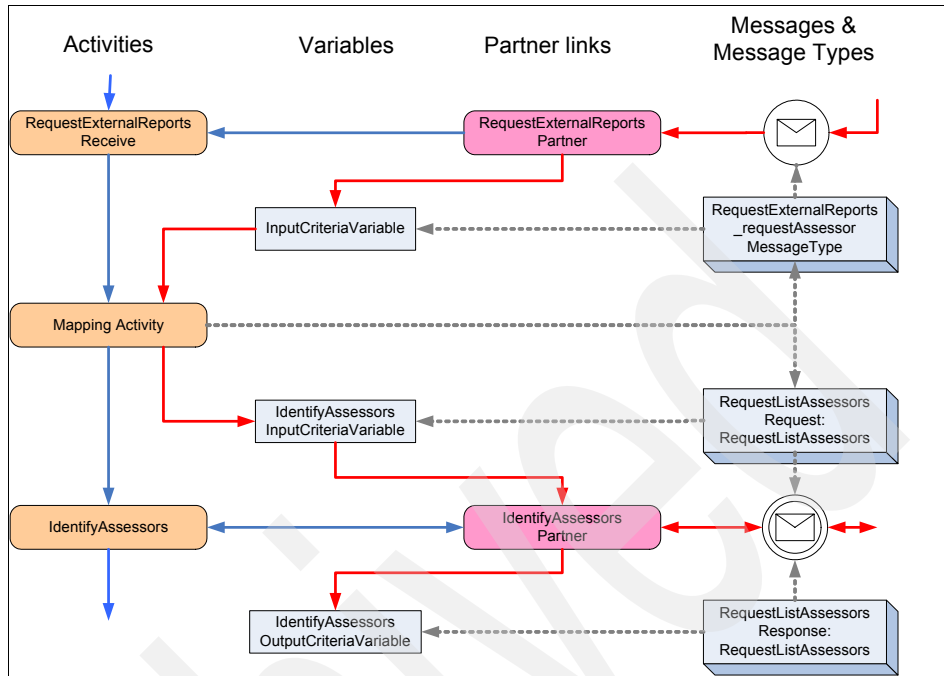
Figure 10-20 Mapping data between messages using a mapping activity

The BPEL flow exported from the Modeler generated some skeleton assignment activities. We are going to replace these and add some other data mappings to complete the data transformation part of the flow. We are limited to the extent that we can use assignment activities, because in a number of messages the WSDL requires more complex parsing, and we need to use either a Java Snippet or a transformer activity. The assign activity is much more versatile in version 6 of the Process Choreographer, and is the activity of choice for many of the mappings performed using transformers in version 5 because of its simplicity and performance.

The are several mappings, and we cover a variety of mapping techniques in the following sections:

► "Mapping IdentifyAssessors and ResponseTime" on page 402 uses two separate Transform activities.

► "Mapping RequestAvailability" on page 404 shows how to do aggregation using a Transform.

► "Mapping SelectAssessors" on page 407 uses another simple Transform activity.

► "Mapping RequestAvailability" on page 407 introduces Java Snippets as well as use a Transform activity to aggregate data from three input sources.

► "Mapping StoreReport" on page 414 is another simple Transform activity.

► "RequestExternalReport Reply" on page 415 is another simple Transform Activity.

Mapping IdentifyAssessors and ResponseTime

The first two transformations required are from the `requestAssessorMessage` to the `RequestListAssessorsMessage` and to the `requestResponseTimeRequest` message. We shall use two transformer activities.

1. Remove the Fork and two assign activities between the RequestExternalReports Receive activity and IdentifyAssessors and ResponseTimeBasedOnPolicy.

2. Create a new Transformer service. Click the icon on the action bar, or **File** → **New** → **Transformer service** (Figure 10-21).
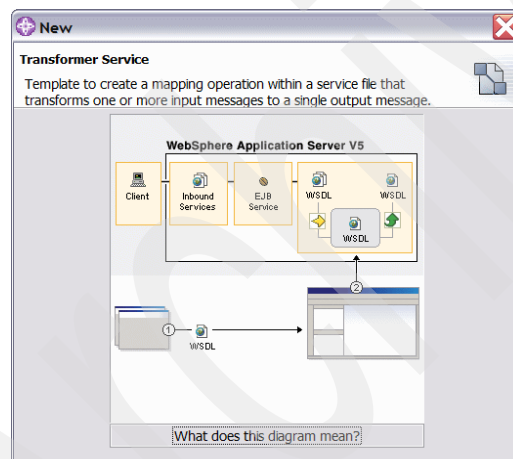
Figure 10-21   Creating a new Transformer service

3. Click **Next** → type `IdentifyAssessorsTransformer` → **Next** → type `toIdentifyAssessors` as the Operation Name → **Add** → **Browse** to ExternalClaimsAssessor.wsdl to locate the **RequestExternalReports_RequestAssessors Message** as the input message, → **Browse** to **RequestListAssessors Request:requestListAssessors** as the output message.

4. Expand the messages in the transformer editor window and drag and drop the three input fields onto the corresponding output fields. The result is illustrated in Figure 10-22 on page 403.
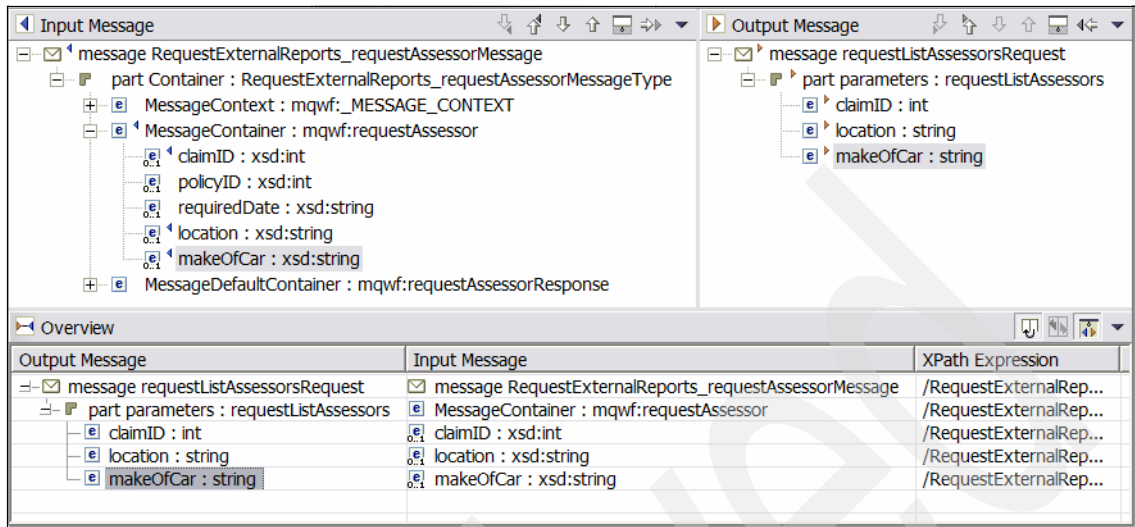
Figure 10-22   Mapping fields into the input message for IdentifyAssessors

5. Repeat this for the ResponseTimeBasedOnPolicy activity. Call the service `ResponseTimeBasedOnPolicyTransformer`.

6. **Drag** the two transformer .WSDL files onto the RequestExternalReports.bpel process and **rename** the transformer activities `ToIdentifyAssessors` and `ToResponseTimeBasedOnPolicy` (matching the operation names, just for documentation). The BPEL process now looks like Figure 10-23.
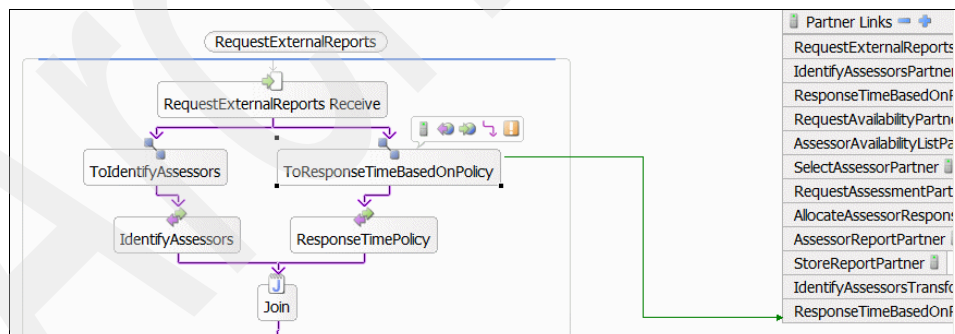
Figure 10-23   Inserted transformer services into the RequestExternalReports flow

You can explore the new partner links and activities just like any other. Open the **Implementation** tab of the ToIdentifyAssessors activity and click **Edit** to review and modify the field mappings. You can also select new operations and new transformer services for this activity. In the Services Explorer find the

IdentifyAssessorsTransformer.wsdl file, and open it using the standard .wsdl editor.

Mapping RequestAvailability

This mapping aggregates the results from the IdentifyAssessors and ResponseTimeBasedOnPolicy activities, and some of the original input data from the RequestExternalReports message into a new message. It sends the new message to the RequestAvailability partner link which will be routed by the Enterprise Service Bus to all the external claim assessors who might be able to handle this claim.

The obvious way to accomplish this task is to use another transformer activity to perform this aggregation mapping. You could repeat the procedure from the previous section to create a new Transformer service called **TransformerRequestAvailability.** This time you would add three input messages to the service as shown in Figure 10-26 on page 406.

> **Restriction:** However, starting with the Transformer Wizard and then following the same procedure as for simple transformations does not work.
>
> The solution to this difficulty is to generate the transformations using a slightly different procedure which requires more manual input and, hence, requires more care. But when it is performed correctly, it is reliable. This is the procedure that is documented next, and the one you should use for any transformation that involves multiple input messages.

1. Ensure the IdentifyAssessors, ResponseTimePolicy and RequestAvailability activities are fully connected as on the left side of Figure 10-24.

2. Drop a new transformer activity into the flow, call it `ToRequestAvailability`, and wire it up immediately before the RequestAvailability activity as on the right side of Figure 10-24.
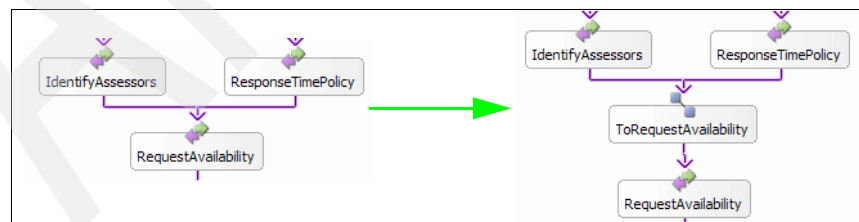
Figure 10-24   Adding ToRequestAvailability transformer activity

3. Open the ToRequestAvailability activity and set its **Response** variable to **RequestAvailabilityInputCriterionVariable**.
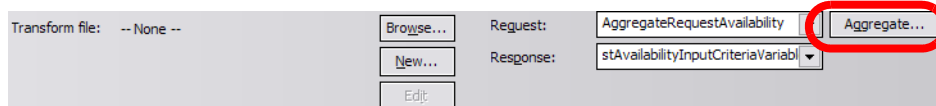
Figure 10-25   Variables defined for the ToRequestAvailability transformer activity

4. Click the aggregate button and select the input variables to be aggregated:

   – **InputCriteriaVariable**
   – **IdentifyAssessorsOutputCriteriaVariable**
   – **ResponseTimePolicyOutputCriterionVariable**

5. On the next panel type the filename `AggregateRequestAvailability`.

6. Arrange the flow contents and you can see a new Java Snippet has been inserted. Rename the snippet `AggregateRequestAvailability`.

7. Returning to the new transformer activity, click the **New...** button to create a new Transform file. In the Create a new transform dialog, rename:

   – partner link and file name to `TransformRequestAvailability`.
   – Target namespace to `http://Claim.TOBE.RequestExternalReports` (the same package as the process).
   – The port type to `TransformRequestAvailabilityPT`.
   – The Operation name to `ToTransformRequestAvailability`.

8. Press **OK**.

9. Now wire up the transformation as in Figure 10-26 on page 406.
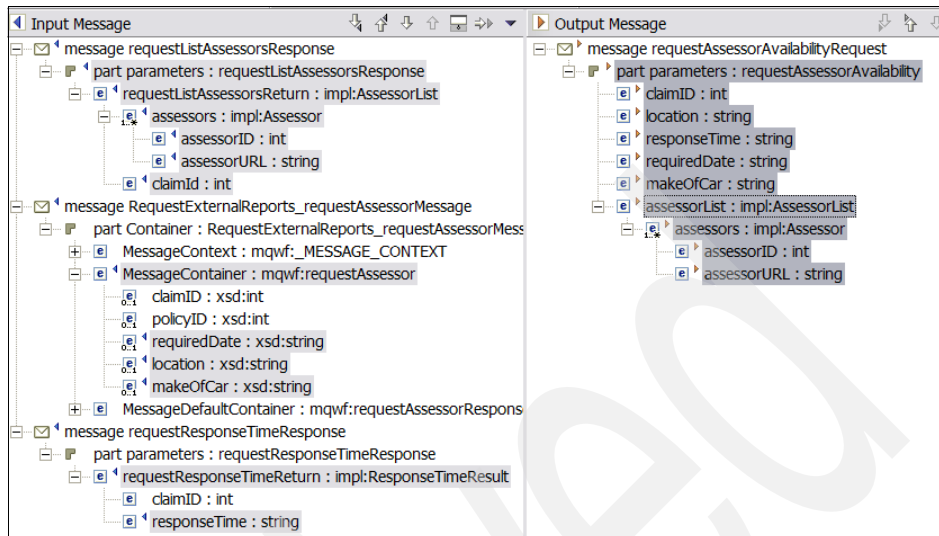
Figure 10-26   Aggregating the data for calling RequestAvailability

> **Tip:** Naming is so important! If you reformat a flow, it is so much easier to reassociate an activity that has become disassociated from its position in the flow if you named that activity with a well thought-out naming convention.
>
> We have chosen to name the Transformer and Aggregation WSDL files so that they are sorted into Aggregation and Transformer categories in the Services Navigator. An alternative approach, is to store all the WSDLs in the services package, and suffix them with Transformer or Aggregation, so that they are sorted by affinity to the original service. The important point is to think through a naming scheme to begin with and then apply it consistently. Naming consistency reduces hard-to-fix bugs caused by confusion about names.

10. The mapping activity must wait until all the input data is available. On the **Join Behavior tab of the properties editor** for the newly created **ToRequestAvailability** activity, set the condition value to **All. S**ee Figure 10-27 on page 407.
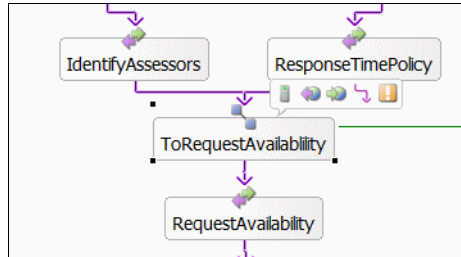
Figure 10-27   Joining results of IdentifyAssessors and ResponseTimePolicy

Mapping SelectAssessors

We use another transformer service for this mapping. Call it `SelectAssessorTransformer`. Give it an operation name of `toSelectAssessor`. `listAvailableAssessors` is the input message and `selectAssessorRequest` is the output message in the transformer mapping creation dialog box. The resulting mappings are show in Figure 10-28.

Figure 10-28   mapping the SelectAssessor request

▶ **Drag** the service onto the RequestExternalAssessors process editor, call it **ToSelectAssessor**, and **wire** it up between the Any Assessors? and SelectAssessors activities.

Mapping RequestAvailability

The transformer service for this mapping is called `RequestAvailabilityTransformer`.

1. Give it an operation name of `toRequestAvailability`. It needs to aggregate two input messages: the output of **SelectAssessors**, and the original input message to the process, **RequestExternalReports_RequestAssessors** (See Figure 10-27 on page 407). The output message is **actionAssessorRequest**.

   At this point we hit a problem. The actionAssessor message requires two fields that are not in the selected input messages:

   – carDetails.registration
   – assessor.assessorURL

2. The car's registration was accidently omitted from the input container passed to the RequestExternalReports process by the ClaimInvestigation_TOBE workflow. That can be fixed later, For now, we copy the make of car into the registration field.

3. The lack of the AssessorURL is more significant. The Business Rules Engine's interface only returns the selected Assessors ID along with the claimID. We need to reassociate the assessorID with their URL so that the ESB can route the assessment request to the right assessor.

Where should data be correlated?

We have not focused on design patterns for data modeling in this red book to help us to decide how to pass data around the process and the associated services. There are a couple of architectural questions to decide before working out the details of matching the AssessorID with its URL.

► Who owns the routing data for the assessors?

The answer to the this question is straightforward The AssessorManagement system owns the routing data for the assessors.

► Who is responsible for assembling the information needed by the ESB to communicate with the assessors?

This second answer is greyer. Either the process engine or the ESB could be responsible for the association between the assessorID and the assessorURL. There are pros and cons to both approaches.

One data flow pattern is to keep the coupling of data between components as lean as possible, and require each component to request the data it requires from other services as and the data is needed. This would argue for passing only the assessorID, the claimID, and the function that the process engine wants the ESB to perform to the ESB. The ESB would then need to look up the assessorURL in the assessor management system to route the assessment, and gather the required claim information from the claim system to issue the RequestAssessment service request to the chosen assessor.

An alternative data flow pattern commonly used in process integration designs is to make the process engine responsible for collecting all the information required by the services it uses into one or more large containers (often corresponding to industry standard data models), and to pass the services it uses the data required by them.

Correlating the AssessorID and AssessorURL in the process engine

We are using the standard process integration data pattern, so we need to reassociate the assessorURL of the selected assessor using the assessorID returned from the Business Rules Engine. To do this:

1. Add a Java Snippet called `AssessorURL`.
2. Insert AssessorURL into the RequestExternalReports flow as shown in Figure 10-29.
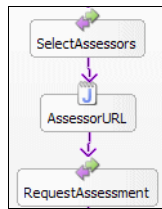
Figure 10-29   Inserting AssessorURL Java Snippet into the flow

3. A global variable, called `SelectedAssessor`.
4. Create a WSDL message called `SelectedAssessor` with two parts:
   a. AssessorID
   b. AssessorURL

   Follow these instructions to add AssessorID and AssessorURL to the RequestExternalReportsInterface.wsdl file (or creating a new wsdl is just as good).

   i.   Open RequestExternalReportsInterface.wsdl in the graphical wsdl editor.

   ii. In the messages area right-click → **Add Child** → **Message** and call it `SelectedAssessor`.

   iii. Right-click SelectedAssessor → **Add Child** → **Part** and call it `AssessorID` and make it an `xsd:int.`

   iv. Similarly, add `AssessorURL` and make it an `xsd:String.`

The following tasks are described in more detail in the next sections:

5. Import the SelectedAssessor message into the TransformerRequestAssessment service and map from SelectedAssessor.assessorURL to actionAssessmentRequest.assessor.assessorURL.

6. Code for the AssessorURL Java Snippet.

Add SelectedAssessor to TransformerRequestAssessment

Now that we have the assessorURL for the selected assessor, we need to map it to the assessorURL field in the request message being sent to

RequestAssessment. We need to create an aggregating transformer just like ToRequestAvailability.

1. Call this transformer `ToRequestAssessment`.

2. The response variable will be `RequestAssessmentInputCriterionVariable`.

3. The Aggregation filename will be `AggregateRequestAssessment`.

4. The Input variables to be aggregated are:

   – InputCriterionVariable
   – SelectAssessorOutputCriterionVariable
   – `SelectedAssessor`

   You will need to create this input variable and type it with the SelectedAssessor message you defined in the last step.

5. The Transformer filename is TransformRequestAssessment. Follow the pattern from before for naming the partner link, operation, and so forth.

6. Reflow the flow diagram when you have completed the tasks so that it looks similar to Figure 10-30.
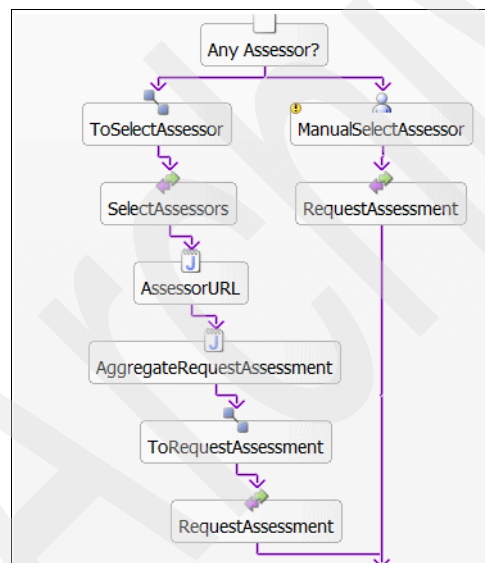
Figure 10-30   Select and request assessment flow diagram part

That should complete the mappings. Now, you need to set the value of AssessorURL in the Java Snippet.

Creating the AssessorURL Java snippet

In this Java snippet, we need to retrieve the list of all the eligible assessors that were examined by the Business Rules Engine and identify the assessor the rules engine chooses. The assessors in this list have a data structure that includes assessorURL. Having found the chosen assessor, we create the SelectAssessor global variable to pass the assessorURL on to the ToRequestAssessor transformer service which will assign assessorURL into the actionAssessment Request message. The completed Java Snippet is shown in Example 10-4 on page 413. We we go through the steps of how to create this Java snippet using the autocompletion assistance in WebSphere Studio Application Development Integration Edition.

1. Select the newly created AssessorURL Java Snippet and open the implementation tab in the properties editor.

2. Insert the statement to declare we will update the assessorURL part of the SelectedAssessor global variable.

    Right-click to open the menu → **Update Variable** → **Part...** → and select **assessorURL** from the SelectedAssessor message. See Example 10-1.

Example 10-1   AssessorURL Java Snippet, lines from Update Variable wizard:

```
SelectedAssessorMessage SelectedAssessor = getSelectedAssessor(true);
java.lang.String assessorURL = SelectedAssessor.getAssessorURL();
<focus>
SelectedAssessor.setAssessorURL(assessorURL);
```

3. At the focus, declare the local variable `Assessor assessorList []` which will contain the list of assessors from the response message from IdentifyAssessors.

    a. Complete the declaration by getting the list of possible assessors from the IdentifyAssessorsOutputCriteriaVariable.

        i. Right-click to open the menu → **Get variable** → **Part...** → and select **IdentifyAssessorsOutputCriteriaVariable.**

        ii. Autocomplete the line by using **.** → **CTRL_SPACE** → **getParameters** and so forth.

    The completed line of code is in Example 10-2.

Example 10-2   AssessorURL Java Snippet, lines from Get Variable wizard

```
Assessor assessorList [] =
getIdentifyAssessorsOutputCriteriaVariable().getParameters().
getRequestListAssessorsReturn().getAssessors();
```

> **Tip:** When you start typing Assess... and use autocomplete to finish entering the Assessor class, does it work? If not, then there is no import statement for itso.lgi.assessormgmt in the Java source file containing the Java Snippet.
>
> Itso.lgi.assessormgmt is the target namespace for the AssessorManagement WSDL messages. WebSphere Studio Application Development Integration Edition creates a Java package with the name of this namespace to contain all the Java classes that implement the setters and getters for the message parts.
>
> Look in the Services navigator for the package and its classes. If you find it is there, and it is inside the ITSOLGI project, then you need to coerce WebSphere Studio Application Development Integration Edition to include an import statement for the package. If the first method does not work, try the second:
>
> ► On the right-click menu click **Source** → **Add Import** → and select **itso.lgi.assessormgmt**. If that does not work, then try
>
> ► Type n the declaration the old fashioned way → **Save** → right-click → **Source** → **Organize Imports** → **Save**.
>
> This should create the import statement, make autocomplete work, and clear any errors that have been marked in the code you have just written.

4. Add the line of code to get the selected assessor ID from the Business Rules Engine output variable, selectedAssessorOutputCriteriaVariable. We need to create a new local variable to hold the value:

```
Integer selectedAssessorID
```

And then initialize it using autocomplete with:

```
= getSelectAssessorOutputCriteriaVariable().getParameters()
.getSelectAssessorReturn().getAssessorID();
```

5. Construct a loop to match the selected assessor and set the assessorURL in the global variable. We use the code template wizard to generate the loop.

   a. Type **for** and **press CTRL_SPACE** without typing a space.
   b. Select **for - iterate over an array with temporary variable.** The following code in Example 10-3 is generated:

Example 10-3   Generate for loop with temporary variable

```
for (int i = 0; i < assessorList.length; i++) {
        Assessor assessor = assessorList[i];
<focus>
}
```

6. At the focus, we insert an if statement to test the assessorID for a match.

    a. Type `if` and use autocomplete to create a simple if template:

        if (*condition*) { }

    b. Use autocomplete to replace *condition* with:

        ```
        assessor.getAssessorID() == selectedAssessorID
        ```

7. In the if statement, block set the assessorURL in the global variable.

    a. Use autocomplete to create the local assignment statement:

        ```
        assessorURL = assessor.getAssessorURL();
        ```

    b. Use the right-click menu → **Set Variable** → **Part...** → select the
        AssessorURL part of the **SelectedAssessor** message to generate the
        code:

        ```
        getSelectedAssessor(true).setAssessorURL(<focus>);
        ```

    c. Replace the focus by the assessorURL local variable.

The completed java snippet should save without errors. The full source is shown
in Example 10-4.

Example 10-4   Complete AssessorURL Java Snippet

```
// we are going to update the SelectedAssessor assessorURL global variable
SelectedAssessorMessage SelectedAssessor = getSelectedAssessor(true);
java.lang.String assessorURL = SelectedAssessor.getAssessorURL();
// Get all the assessors that bid for the claim assessment work
Assessor assessorList[] =
    getIdentifyAssessorsOutputCriteriaVariable()
        .getParameters()
        .getRequestListAssessorsReturn()
        .getAssessors();
// Get the assessorID or the assessor who was selected
Integer selectedAssessorID =
    getSelectAssessorOutputCriteriaVariable()
        .getParameters()
        .getSelectAssessorReturn()
        .getAssessorID();
// Iterate thru the list of assessors who bid till we match one
for (int i = 0; i < assessorList.length; i++) {
SelectedAssessor.setAssessorURL(assessorList[i].getAssessorURL());
SelectedAssessor.setAssessorID((assessorList[i].getAssessorID()).intValue());
    if (assessorList[i].getAssessorID() == selectedAssessorID) {
       break; // break loop with correct assessor, else last in list
}
// Update the SelectedAssessor message in the global container
setSelectedAssessor(SelectedAssessor);
```

Add SelectedAssessor to RequestAssessmentTransformer

Now that we have the assessorURL for the selected assessor, we need to map it to the assessorURL field in the request message being sent to Request Assessment.

1. Open the RequestAssessmentTransformer.wsdl file with the Transformer Editor. There are any number of ways to do this. Here is one of them:

   a. In the Services Explorer, Right click **ITSOLGI** → **Claim.TOBE. RequestExternalReports** → **RequestAvailabilityTransformer**.**wsdl.**

   b. **Open With** → **Transformer Editor.**

2. Add the new SelectAssessors message to the input messages.

   a. With the RequestAssessmentTransformer.wsdl open in the Transformer Editor, select **Transformer Editor** in the main action bar → **Add Input Message** → Select **ProcessMessages.wsdl** in the ITSOLGI\Services\ ITSOLGI Architecture directory.

   b. Choose the **Selected Assessor** message.

3. Map the assessorURL part from the SelectedAssessor message to the assessorURL in the actionAssessorRequest message.

To complete the mappings for now, map the makeOfCar to both the makeOfCar and the registration fields so we have all the fields mapped to something.

Mapping StoreReport

For this mapping we cannot use a simple assignment because the input and output message formats do not match exactly. Again, the easier mapping mechanism is a Transformer service.

1. Add a new Transformer Service called `StoreReportTransform`.

   a. Set the operation name to toStoreReport,

   b. Set the Input message to receiveAssessorReportRequest in the AssessorReport(9).wsdl file.

   c. Set the Output message to StoreAssessorReportURLRequest in the StoreAssessorReport(10).wsdl file.

   d. Map the fields across and save the StoreReportTransform service.

2. Drag and drop the **StoreReportTransform** service, wire it into the flow, and name it as show, in Figure 10-31.
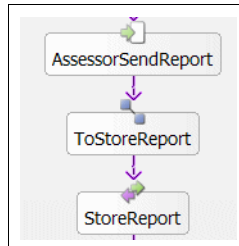
Figure 10-31   Wiring in the ToStoreReport Transform activity

RequestExternalReport Reply

Follow these steps.

1. We use another Transformer service to aggregate the response fields from the following messages:

   a. **PolicyID** from RequestExternalReportsRequest.
   b. **assCompDate** from StoreAssessorReportURLRequest
   c. The **claimID** and the **reportLocation** from StoreAssessorReportURL Response

   The output message is requestAssessorResponseMessage.

2. Call the new Aggregation `AggregateRequestExternalReportsReply` and the Transform service `TransformRequestExternalReportsReply` with an operation code of `ToRequestExternalReportReply`.

3. Create the aggregating transformer service as before (Figure 10-32).
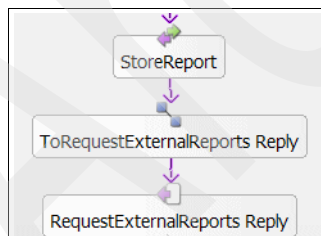
Figure 10-32   Wiring the ToRequestExternalReports Reply Transform Activity

Summary

That was a long task, but we have completed the basic mapping of data between services and activities. There is still some more fine tuning to the control structures and manual activities, but at this point we return to integrating other aspects of the services with the flow before dealing with the control structures.

10.4.7 Configuring the flow to wait for responses from the assessors

Three of the activities wait to receive responses from the assessors with the proxyAssessorSystem. We need to set up the process to wait for these messages[2] and ensure that the messages sent to the Automated Assessor are received by the right instance of the process.

Business Process Choreographer has a mechanism to store process identification information which is called a *correlation set*. Correlation sets hold aliases to map fields from different messages onto the variables used for correlation. This is a more flexible mechanism than requiring all messages to have the same key field, clearly impractical when you consider that the format of the messages can be decided by a business partner in a different organization, or there are messages with formats determined by many different standards.

When a long running process instance is suspended, Process Choreographer stores the status of the instance in nonvolatile storage. When a reply message comes back from WebSphere BI Message Broker, Process Choreographer checks the correlation sets and starts the correct process instance again.

Complete the following steps to add correlation sets.

1. Click the **+** icon on correlation sets bar (Figure 10-33) to add a new correlation set. Call the new correlations set `Claims`.
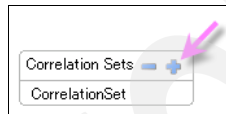
Figure 10-33   Add a new correlation set

2. Click **Claims** and select the **properties** tab in the detail area. Click the **New...** button. The Create message property window opens.

3. Type in `claimID` as a property name and select **xsd:int** from the building-type drop-down list.

4. Click the **New...** button on the top of the aliases list. Click the **Browse...** button in the Create property alias window and select **ITSOLGI** → **services** → **ITSOLGI Architecture** → **WSDL** → **Proxy(1).**wsdl → Select **RequestExternalReports_requestAssessorMessage** in the spin box →

---

[2] In the overall interaction design, there are a number of communicating sequential processes that we assume interact in the way specified by the business process analyst. The assumption is critical for this design to work. If the messages for a claim instance could arrive in a different order, then we would need to deal with out of sequence messages. The BPEL Pick activity might be more appropriate in the implementation than the Receive activity.

> OK → Expand **Part: Container : → MessageContainer : requestAssessor → claimID : Int → OK**.

5. Repeat step 4 to add the aliases in Table 10-4.

Table 10-4   Aliases for correlation set

Activity name	WSDL file name	Message	Part
RequestExternal Reports Receive	Proxy(1)	RequestExternal Reports_request AssessorMessage	MessageContainer : requestAssessor: claimID
Assessor Availability Receive	Assessor AvailabilityLis(3a)t .wsdl	listAvailable Assessors	parameters -> claimID:int
AllocateAssessor Repsonse	AllocateAssessor Response(6a).wsdl	Assessor Confirmation Request	claimID:int
AssessorSend Report	AssessorReport(9) .wsdl	receiveAssessor ReportRequest	claimID:int

Next we associate the correlation set with each of the receive activities.

6. Click **RequestExternalReports Receive** activity in BPEL editor and select **correlation** tab in the detail area.

7. Click **Add** and change the initiation value to **Yes (**Figure 10-34**)**.
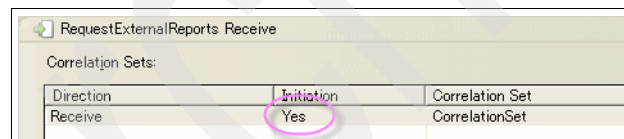
Figure 10-34   Assign correlation set to receive activity

8. Repeat steps 1 through 7 to assign correlation sets to the receive activities in Table 10-5.

Table 10-5   Correlation set prop

Activity name	Direction	Initiation	Correlation set
RequestExternalReports Receive	Receive	Yes	Claims
AssessorAvailabilityReceive	Receive	No	Claims
AllocateAssessorRepsonse	Receive	No	Claims
AssessorSendReport	Receive	No	Claims

10.5 Controlling the path through the process

In the next steps of the configuration, we provide the logic to control the path through the process. There are two parts to this task.

1. In "10.5.1, "Checking the results from RequestAvailability", if the Assessor Management system does not find any assessors to perform the assessment, then control is passed to the Claim handler rather than to the Business Rules Engine to select an assessor.

2. In 10.5.2, "Create a While activity to test for a committed assessment" on page 423 dealing with the case where the selected assessor decides not to perform the assessment after all.

10.5.1 Checking the results from RequestAvailability

The RequestAvailability partner link responds with a synchronous response from the Enterprise Service Bus. The response is a simple acknowledgement that the request to ask the assessors for their availability has been received by the bus. Because it is not part of the business logic here, we ignore the response in this flow. The response is is outside the immediate scope of our scenario. Remember 1.1.2, "Scope of the scenario" on page 5.

Monitoring the scenario is itself a large topic and beyond our scope in this book. Monitoring goals (Service Level Agreements) have to be set and analyzed to produce business measures, and the measures then need to mapped to event data that is collected from the business process and the IT platform. The events and measures have to be presented in a form suitable for the different roles responsible for monitoring and maintaining the process, from the CEO to the claims supervisor and the IT shift operators.

However, we are interested in receiving the bids from the assessors for performing the assessment. The first step in analyzing these responses is to check if any have been received. The ESB waits for the agreed period of time for the assessors to respond and then collects their bids into a single list. The ESB sends the list back using the automated assessor process. The automated assessor process is waiting on the AssessorAvailability Receive activity to pass the list on to select an assessor automatically using the Business Rules Engine.

We must check if any bids were received. If no bids were received, we hand the problem to the Claim handler as a manual activity. If bids have been received, then we pass control to the SelectAssessor automated activity, which calls the Business Rules service to choose the best assessor for the job from the list.

How can we tell if any bids were received? The only way to is test the AssessorEstimationArray for having any elements (Figure 10-35).
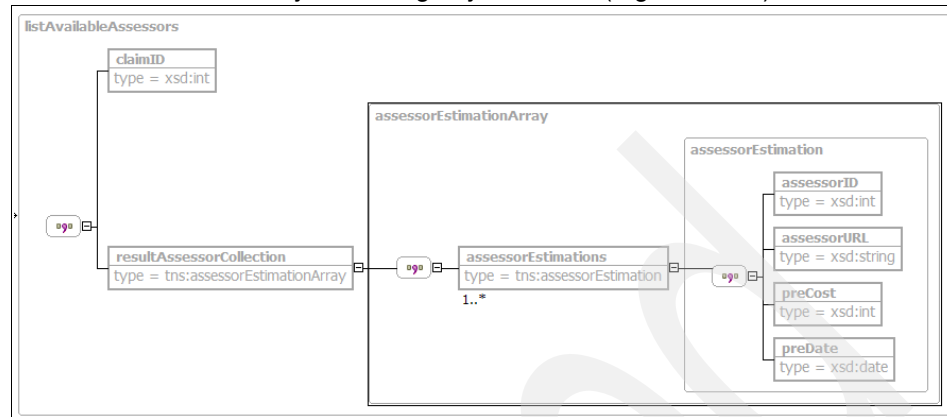
Figure 10-35   assessorEstimationArray

The results of the test are used to set the Boolean value of the links A and B in Figure 10-38 on page 421. For each link we can open up the properties tab for the link and set the condition that must be met for the link to be traversed. We plan to use either the expression or the visual expression editor to construct the condition. However, it turns out that neither the expression editor nor the visual expression editor can handle an expression complex enough to test for the existence of the AssessorEstimationArray. Instead we shall write some Java code in a Java Snippet to set a global variable. Each link can then test the variable in a simple expression to set the link's truth value.

The overall implementation plan is as follows:

1. Create a global variable to hold the length of the AssessorAutomationArray.
2. Create a Java Snippet which sets the value of the global variable.
3. Create a visual expression for each of the links A and B to test the value of the global variable and set the truth condition for each link.

Create EstimationArrayLength global variable

To create the EstimationArrayLength global variable, follow these steps:

1. In the RequestExternalReports process editor click **Variable +** to create a new global variable called EstimationArrayLength.

2. The variable needs to be typed to xsd:int to hold a length value. To do this, we need to define a WSDL message we call `EstimationArrayLength` containing a xsd:int part called `lengthValue`. The message will be stored in the same WSDL file as the AssessorID and URL variables.

   a. Create the EstimationArrayLength message and lengthValue part.

i. Open **RequestExternalReportsInterface .wsdl** with the graphical WSDL editor.

ii. Click **Messages** → **Add Child** → **EstimationArrayLength** → **Add Child** and call the part `lengthValue` → **Set Type** → **xsd:int.** See Figure 10-36.
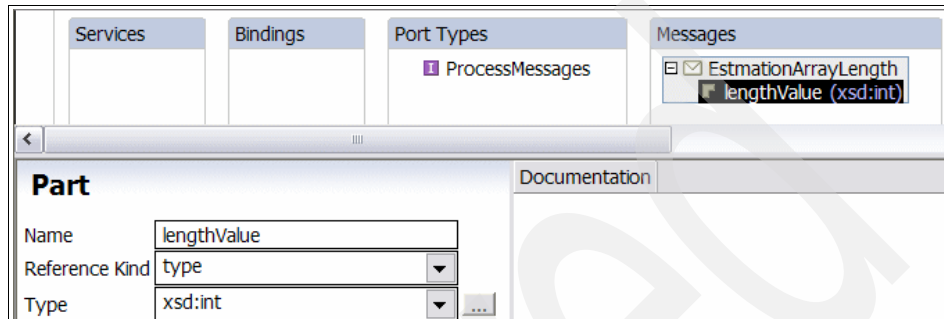
Figure 10-36   Creating the EstimationArrayLength message

b. Type the EstimationArrayLength global variable using the new message.

i. Select the **EstimationArrayLength** global variable in the Request ExternalReport process and open the **Message** tab in its properties editor.

i. **Browse**... to pick the RequestExternalReportsInterface.wsdl file and Select the **EstimationArrayLength** message in the messages spin box (Figure 10-37).
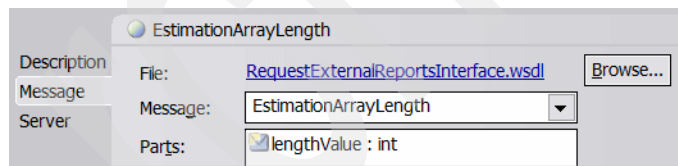
Figure 10-37   Typing the EstimationArrayLength variable

Create the Any Assessors? Java Snippet to set lengthValue

To creat eh AnyAssessors? Java Snippet, follow these steps:

1. Change the Any Assessors? null activity to a Java Snippet as illustrated in Figure 10-38 on page 421. In this Java Snippet we shall set lengthValue in the EstimationArrayLength global variable so it can be tested in simple expressions associated with links A and B.
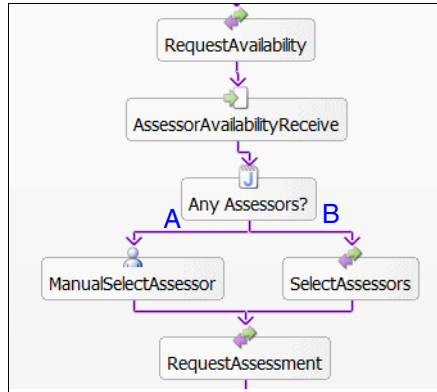
Figure 10-38   Checking the results from RequestAvailability

2. In the Java Snippet we have to complete the following tasks:

   a. Establish update access to the EstimationArrayLength global variable.

      i. Select the **Add Assessors?** Java Snippet and open the **Implementation** tab in its property editor.

      ii. **Right-click** to open up the menu → **Update Variable** → **Part ...** Select **EstimationArrayLength: lengthValue : int** (Figure 10-39). This creates three statements.

```
EstimationArrayLengthMessage EstimationArrayLength = getEstimationArrayLength(true);
int lengthValue = EstimationArrayLength.getLengthValue();

EstimationArrayLength.setLengthValue(lengthValue);
```

Figure 10-39   Results of Update Variable wizard

The first statement instantiates a new EstimationArrayLengthMessage in the JVM™, and (by setting the parameter of getEstimationArrayLength to true) indicates that the Java Snippet requires update access to the EstimationArrayLength global variable.

The second statement instantiates a new int, lengthValue. (It also initializes the value of the int from the value in the global variable, but that is not of interest here.)

The third statement uses the new int, lengthValue, to set the new value of the global variable. This also has the side effect of unlocking the variable as it is no longer required for update.

The Update wizard leaves the editor focus ready to set the value of the new int variable, lengthValue. See Figure 10-40 on page 422.
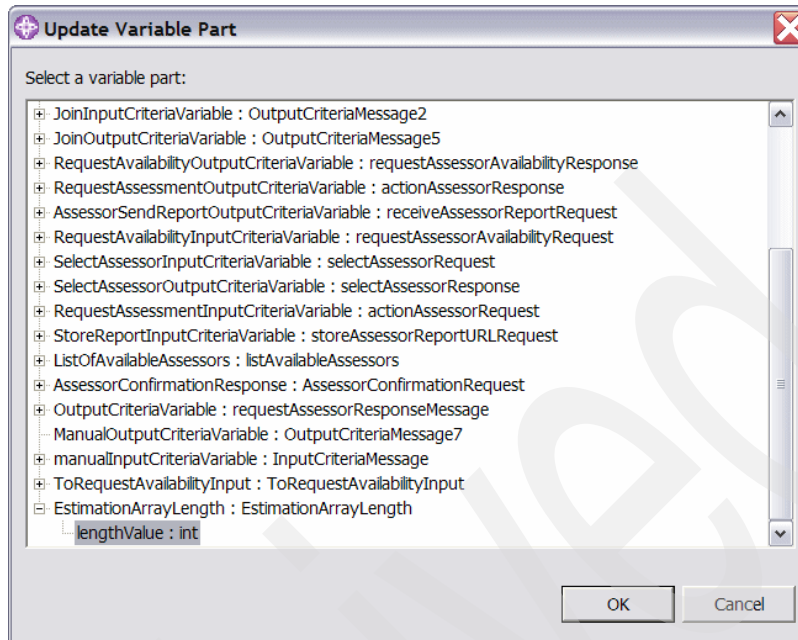
Figure 10-40   Selecting part of a variable to update in a Java Snippet

b. Get the length of the array holding the assessors responses. This time we use in-line autocompletion to get the length of the AssessorEstimationArray. Unfortunately the menu-driven approach is not able to navigate to the part of the message we want. Using Ctrl+Space to autocomplete each element, we end up with the following statement:

```
lengthValue = getListOfAvailableAssessrs()[3].getParameters().
getResultAssessorCollection().getAssessorEstimations().length;
```

The full snippet is shown in Example 10-5.

Example 10-5   Any Assessors? Java Snippet

```
EstimationArrayLengthMessage EstimationArrayLength =
  getEstimationArrayLength(true);
int lengthValue =
  getListOfAvailableAssessors()
    .getParameters()
    .getResultAssessorCollection()
    .getAssessorEstimations()
    .length;
EstimationArrayLength.setLengthValue(lengthValue);
```

---

[3] This is a typo in our code. Check that your spelling is consistent!

Test the value of lengthValue in the links A and B

The final step in the process is to add visual expressions to the two links leading from the Java Snippet, Add Assessors?, so that either the ManualSelectAssessor or the Automatic SelectAssessor activity is executed, but not both.

1. Click the link leading to **ManualSelectAssessor** and open up the **Condition** tab in its property editor. Select the visual expression editor to construct the following statement:

    ```
    EstimationArrayLength.lengthValue == 0
    ```

    a. Using the menu at the right-hand side of the editor, pick **EstimationArrayLength.lengthValue** from the global variable list.

    b. Click the equivalence operator (**==**).

    c. Click the Number function and type **0.**

2. Click the link leading to **SelectAssessor** and repeat the procedure to produce the statement:

    ```
    EstimationArrayLength.lengthValue ¬= 0
    ```

10.5.2 Create a While activity to test for a committed assessment

The RequestAssessment service sends a request to the selected assessor to perform the claim assessment for LGI. Eventually, the request results in three message flows back the to the RequestExternalReports process.

1. A synchronous reply from the ESB that the request is being processed

2. An asynchronous acknowledgement from the assessor that they will (or will not) be performing the assessment

3. An asynchronous reply containing the assessment report

As in the case of requesting assessors availability, we ignore the reply from the ESB. It is there purely for monitoring purposes and outside the scope of this scenario. The ultimate reply containing the report is handled by the AssessorSendReport activity. In this section, we examine handling the acknowledgement from the assessor that is received in the AllocateAssessorResponse Activity.

Choosing a design

If the assessor declines the request, then because this is an unusual or uncommon occurrence, rather than automatically select a second-best assessor, the Claim handler is passed the job of finding another assessor.

From the business perspective, it is important that when anything unusual happens in the process, responsibility passes to the Claim handler. Although there is technical scope for using some of the automation provided in the happy path to assist the Claim handler select a new assessor in this unusual case, the business specification, as captured in the BPEL from WebSphere Business Integration Modeler, is for control of the selection to pass to the Claim handler. After the selection is made, control is returned to the automated process to pass the assessment request to the chosen assessor. This specification should be honored in the executable BPEL being generated by the IT specialist, but there are multiple ways of achieving this business goal. Should the process use one or more manual activities? How should requests to second and possibly subsequent assessors be coded? These questions are not addressed by the business process specification and are being left to the IT process specialist to decide.

Use the existing manual activity or add a new manual activity?

There is already one place in the flow where the Claim handler is involved in a manual activity, if there are no eligible assessors to assess a claim. One design approach is to use this same manual activity for two situations:

► If there are no eligible assessors
► If the chosen assessor declines to perform the assessment

An alternative way to design the process is to add a new manual activity to deal with the negative acknowledgement from the assessor. The results of the new manual activity need to be fed back into the RequestAssessment activity for the process to continue.

If we were focusing on the specification of the user interface to the Claims process, then the question whether to use one or two manual activities would almost certainly resolve itself into using two activities: the Claim handler needs different information in each case. But for the purposes of this scenario, we stick with having only one manual activity which will ask the Claim handler for a new Assessor ID, whether this is the first attempt at assigning an assessor or not.

Designing a control loop in BPEL

We need to create a loop in the BPEL process to pass control back to the manual activity if the assessor declines to produce the assessment report. Simply returning control to an earlier point in the flow by wiring the flow with a loop is not valid BPEL. There are two alternatives to use:

► a Compensation
► a While activity

Compensation is a way of recovering from a failed activity. In its simplest form, each activity has two services:

► A *forward* service, the one that is performed in the happy path
► A *compensation service,* the alternative service performed when things have gone wrong to undo any commitments made in the forward service

For more information about compensation services, see:

► An article by Susan Hermann on developerWorks, *Modeling compensation in WebSphere Business Integration Server Foundation Process Choreographer*, for a good account of using Compensation Services in WebSphere Studio Application Development Integration Edition:

  http://www-128.ibm.com/developerworks/websphere/techjournal/0412_herrmann/0
  412_herrmann.html

► A tutorial about using compensation in the Claims Handling scenario implemented with an earlier version of WebSphere Studio Application Development Integration Edition: Larry Yusuf, *Create Compensation in a Business Process*:

  https://www6.software.ibm.com/developerworks/education/i-merge7/index.html

In our case, a compensation possibility might be to invoke Manual SelectAssessor instead of Automatic SelectAssessor as the compensation for the failure of the selected assessor to commit to performing the assessment. But closer examination should convince us that what we need to do does not quite fit the compensation model:

► We do not actually have any resource changes to back out of, nothing has changed as a result of the assessor declining to perform the assessment.

► Compensation works at the level of the process as a whole, and we only want to adjust the results of a few activities. We would have to separate the activities to be compensated into a separate process.

► Performing the Manual SelectAssessor is not a compensation service to the Automatic SelectAssessor. It is doing the same activity a different way.

The While activity provides the capability we require. Figure 10-41 on page 426 shows which activities are moved inside the While activity, and where the activity fits into the process as a whole.

► The *while condition* is the lack of positive acknowledgement from the AllocateAssessment Response Activity.

► The Automatic SelectAssessors activity is inside the loop, although it is only ever performed on the first iteration. This is because there might be no selected assessors on the first iteration and we use the manual Select

Assessor activity for finding an Assessor the first time around, if the automatic activity has failed to find one.

► We have not focussed on the data movement in and out of ManualSelectAssessor, so the manual leg of the loop might need further refinement.

► A noop activity was added at the start of the While activity purely to tidy up the graphic (Figure 10-41).
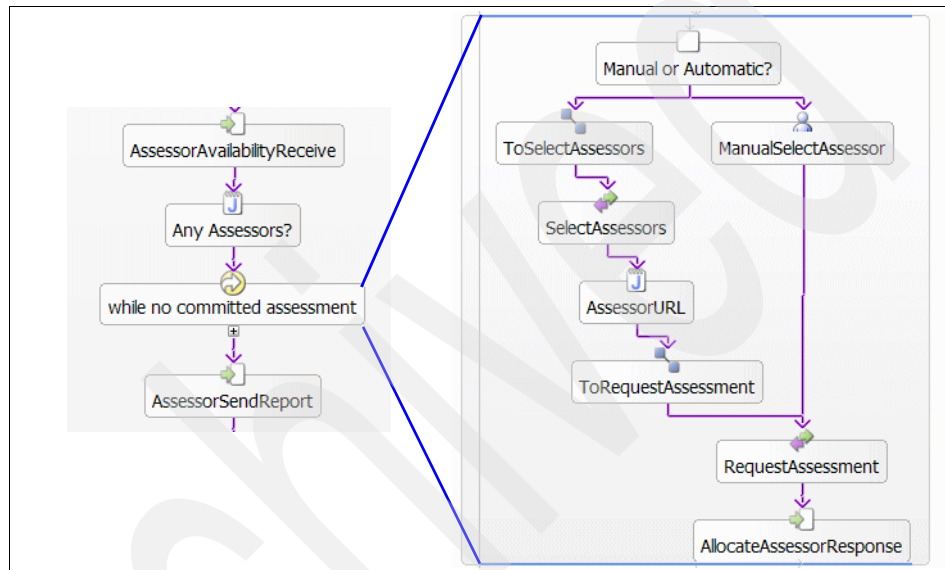
Figure 10-41   While no committed assessment

Creating the While activity

Attention! Be sure to follow these instructions when working in this section:

► The best way to create the While activity is to add the Manual or Automatic Noop activity to the flow first, then you will find it easier to select the relevant activities and links and preserve the conditions your have just written.

► It is absolutely essential you do a project interchange export before doing this edit. Nine times out of ten, the first attempts at this edit fail badly and cannot be undone.

► Make sure auto arrange is not enabled during these editing steps. It will make your task almost impossible.

1. Wire up the While Activity:

   a. **Drop** the While activity from the palette onto the process and name it `while no committed assessment`.

   b. Select all the activities between **Any Assessor?** and **AssessorSendReport** and drag them inside the While activity.

   c. **Drop, name and wire up** the **Noop** activity calling it `Manual or Automatic?` Be careful to preserve the same links (A and B) to ToSelectAssessors and ManualSelectAssessor which have conditions coded for them.

2. You will get a lot of error messages (Example 10-6).

Example 10-6   Error messages for wiring the While activity

```
Error BPED0211E: Link 'Linknn' does not link immediate children of flow
'RequestExternalReport' of process model 'RequestExternalReports'.
RequestExternalReports.bpelITSOWorkshop/Claim/TOBE/RequestExternalReports
```

To fix them, follow these steps:

   a. Right-click **RequestExternalReports**.bpel flow → **Open with** ... Text editor.

   b. Cut all the offending link definitions from the `<link> ...</link>` section of the main flow and paste them into notepad.

   c. Create a new `<link> ... <.link>` section in the while flow and paste in all the links from notepad as illustrated in Example 10-7 on page 427.

Example 10-7   New While links section

```
<while condition="DefinedByJavaCode" name="Whilenocommittedassessment"
wpc:displayName="While no committed assessment" wpc:id="37">
```

```
                       <wpc:condition>
<wpc:javaCode><![CDATA[//@generated
//
//AssessorAcknowledgement.Ack.equals("YES")

return getAssessorAcknowledgement().getAck().equals
"YES";]]></wpc:javaCode>
                       </wpc:condition>
                       <target linkName="Link27"/>
                       <source linkName="Link5"/>
                       <flow name="Flow" wpc:displayName="Flow" wpc:id="38">
               <links>
                   <link name="Link14"/>
                   <link name="Link15"/>
                   <link name="Link4"/>
                   <link name="Link16"/>
                   <link
name="ManualSelectAssessorOutput_to_RequestAssessmentInput2"/>
                   <link name="Link19"/>
                   <link name="Link18"/>
                   <link name="Link17"/>
                   <link name="Link3"/>
                   <link name="Link25"/>
               </links>
```

3. Create a variable to store the while condition:

   a. **Add** the Message `AssessorAcknowledgement` to ProcessMessages.wsdl to join the other internal messages defined for the process. **Create** a **Part** called `Ack` with type xsd:string.
   b. **Add** a variable `AssessorAcknowledgement` to the process.
   c. Type `AssessorAcknowledgement` using the newly defined message.

4. Set and Test the While condition:

   a. **Add** an **Assign activity** before the While loop to initialize AssessorAcknowledgement.Ack to `No Assessors selected yet.`
   a. Open the **condition** tab of the While activity in its properties editor and select the **Visual Expression** editor.
   b. Complete the condition with the following expression:

   `AssessorAcknowledgement.Ack.equals("YES")!=true`

   Java has its quirks. You must test the value of the object not that it is the *same* object.

10.6 Implementing the Claim handler staff activity

The final step in developing the flow is to implement the Staff Activity.

The staff activity is used to represent the point in the process at which the Claim handler involvement is required to proceed. The staff activity is treated as a special form of the invoke activity, however its signature does not refer to a partner link. The staff activity is defined by an operation that has associated input and output messages. This operation and these messages are defined in a WSDL file, typically on the interface to the process. The process is then defined to have two variables with message types that match the input and output for the operation associated with the staff activity.

Exploring ManualSelectAssessor's staff properties

We shall modify the staff activity generated by the Modeler. For now, we just explore some of the special characteristics of a staff activity.

Implementation

Click the **ManualSelectAssessor** activity and open the **implementation** tab in the property explorer. You can see it is associated with the ManualSelectAssessorPT PortType in the RequestExternalReportsInterface.wsdl file generated by the Modeler. We shall be changing the operations and messages used by the ManualSelectAssessor staff activity illustrated in Figure 10-42.
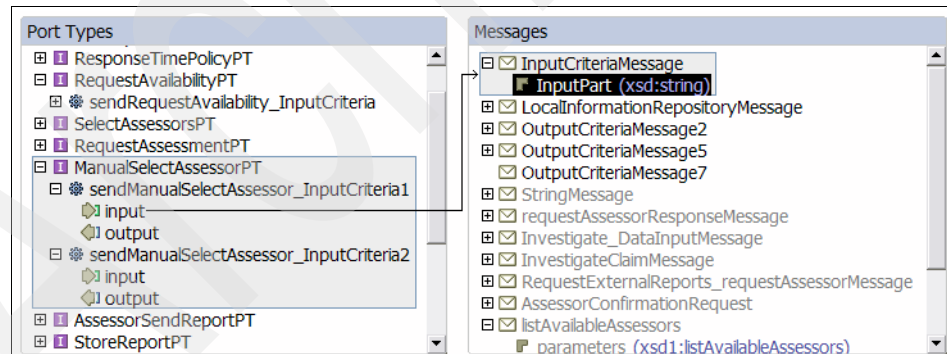
Figure 10-42   ManualSelectAssessor Port types and Messages generated by Modeler

The messages and operations will drive the information that is exchanged with the Web or portal client to be used by the Claim handler.

Web Clients

When the work item is claimed by the Claim handler, the request variable and its contents are passed to the Claim handler using a Web or portal client. The Claim handler decides which assessor will be asked to process the claim, and the Web client returns the assessor ID in the response variable.

The most likely method of interaction between a Claim handler and his or her work items will be through a Process Web Client. A supplied application enables an authorized Claim handler to log in and then interact with the RequestExternalReports claims process. The Claim handler can take one of the instances of the RequestExternalReports processes. The instance is then unavailable to any other Claim handlers. The Claim handler has responsibility to complete the staff activity by providing the assessorID to pass back to the process instance to allow execution to continue.

There is a choice of Web clients that can be written or configured to suit business needs. A fuller explanation is provided in section 9.2 of the Redbook, *"WebSphere Business Integration Server Foundation 5.1 Handbook"*, SG24-6318.

Open the **Client** tab of the ManualSelectAssessor properties page (Figure 10-43). These definitions for the default Web and Portal clients are extracted from the ClientSet.xml file that is in the `...\install_root\IBM\ WebSphere Studio\Application Developer IE\v5.1.1\runtimes\ee_v51\ ProcessChoreographer\client` directory.

Figure 10-43   Web client definition tab
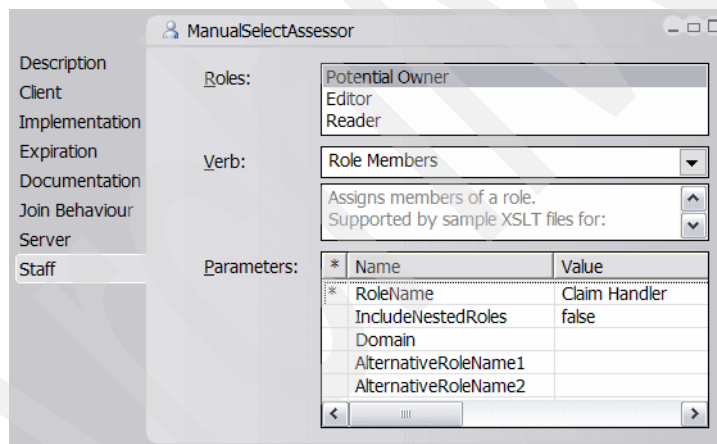
We can modify the ClientSet.xml file to define a different client interface, or we can modify the value for each of the JSPs to provide an alternative implementation for the default interfaces. Details how to do this are provided in the referenced Redbook. We shall use the default web client using customized input and output messages.

Staff roles

During development, the persons who are allowed to perform the staff activities are defined abstractly with roles. At this point, although the developer indicates what kind of person can perform the activity, they do not know about the technology that will be used to identify the person who will perform the role, or how to handle the authentication of the person.

Open the Staff tab of the ManualSelectAssessor activity and you will see that the Claim handler is already defined in the BPEL imported from Modeler. See Figure 10-44.

To bypass dealing with security configuration during test, change the Staff verb to Everybody.

Figure 10-44   Staff role assignment for ManualSelectAssessor

At runtime, when the staff activity is reached, the WebSphere Process Choreographer engine creates a work item that can be processed by any Claim handlers. The WebSphere Process Choreographer container maps the Claim handler role to a real query against the identity manager implementation, such as the Windows User Registry, or an LDAP directory service. This mechanism allows the runtime to determine who is a potential owner and then enforce its authentication policy.

Because the mapping of the Claim handler role is performed at run time, the mapping definition can be left to the description of how to deploy the process to WebSphere Business Integration Server Foundation.

Expiration

The staff activity is very like an Invoke activity, and, like an invoke activity, it has expiration settings if the Claim handler fails to respond. There are various ways to set the expiration time. When the expiration occurs, the staff activity state is set to STATE_EXPIRED. This state value can then be checked in the Control Links from the output terminal of the Staff activity. For one Control link, you can check for this state and return a value of `true` if the staff activity's state is STATE_EXPIRED. This will cause execution to proceed down that Control Link.

Commonly, specification of what to do if a Claim handler takes ownership of a claim, and then fails to perform the activity, are not made clear in the business process model provided by a business analyst. The IT specialist should consult the business analyst and the IT architect rather than make an ad hoc decision.

In real life, if the case is not thought to pose a business issue, then choosing an appropriate expiration time, such as a day, and then iterating the while loop would be a reasonable solution. We won't code any expiration in this scenario.

Configuring ManualSelectAssessor

The configuration of ManualSelectAssessor will require us to:

1. Modify the Port type generated by the Modeler to configure input and output messages for ManualSelectAssessor.

2. Reconfigure the ManualSelectAssessor activity to use the new Port type.

3. Rewire the detailed logic of the while no committed assessment while activity.

Modify the Port type and operations for ManualSelectAssessor

The interface to the ManualSelectAssessor activity we will use is:

► Input: The While activity condition

   This contains a string with either the information no assessor has been selected, or that the selected assessor has declined to perform the assessment.

► Output: The assessor ID of the assessor chosen by the Claim handler to perform the assessment

   This needs to be a new message, a xsd:int. The default Web client will construct a form from the parts of the output message and perform conversions for types defined in the JAX-RPC specification.

1. Open the RequestExternalReportsInterface.wsdl file as illustrated in Figure 10-42 on page 429. See Figure 10-45.

    a. **Delete** the two existing operations for ManualSelectAssessorPT.

    b. **Add Child** → **Operation** → call it `requestAssessorID` → **Add Child** → **Input** → **Add Child** → **Output.**

    c. Select **Input** → **Create a new message** → and call it `ManuallyChosenAssessorInput`.

    d. Select **Output** → **Specify Message** → **Create a new message** → and call it `ManuallyChosenAssessorOutput`.

    e. Select the new message **ManuallyChosenAssessorInput** → **Add child** → **Part** and name the new part `AssessorSelectionState` and leave it with the default type `xsd:string`.

    f. Select the new message **ManuallyChosenAssessorInput** again → **Add child** → **Part** and name the new part `claimID` and give it the type `xsd:int`.

    g. Select the new message **ManuallyChosenAssessorOutput** → **Add child** → **Part** and name the new part `AssessorID` and give it the type `xsd:int`.

Figure 10-45   Input and Output messages for the ManualSelectAssessor activity

Reconfigure the ManualSelectAssessor activity

To reconfigure the ManualSelectAssessor activity, follow these steps:

1. On the **implementation** tab of the ManualSelectAssessor activity properties, ensure the Request and Response variables are set to **ManualInputCriteria** Variable and **ManualOutputCriteriaVariable,** which should already exist. If they do not, create them.

2. Type the Request and Response messages by selecting each variable, and opening the Message tab of its property editor.

    a. For the manualInputCriteriaVariable, scroll down through the Message spin box to select **ManuallyChosenAssessorInput.** You might need to **Browse...** to the RequestExternalReportsInput.wsdl file if the messages do not appear. The .WSDL file is cached.

    b. For the ManualOutputCriteriaVariable, scroll down through the Message spin box to select **ManuallyChosenAssessorOutput.**

3. Back in the ManualSelectAssessor properties box, select the newly created operation, **requestAssessorID**, for the **Operation** property on the **Implementation** tab.

Transformations to and from the manual activity

The next steps manage the data in and out of ManualSelectAssessor. The flow ends up looking like Figure 10-46.
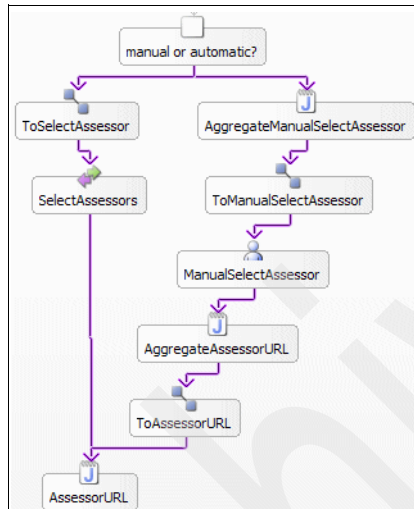
Figure 10-46 ManualSelectAssessorFlow

Create an Aggregation and Transform into ManualSelectAssessor

Follow these steps:

1. Create a new Transform activity and call it `ToManualSelectAssessor`.

2. The Response variable will be `ManualInputCriteriaVariable`.

3. Create an aggregation called `AggregateManualSelectAssessor` that will combine the InputCriterionVariable, and the AssessorAcknowledgement variable.

4. Create a new Transformer service called `TransformerManualSelect|Assessor`:

   a. **Add** the AssessorAcknowledgement message from the RequestExternalReportsInterface.wsdl file and the RequestExternalReportsRequest Message from the ExternalClaimsAssessorsInterface.wsdl file to the Transform input messages.

b. **Select** the ManuallyChosenAssessorInput message from the RequestExternalReportsInterface.wsdl file as the output message.

c. Wire up the claimID and the Ack parts as in Figure 10-47.

Output Message	Input Message
☐ message ManuallyChosenAssessorInput	☐ message AssessorAcknowledgement, message RequestExternalReports_requestAssessorMessage
┣ part ClaimID : int	claimID : xsd:int
┗ part AssessorSelectionState : string	part Ack : string

Figure 10-47   Input transformation for ManuallyChosenAssessorInput

Create an Aggregation and Transform out of ManualSelectAssessor

Follow these steps:

1. Build another aggregating transformer starting with creating a new transformer activity called `ToAssessorURL`, an aggregation called `AggregateAssessorUR1` and a transformer service called `TransformerAssessorURL`.

   a. Combine the AssessorID from the ManuallyChosenAssessorOutput message in RequestExternalReportsInterface.wsdl file with the ClaimID from the RequestExternalReportsRequest message in the ExternalClaimsAssessorsInterface.wsdl file as the input messages. See Figure 10-48.

   b. Select the **SelectAssessorResponse** message from the PreferredAssessor(5).wsdl file as the output message.

Output Message	Input Message
☐ message selectAssessorResponse	☐ message ManuallyChosenAssessorOutput, message RequestExternalReports_requestAssessorMes
┣ part parameters : selectAssessorResponse	message ManuallyChosenAssessorOutput, claimID : xsd:int
┗ selectAssessorReturn : impl:SelectedAssessor	message ManuallyChosenAssessorOutput, claimID : xsd:int
assessorID : int	part AssessorID : int
claimID : int	claimID : xsd:int

Figure 10-48   Output transformations for ManuallyChosenAsssessorOutput

Rewire the While activity

Important: Be careful to preserve the link with the condition filter set that goes to ManualSelectAssessor.

Drop the two transform services into the While activity and rewire as shown in Figure 10-46 on page 434.

10.7 Build

At this point and with process saved, we should have only one remaining warning message, `The deployment code for this process needs to be generated.`

BPEL process needs deployment code to be executed on the server. We can generate deploy code for the process in WebSphere Studio Application Developer Integration Edition after all errors in the process have been fixed. WebSphere Studio Application Development Integration Edition generates an EAR module which includes EJB module created from definitions of the business process. We can execute our business process by deploying this EAR module to the server.

There is one additional build step required after completing testing on the test server to ready the process for deployment to use DB/2 on the production server. (see 10.7.2, "Building for a production server" on page 438). We use Cloudscape™ for persisting objects on the test server, and DB/2 as the database on the production server.

10.7.1 Building the business process

To build the business process, follow these steps:

1. Ensure the business process has no errors by saving all the workspace.

2. We do not want the process-instance to be deleted automatically when it finishes, to give ourselves a chance to see the results. The default option is to delete the instance when a process instance completes. To change this, go to the RequestExternalReports activity bubble at the top of the flow, and change its server properties as in Figure 10-49.
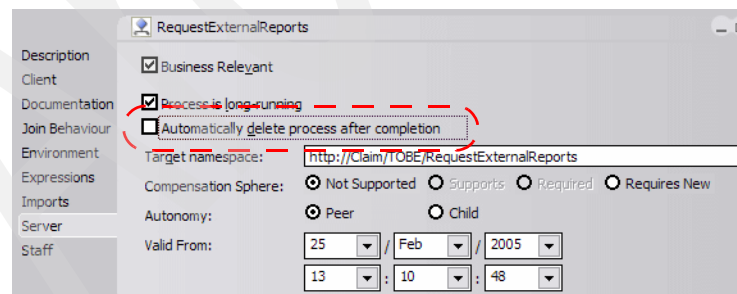
Figure 10-49   Retain process instance after completion

3. To generate deploy code, right-click **RequestExternalReports** process and select **Enterprise Services** → **Generate Deploy Code**. The window for setting code generation opens (Figure 10-50).
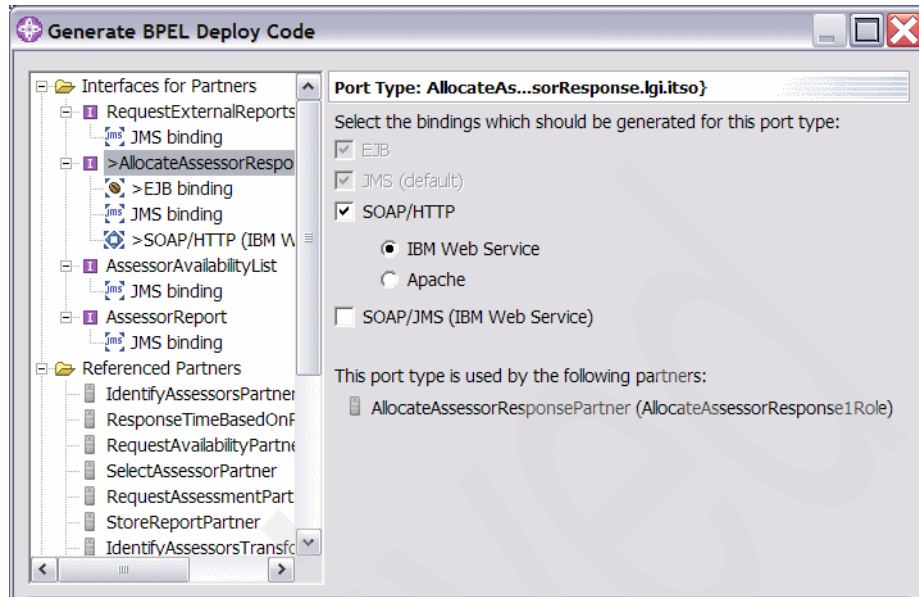
Figure 10-50   Generate BPEL deploy code

4. Select transport bindings between the process and its partners for when the process is acting as the server.

   a. Leave the protocol to invoke the process as JMS.

   b. For the other three interfaces, define the bindings as SOAP/Http using the IBM Web service as shown in Figure 10-50.

   c. For each of the three interfaces, define the SOAP style as show in Figure 10-51 on page 437. The WSDLs were created to be WS-I compliant and to use the Document Literal interface. For more information about SOAP and WS-I compliance refer to the redbook *"WebSphere and .Net interoperability using Web services"*, SG24-6395.
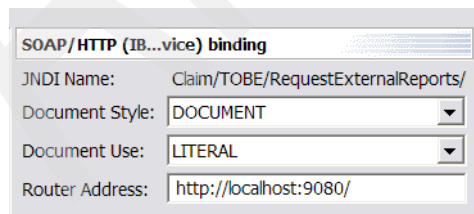
Figure 10-51   SOAP Binding style

5. Check the referenced partners. You should not need to change anything. Press **OK** to start the generation. It will take several minutes. When the generation completes, there are 15 warning messages about compensation objects that are unresolvable that should be ignored.

6. Find a new EAR module with the name **ITSOLGIEAR** (service project name + EAR) in the J2EE Hierarchy view. This EAR module includes the Web module (ITSOLGIWeb) and EJB module (ITSOLGIEJB).

> **Tip:** When saving a project interchange file, do not save the deployable services, but regenerate them after restoring the service project.

10.7.2 Building for a production server

To build for a production server, we can simplify the configuration on the production server by removing the reference to the Cloudscape database used in the test environment. That way, when then process .ear file is deployed to the production server, the .ear file will be configured to use the database defined on the production server automatically.

1. Select the J2EE hierarchy view and delete the map and schema from the Cloudscape resource. See Figure 10-52 on page 438.
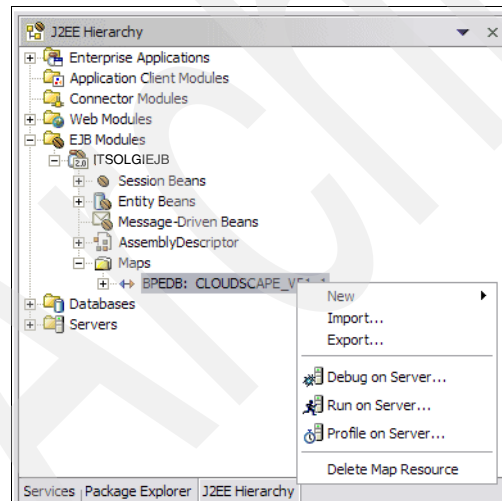
Figure 10-52   Deleting Cloudscape mappings

2. Export the ITSOLGI .ear file.

10.8 Test and debug the process

WebSphere Studio Application Development Integration Edition provides a test server environment which has the same server component as WebSphere Business Integration Server Foundation. This means you can test business processes without installation and configuration of a test server outside of your developer environment. It also provides functions for debugging such as setting break points, executing business processes step by step and monitoring values of data during execution.

With these functions, you can find bugs which are not expected in developing phase and can fix them before deploy the business process to actual runtime environment.

10.8.1 Prepare to test

Follow these steps to set up for testing.

1. Create new test server. Open the server configuration view. Look for it in the bottom of the service perspective or open it from **Window** → **Show view** → **Server Configuration** in menu bar. Right-click and select **New** → **Server and Server Configuration**. See Figure 10-53 on page 439.
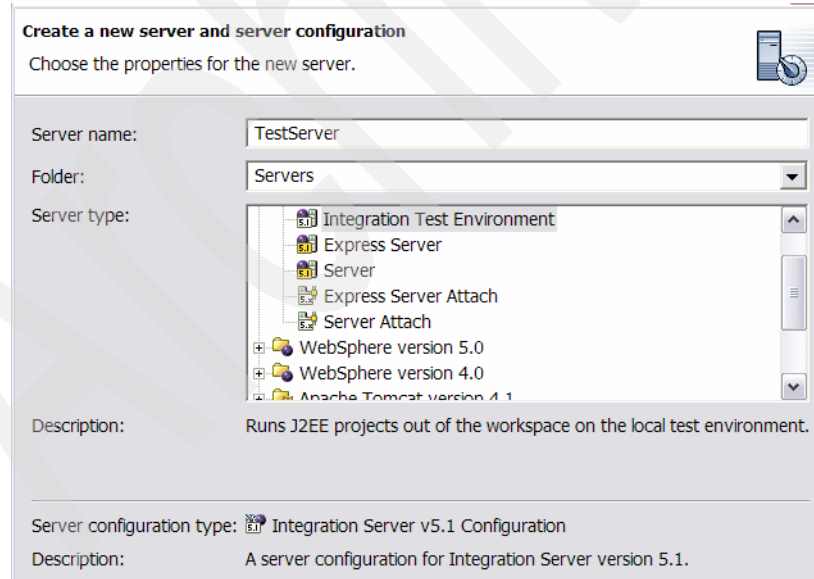
Figure 10-53   Create server and server configuration

2. Type in `TestServer` as a new test server name and make sure that **Integration Test Environment** is selected in server type list. Click **OK** to create a new server.

3. The new TestServer is added under servers folder in server configuration view. Right-click **TestServer** and select **Add and remove projects.**

4. Click the **Add** button (Figure 10-54) the ITSOLGIEAR EAR module we have just created. Click **Finish** to close the window.
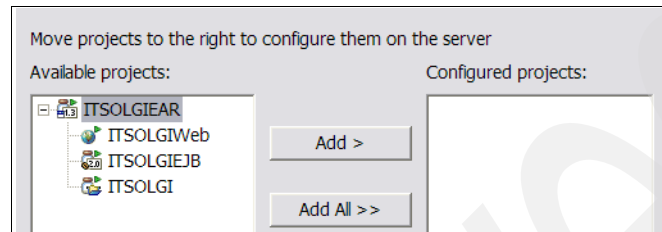
Figure 10-54   Adding ITSOLGIEAR to TestServer

5. Right-click **TestServer** again and select **Create tables and data sources**. We need a database to execute RequestExternalReports process because it is a long-running process and needs to store status of the process instances in database.
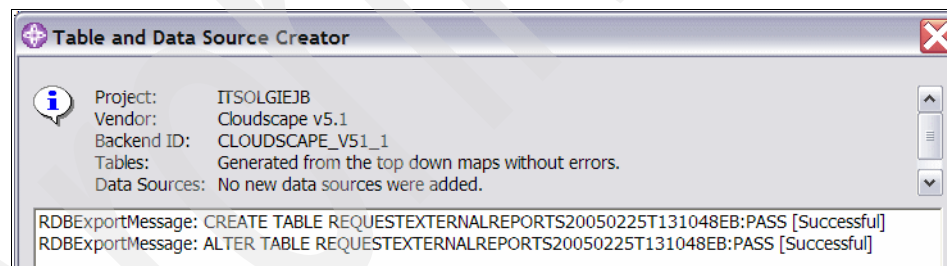
Figure 10-55   Confirmation window of creating database table

Check the messages in the confirmation window to see that database table and data source creation are successfully completed.

10.8.2 Publishing the business process to the test server

To publish to the test server, follow these steps:

1. Right-click **TestServer** → **Publish.**

   a. Quite possibly on publishing the service, it will fail with one of the following staffing errors in Example 10-8 or Example 10-9.

Example 10-8   TestServer publish error 1

```
Project ITSOLGIEJB deployment failed.
   BPEA0010E: Unexpected exception during execution.
java.lang.reflect.InvocationTargetException:
com.ibm.bpe.api.UnexpectedFailureException: BPEA0010E: Unexpected exception
during execution.
com.ibm.bpe.util.ProcessAssertionError: Assertion violation !(paramValue !=
null && paramValue.length() != 0) in method >>at
com.ibm.bpe.staff.StaffPluginUtil.deployStaffVerb(StaffP
```

    i. This first error is cause by not implementing the role Claim handler . For this workshop, the workaround is to define the staff verb in the ManualSelectAssessor activity as **Everybody**.

Example 10-9   Test Server publish error 2

```
Project ITSOWorkshopEJB deployment failed.
   BPED0203I: Validated process model 'RequestExternalReports' with
findings ( 0 information, 0 warnings, 1 errors ):
BPED0267E: Syntactical error found in BPEL file
'Claim/TOBE/RequestExternalReports/RequestExternalReports.bpel' (row: 223,
column: 73). Detail message: cvc-complex-type.2.4.b: The content of element
'wpc:webClientSettings' is not complete. One of
'("http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/":customSe
tting,
"http://www.ibm.com/xmlns/prod/websphere/business-process/v5.1/":jsp)' is
expected.
java.lang.reflect.InvocationTargetException:
com.ibm.bpe.plugins.DeploymentBPELProcessValidationException: BPED0203I:
Validated process model 'RequestExternalReports' with findings ( 0
information, 0 warnings, 1 errors ):
BPED0267E: Syntactical error found in BPEL file
'Claim/TOBE/RequestExternalReports/RequestExternalReports.bpel' (row: 223,
column: 73). Detail message: cvc-complex-type.2.4.b: The content of element
'wpc:webClientSettings' is not complete. One of
```

    ii. If you get this second error, remove the line `<wpc:webClientSettings clientType="Web Client"/>` from the RequestExternalReports.bpel file.

10.8.3 Creating the test environment

For the unit test, we will use all the components downstream of the process server, in other words, all the components except workflow.

There is a choice of two assessor systems to use:

- ▶ One runs in WebSphere Application Server.

► The other is scaffolded in the broker.

Which one to use depends on which version of the assessor management system is installed on WebSphere Application Server. The table of assessors and their URLs is hard coded into the assessor management system. One version points at the WebSphere Application Server and the other at WebSphere Business Integration Message Broker.

Selecting an assessor system

To select a system, follow these steps:

1. Check which system is currently installed:

   a. Open a browser and open the WebSphere Application Server administrative console on SAH414A by connecting to `http://sah414a:9090/admin/`.

   b. Open the installed applications → place a check against **AssessorManagementService** → **Export** (see Figure 10-56 on page 442).
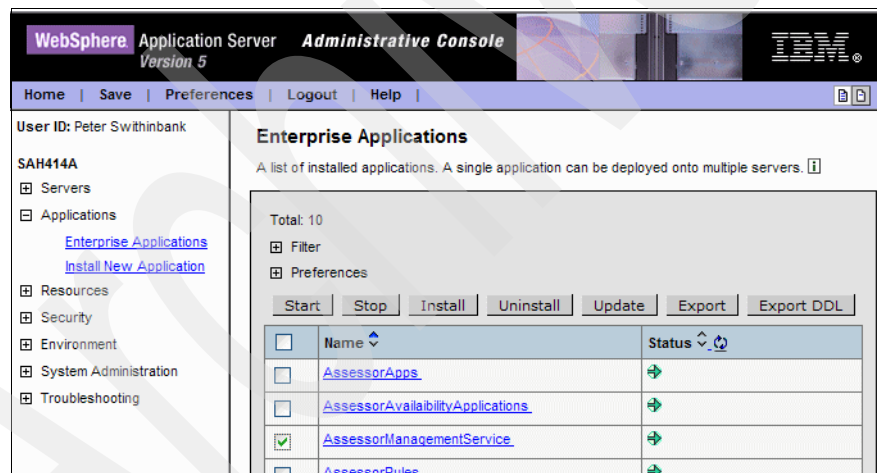
Figure 10-56   WebSphere Application Server admin console

   c. Download the AssessorManagementService.ear file.

   d. You can either browse the .ear file using an unzip utility like WinZip, or you can import it into WebSphere Studio Application Development Integration Edition as a new .ear project, which we describe here.

   i. Click **File** → **Import** → **EAR file** → **Next** → browse to the downloaded `AssessorManagementService.ear file` → **Finish**.

ii. Open the J2EE perspective and the J2EE Hierarchy view → **EJB Modules** → **AssessorManagementServiceEJB** → **Session Beans** → **AssessorManagement** → double-click **AssessorManagementBean** to open it → Open the **Outline** view in the window underneath → double-click the remote interface **requestListAssessors** and examine the code in Example 10-10 on page 443.

Example 10-10   AssessorManagementSystem: requestListAssessors Web service

```
//Create Array of hard coded assessors and create new AssessorList to return
Assessor stubAssessor = new Assessor();
stubAssessor.setAssessorID( new Integer(9999));
stubAssessor.setAssessorURL( "http://SAH414A:7080/Availability");
Assessor anotherStubAssessor = new Assessor();
// We've only got one assessor, just have two entries with the same destination
anotherStubAssessor.setAssessorID(new Integer(5555));
anotherStubAssessor.setAssessorURL("http://SAH414A:7080/Availability");
Assessor [] assArray = new Assessor[2];
assArray[0] = stubAssessor;
assArray[1] = anotherStubAssessor;
AssessorList stubAssList = new AssessorList();
stubAssList.setAssessors( assArray);
stubAssList.setClaimId( claimID);
```

You can see the deployed .ear file will use the assessor implemented in the broker.

e. To use the alternative assessor (Example 10-11[4]), browse to .\SG24-6636\WAS\Flow2 and deploy the Flow2_AssessorManagementService_SOURCE.ear file to WebSphere Application Server. The broker version is in the Broker Assessor subdirectory.

Example 10-11   AssessorManagementSystem: snippet of alternative assessor code

```
stubAssessor.setAssessorURL(
"http://SAH414A:9080/AssessorAvailaibilityApplicationsEJBRouter/services/Availa
bility");
Assessor anotherStubAssessor = new Assessor();
// We've only got one assessor, just have two entries with the same destination
anotherStubAssessor.setAssessorID(new Integer(5555));
anotherStubAssessor.setAssessorURL("http://SAH414A:9080/AssessorAvailaibilityAp
plicationsEJBRouter/services/Availability");
```

---

[4] Apologies for the typo in the URL - the misspelling is "correct".

> **Restriction:** Currently the asynchronous responses from the assessor system have not been implemented for the WebSphere Application Server assessor. The acknowledgement and report responses need to be sent manually. For this reason we preferred to use the broker implementation of the assessor. WebSphere MQ is used to connect the steps in the assessors response, and by disabling and enabling get on the assessor queues Flow7A and FLOW8, as described in 9.9.2, "Scaffolded Assessor and Claim system" on page 370, the acknowledgement and report responses can be delayed.

Calling choreographer services

As you recall, the proxyAssessorSystem calls three one-way SOAP services hosted by the RequestExternalReports flow. How do we know what URL to code into the proxyAssessorSystem HTTPRequest nodes in the broker for these services?

In the choreographer's generate deployment code step in "Building the business process" on page 436, you provided the router address of each of the three services "`http://localhost:9080/`". Is this the correct path to code into the ESB for the Http Request nodes that respond to the Process Choreographer? It turns out it is not.

The Web service URL you need to configure in the broker to call the availability response server in the choreographer needs to be coded as

```
http://SAH414:9080/ITSOLGIWeb/services/RequestExternalReportsAssessorAvailabili
tyListHTTPServicePort
```

Where does this path come from?

1. In WebSphere Studio Application Development Integration Edition open the **ITSOLGIWeb** project in the **Deployable** services folder.

2. Find the **web.xml** (web deployment descriptor) file and open it.

3. You can see three servlets and JSPs defined in the project. These are the services the broker calls.
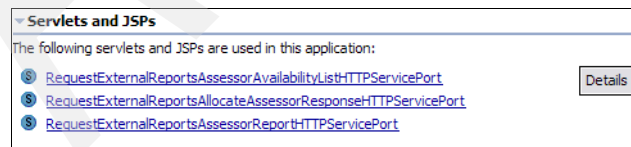
Figure 10-57   Web services deployed in RequestExternalReports.bpel

Click **Details** and examine the URL mappings. The path we require for the AssessorAvailabilitylist service would appear to be:

```
services/RequestExternalReportsAssessorAvailabilityListHTTPServicePort
```

So, add this path to the router address portion of the URL to make a complete path to the Web service, and the equivalent for the other two services.

```
ITSOLGIWeb/services/RequestExternalReportsAssessorAvailabilityListHTTPServi
cePort
```

4. We need to change the URLs in the broker because they do not match the addresses of the Web services in the new process you have created in the workshop. You can do this without editing the flow in message broker directly. This is a *one-shot* modification. When you redeploy again, the parameter value in the flow will be restored. So, you are better advised to change the parameter in the flow.

  a. Open the administrative perspective in broker → find the Deployables folder in the Broker Archives directory and open **LGIAvailability**.

  b. Click the HTTP Request node in Output3a and replace the Web service URL with the RequestExternalReportsAssessorAvailabilityListHTTPService URL from ITSOWorkshopWeb.

  ```
  http://SAH414:9080/services/RequestExternalReportsAssessorAvailabilityLi
  stHTTPServicePort
  ```

  c. Repeat the procedure for the other two Web services.

  d. Redeploy LGIReport ad LGIAvailability.

  e. Turn Normal tracing on again for both execution groups.

10.8.4 Check that downstream components are operational

Before starting to test the Process Choreographer, invoke all the required services running on WebSphere Application Server and WebSphere Business Integration Message Broker are operational using the Web services explorer tool in WebSphere Studio Application Development Integration Edition.

10.8.5 Testing and debugging the business process

To test the business process, follow these steps:

1. Right-click **TestServer** → **Start**. The server will be started. Check that `message server1 is opened for e-business` is displayed in console view.

  a. Here are a couple of common problems for this stage and their fixes:

- The most common cause of a server starting with errors is path lengths being longer than the Windows maximum. A quick solution is to remap the directory containing your workspace using the Windows SUBST command.

- Another common cause of problems, for example syntax problems in Application.xml being reported, is a wrongly applied service.

  Check that you can build, deploy, publish, start and browse the simpleProcess documented in the infocenter.

2. Open **Servers** view and select **Launch Business Process Web client** form menu. A browser window opens.

3. Select **my template** from the menu list on the left side of the window. You can see RequestExternalReports process is ready. see Figure 10-58.
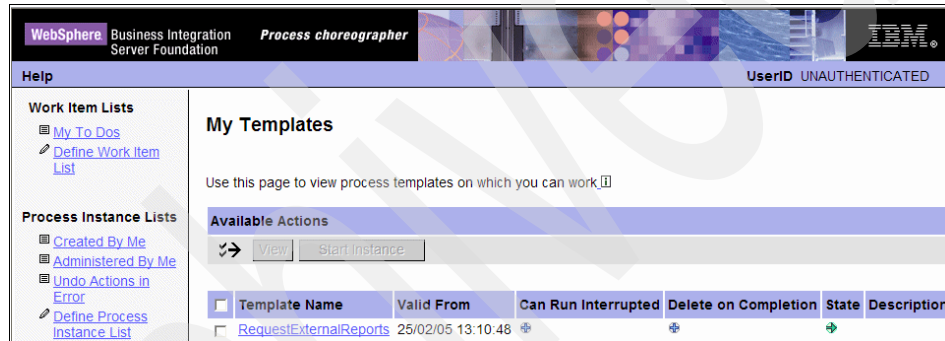
Figure 10-58   Launched Process Choreographer client with My Templates selected

4. Click the check box adjacent to **RequestExternalReport**s → **Start instance.** A panel showing the input message is displayed (Figure 10-59).

Figure 10-59   Input message to process

5. Fill in the details. The only requirement is the input data matches the types. Use a date of the format `mm-dd-yyyy` in the date string field.

6. Press **Start instance** to start process.

7. On the Created by Me panel, select the process just started and check it → **Monitor.**

8. If all is well, the process shows a whole list of finished tasks and waits for input. See Figure 10-60 for an example where the process is waiting for the acknowledgement from the selected assessor.

Figure 10-60   Waiting for a response from the proxyAssessorSystem.

9. Because we are using the broker to simulate assessors, the WebSphere MQ explorer will have a message each on the FLOW7A and FLOW8 queues which are get inhibited to simulate a time delay. Releasing the messages in the right order allows the process to continue to completion.

Debugging the process

If the process does not work as expected, there are a number of debugging options:

▶ Checking the execution of process instance step by step
▶ Adding break points
▶ Observing the values of data in each of the steps

To test a process in debug mode, follow these steps.

1. Add a break point to check the execution steps. Right-click an activity in the BPEL editor and select **Add Entry/Exit break point**.

2. Start the TestServer in debug mode. Right-click **TestServer** (Figure 10-61) and select **Debug**. The debug perspective opens.

3. Execute the process step by step by clicking the **step over** or **step in** button in debug view. Variables are shown in the **variable view** on the right side of the perspective.
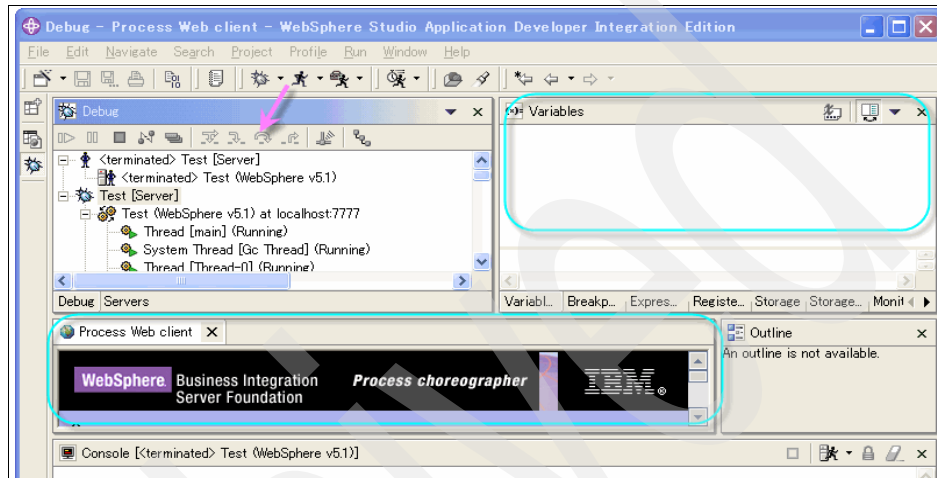
Figure 10-61   Debug perspective

As an example, Figure 10-62 on page 449 shows the process suspended at the entry to the java snippet to aggregate the results of querying which assessors are potential candidates for sending for assessment, and what the customer's ResponseTimePolicy is.
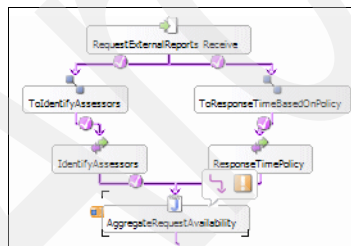
Figure 10-62   Process suspended at the start of the java snippet

The debugging process to verify the data going into the next transformation mapping would take the following steps.

Make sure the debug view is showing in the top left window and the current instance selected. There will be a list of variables in the top right window.

1. Open the **IdentifyAssessorsOutputCriteriaVariable** and verify that the AssessorManagement service has returned a list of assessors.

2. Step into the Java Snippet, the snippet implementation tab opens.

3. Step over each statement.

4. On exit from the snippet, check the AggregateRequestAvailability variable has its parts set up correctly, especially the list of assessors.

5. Step over the ToRequestAvailability transformer service. There is no way to debug the stylesheet implementation.

6. Now check the RequestAvailabilityInputCriteriaVariable, is the assessorList set?

   Perhaps in this case the assessor list was not set, so the cause of the error has been identified as being in the proxyAssessorSystem :ESB.

10.9 Deploy the process to the server

Business processes which have been developed in WebSphere Studio Application Development Integration Edition are deployed to WebSphere Business Integration Server Foundation in the format of .ear file and executed in the business process container provided by WebSphere Business Integration Server Foundation.

WebSphere Business Integration Server Foundation provides Java 2 Enterprise Edition based process engine which supports following features:

► Multi-style Process Support

   Non-interruptible (1-transactional) and interruptible (multi-transactional) Processes are supported.

► Compensation Support

   Runtime components that support compensation (undo of committed work) for processes.

► Human Interaction Support

   Runtime components that allow people to interact with processes, such as with a Web Browser based user interface to present work items and processes.

WebSphere Business Integration Server Foundation has several components in its process engine.

► Process navigation manages process instances and their status.

- ► Invocation of web services and request for execution of processes is done via both an external and internal interface.
- ► Staff activities are managed by the people interaction component.
- ► All the components of the business process engine are running on WebSphere Application Server and use database and message queuing services.
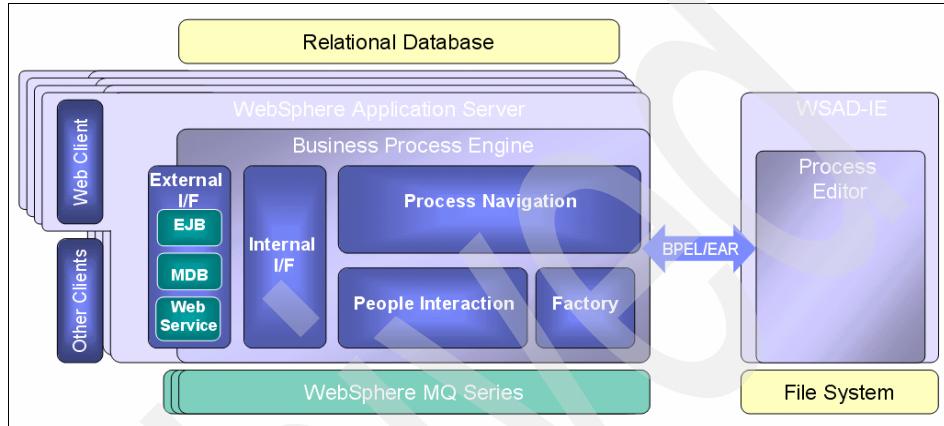
Figure 10-63   Components of WebSphere Business Integration Server Foundation

10.9.1 Installation of business process application

1. Start the Admin console browser (at `http://SAH414B:9080/Admin`) and go to **Applications** → **Enterprise Applications** and click **Install.**

2. Browse to the ITSOLGIEAR.ear file → **Next** → **Next.**

3. On **Step 1** of **Install New Application** specify a short directory name on the root of your chosen drive (D: in our case) to install the application (create the directory before proceeding). Check the **Deploy EJBs** check box.

4. Jump to step 11 and tick the **Enable** box under **Create tables.**

   If there is a choice of databases on the server you will be prompted to choose one, otherwise, as in this case, it will select the installed database and continue.

5. Finish the configuration, confirm the application is installed successfully, and save it to the master configuration database.

6. Switch to the Enterprise Applications panel and start the application. It is now ready to run.

10.9.2 Verify the application

1. Start the business process container web client by providing a browser with the URL `http://sah414b:9080/bpe/webclient`.

2. Reconfigure the broker to call the receive activities in the production server, hosted on SAH414B. Do this without modifying the message flows from the Administrative perspective in the broker toolkit. For example the Output6a flow in Figure 10-64:
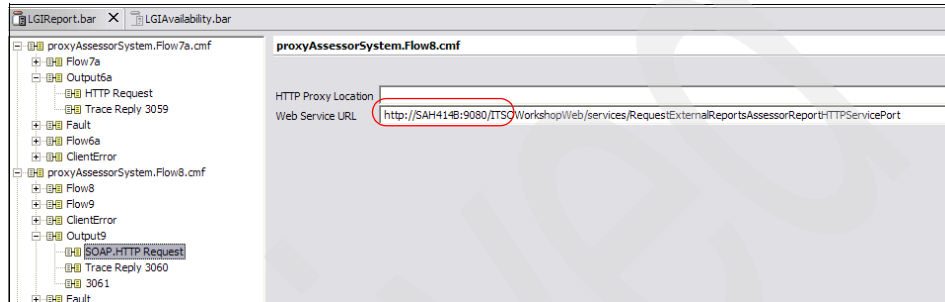
Figure 10-64   Changing the hostname to SAH414B in the Output6a flow

3. Run the validation test. If you want to test the staff activity, modify the response from the assessor acknowledgement to `N0`.

10.10 Summary

This chapter has described how the IT process specialist modifies the business process provided by the business analyst to implement the interfaces provided by the IT architect and then tests and deploys the process to WebSphere Business Integration Server Foundation.

The final step in the solution is to integrate the claims workflow running on the WebSphere MQ Workflow server with the business process using the WA0D integration supportpac.

11

Modify the Claim Investigation process

In this chapter we configure the claim investigation process to call the new automated RequestExternalReports activity instead of the current manual activity.

453

11.1 WebSphere MQ Workflow: long-running processes

WebSphere MQ Workflow contains the necessary software to define, build, and manage business processes. The WebSphere MQ Workflow engine enabled LGI to deploy its policy and claims handling business processes spanning multiple computer systems and also integrating the DCI and LGI IT systems.

Users interact with WebSphere MQ Workflow using Web applications, WebSphere Portal Server portlets, or Microsoft Windows standalone clients. With these user interfaces, Claim handlers can pick up work items as required by the business process, deal with them, and then tell the workflow engine that the work item is finished. The process engine keeps track of all the work items in the system so that ongoing claims can be monitored and the status of claims can be checked. Claims can be rescheduled and reassigned.

To integrate internal and external business systems with WebSphere MQ Workflow, processes can access applications and Web services using WebSphere MQ Workflow execution agents which are configured as part of the workflow. Workitems are either handed by the Claim handlers or by the automated execution agents calling on applications or Web services to process a workitem.

Development tools

A graphical representation of a WebSphere MQ Workflow process can be generated by using WebSphere MQ Workflow Build Time. Alternatively, processes that were designed in WebSphere Business Integration Modeler can be imported into WebSphere MQ Workflow as a Flow Definition Language file [FDL].

Runtime environment

The WebSphere MQ Workflow Runtime is the container in which the deployed processes run. WebSphere MQ Workflow Runtime uses a database management system to keep track of the processes' internal state. WebSphere MQ Workflow uses WebSphere MQ queues to communicate with external actors. WebSphere Application Server is used for providing a graphical user interface to end users who participate in the workflow.

Monitoring and management

The WebSphere MQ Workflow administrator uses the Administration Utility to:

▶ Start and stop servers.
▶ Monitor and analyze error logs.

Additionally, WebSphere Business Integration Monitor V 4.3.5 can be used to analyze trace data as well as generate reports and graphical representations of the business data that is generated by the processes. This statistical data that is obtained by WebSphere Business Integration Monitor can be used by Modeler to reengineer business processes.

11.2 Process Integration: WebSphere MQ Workflow

Looking at the new RequestExternalReports process from the WebSphere MQ Workflow point of view, we find that the WebSphere MQ Workflow only needs to understand the interface to this new process to be able to invoke it using a services interface. This interface can be defined in three points:

► Process name and location

The WebSphere MQ Workflow needs this information to be able to invoke the process, for the RequestExternalReports process the process is a BPEL based business process that is deployed to WebSphere Business Integration Server Foundation.

► Process input structure

This structure is used by WebSphere MQ Workflow to format the message sent to the process, for the RequestExternalReports process its the input of the process as modeled in WebSphere Business Integration Modeler.

► Process output structure

This structure is used by WebSphere MQ Workflow to understand the message returned from the process, for the RequestExternalReports process it's the output of the process as modeled in WebSphere Business Integration Modeler.

Accordingly, WebSphere MQ Workflow deals with the RequestExternalReports process as an implementation of the corresponding activity in the existing ClaimInvestigation process.

11.2.1 Implementing custom invocations in WebSphere MQ Workflow

Activity implementations are usually started by MQ Workflow by sending an internal invocation request message to a program execution agent (PEA) or program execution server (PES) those are built-in components in MQ Workflow system[1]. They, in turn, invoke the program that was modeled to implement the activity. Using the message-based interface, it is also possible that MQ Workflow

---

[1] For more details about PEA and PES refer to WebSphere MQ Workflow documentation, Concepts and Architecture.

sends that invocation request message in XML format to a user-defined MQSeries queue.

From the point of view of MQ Workflow, the MQSeries application listening on that queue has to invoke the program that is modeled as the implementation of the activity. For doing so, all the necessary information is passed to the MQSeries application by means of an XML message. The MQSeries application must return with an appropriate XML response, if requested by MQ Workflow.

Therefore, such an application is called a *user-defined* program execution server). A user-defined program execution server can be any application the user writes or a program such as MQSeries Integrator, provided it can deal with the MQ Workflow XML message format. A UPES and a program activity to be performed by that UPES are both modeled in the MQ Workflow BuildTime.

A UPES is defined and configured for an MQ Workflow system by modeling it in MQ Workflow BuildTime. Essential attributes are the name, the version[2], and the queue it represents.

The application that is listening to the UPES queue is not managed by MQ Workflow. A system administrator is responsible for administering the application. From an MQ Workflow point of view, the invocation of an activity implementation was successful when the invocation message is successfully put into the UPES queue. The following figure and the illustrates the idea and shows the components of the UPES.

---

[2] The UPES version denotes the MQ Workflow API version that is supported by the UPES ad accordingly what messages to be sent to the corresponding queue and the message formats.
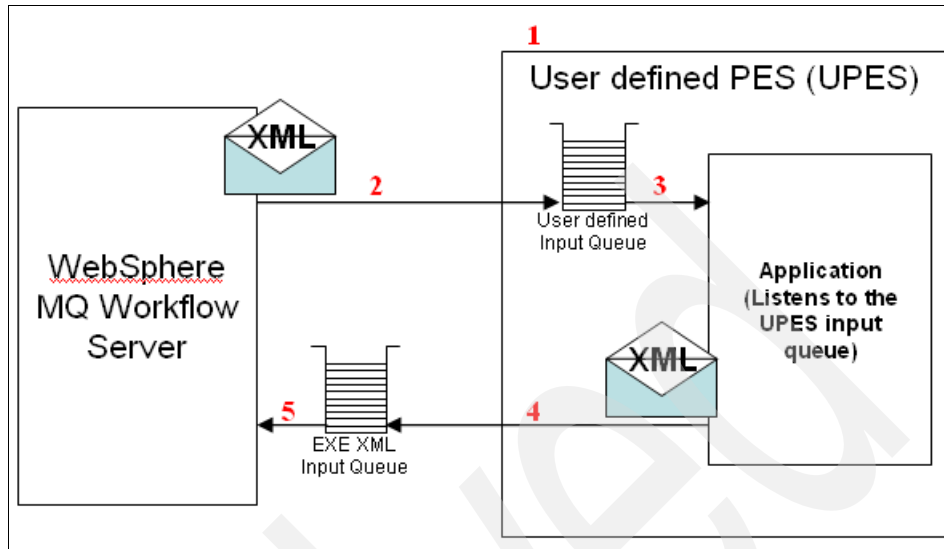
Figure 11-1   Customized activity invocation using UPES

The numbers in the figure illustrate how UPES is architecture:

1. A UPES must have been defined using WebSphere MQ Workflow BuildTime and references an existing WebSphere MQ queue as its input queue. Also, a program needs to be existing that listens to that queue.

2. When an activity implementation is to be started, MQ Workflow sends a program invocation message to the UPES queue in XML format.

3. The application listening to the UPES queue reads the XML message and performs the appropriate action. This action could be invoking the activity implementation, for example, call a program on a platform that is not yet supported by MQ WorkFlow.

4. When the application has finished its action, the application creates a response MQ Workflow XML message, if required, and puts it into the reply queue. Note that the reply queue information is part of the MQMD of the incoming XML invocation message and the default is EXEXMLINPUTQ.

5. MQ Workflow reads the response message, processes it, and changes the state of the activity accordingly.

UPES Invocation modes

While modeling a UPES, two invocation modes for the activity implementation can also be modeled:

► Synchronous invocation (the standard case), where MQ Workflow waits for a completion message containing result data from the UPES before the activity instance is considered to be complete.

► Asynchronous invocation, where no completion message is required and the activity instance is considered to be complete right after the invocation message has been sent. No result data is expected by MQ Workflow and process navigation continues.

From this information. we conclude that UPES is one of the main approaches to implement a custom invocation of applications or processes outside the WebSphere MQ Workflow system.

11.3 Create the ClaimInvestigation_TOBE Workflow

In section, 7.2.4, "Install and configure WebSphere MQ Workflow" on page 227 we built a working WebSphere MQ Workflow system. We now need to modify the existing claim investigation process to automate the RequestExternalReports task running on the workflow system.

The tasks involved are:

1. Recreate the ClaimInvestigation_ASIS process in the WebSphere MQ Workflow buildtime.

2. Create the data structures exchanged with the new RequestExternalReports activity.

3. Define Interface to the RequestExternalReports activity which will call the proxy RequestExternalReports process in WebSphere Business Integration Server Foundation using the JMS compliant XML Enterprise Service interface.

11.3.1 Import the ASIS workflow

To import the ASIS workflow, follow these steps:

1. Open the **WebSphere MQ Workflow buildtime - FMC**

2. Select **Buildtime** → **Import** → find
.\SG24-6636\Workflow\ClaimInvestigation_ASIS.fdl → **OK.** See Figure 11-2 on page 459.
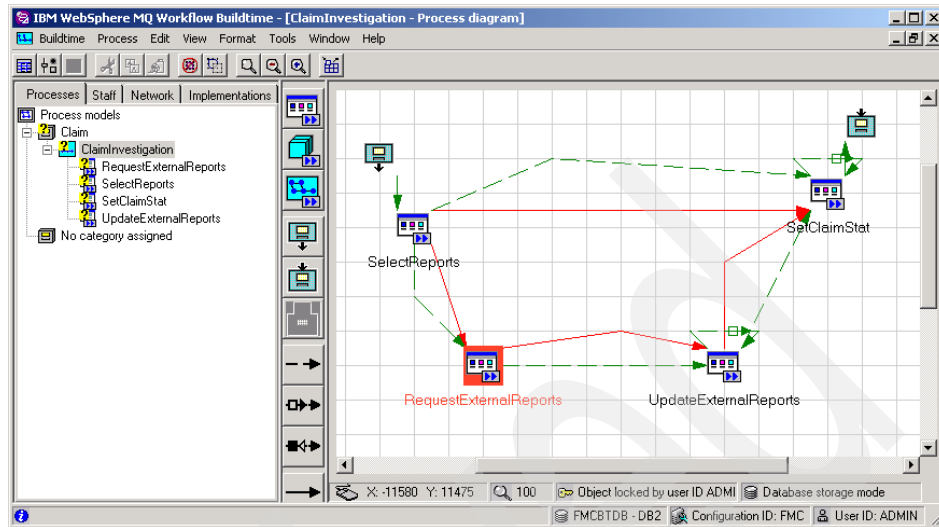
Figure 11-2   ClaimInvestigation_ASIS process

3. To open the ClaimInvestigation process, double-click the top level ClaimInvestigation process.

11.3.2 Create the data structures for RequestExternalReports

There are several ways to do this, but unfortunately no way to import WSDL directly from Rational Software Architect.

1. Model the new data structures directly in WebSphere MQ Workflow.

2. Model the data structures in WebSphere Business Integration Modeler and import as FDL.

3. Create a new XML schema in WebSphere Studio Application Development Integration Edition by massaging the WSDL file and import it as an XML schema into WebSphere Business Integration Modeler and export as an FDL to import into WebSphere MQ Workflow.

In building the scenario, we took the first approach. It was simpler to make no changes to the existing data structures in WebSphere MQ Workflow, and export them as FDL then transform them to WSDL using the FDL2WSDL tool. Later in this section, we create a proxy process as part of the WebSphere MQ Workflow / WebSphere Business Integration Server Foundation integration. In that proxy process, we transform between the data structure used by WebSphere MQ Workflow and the service interface used by the new RequestExternalReports process running in WebSphere Business Integration Server Foundation.

The alternative approach, to start with a WSDL defined by the target service and then modify the FDL used by WebSphere MQ Workflow to conform to the service interface, is more in keeping with the architectural direction we were trying to follow. So for the completeness of the tool-chain we are trying to follow, here is how you would modify WebSphere MQ Workflow to act as a client to an existing service interface.

1. Navigate to the WSDL file which contains the data structures we want to import into WebSphere MQ Workflow. We use the Proxy(1).wsdl file as an example.

2. In the graphical WSDL editor expand the **RequestExternalReports_UPES** Port type and its messages → Select the **requestAssessor** structure.
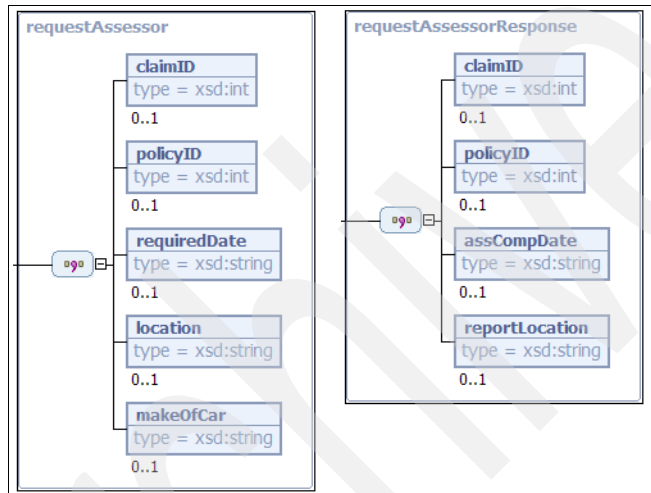
Figure 11-3   Data structures to be converted into schemas

3. Change to the source tab, and the selected structure will be at the focus. Copy the requestAssessor complex type.

4. Paste the type into to the new schema file between the schema tags.

5. Repeat the process for the requestAssessorReponse type.

6. Perform a global replace on xsd: → <blank>; the schema namespace is the default.

7. **Save** the changes; there will be no errors.

8. Set the schema namespace to http://workflow.lgi.itso as in Figure 11-4 on page 461.
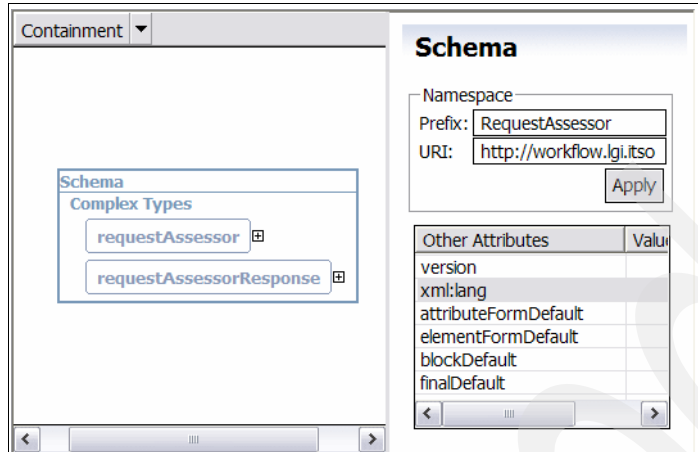
Figure 11-4   RequestExtenalReports schema

9. Export the schema file by selecting the file; right-click → **Export** → **File system** and choosing a target directory.

Convert the schema into FDL

To convert the schema, follow these steps:

1. Open the ITSOLGI project in WebSphere Business Integration Modeler.

2. Click **File** → **Import** → **WebSphere Business Integration Modeler Import** → **XML Schema** → **Next**. Select the **RequestExternalReports.xsd** file and **ITSOLGI** as the target project → **Next** → **Business Item** should be selected → **Finish** → **OK.**

3. Select the **workflow.lgi.itso** folder → **Export** → **WebSphere MQ Workflow FDL.** Select a target directory, the **itso.lgi.workfow** folder → **Export specific items** → **Finish** → **OK.**

Import into WebSphere MQ Workflow

To import the schema, follow these steps:

1. Select the **Implementations** tab in WebSphere MQ Workflow buildtime.

2. Click **Buildtime** → **Import** ... → select the **ITSOLGI.fdl** file that has been exported from Modeler.

3. Remove the **w_** from the data structure names by opening the properties of the two data structures and changing the **Name**

11.3.3 Define the interface to RequestExternalReports

We now need to define the interface in the claim investigation workflow we will use to call the RequestExternalReports process in WebSphere Business Integration Server Foundation.

From the perspective of WebSphere MQ Workflow, RequestExternalReports is a simple automated activity that invokes the RequestExternalReports process by sending a JMS/XML message using WebSphere MQ to initiate the process. Another message is eventually returned to the RequestExternalReports activity when the RequestExternalReports process completes.

The only additional step that is required on the WebSphere MQ Workflow system to invoke a BPEL process on WebSphere Business Integration Server Foundation is to export a WSDL file from the workflow system which contains the full definition of the messages structures passed to and from the workflow system. This file can be used to define the claim investigation process as a partner link in the RequestExternalReports process.

Create the UPES for the RequestExternalReports Activity

A UPES (User-defined program execution server) defines how an automated activity talks to an external program to perform an automated activity on behalf of the workflow. There are two forms of UPES, synchronous and asynchronous. An asynchronous UPES does not return control to WebSphere MQ Workflow. A synchronous UPES forces the activity to wait until completion. It can be long running. We will use a synchronous UPES.

A UPES is actually implemented as a WebSphere MQ queue. Workflow places a message on the UPES input queue when there is work to do, and the program activity listens to the queue, removing the message to perform the activity. On completion, the program activity sends a reply message to Workflow. Although each UPES input queue is created specifically for the program activity to listen to, Workflow typically only has one reply queue by default, EXECXMLINPUTQ, to listen for reply messages.

Create the UPES

To create the UPES, follow these steps:

1. Select **BuildTime** → **Network** tab → **DOMAIN** → **FMCGRP.**

2. Right-click **FMCSYS** and select **New User-Defined Program Execution Server.**

3. In the **General** tab → set **Name** to UPESSVR → select version **3.4.0** from the version drop down list.

4. In the **Message Queuing** tab, set the **Queue Name** to `WPCUPESQ` and leave Queue Manager Name blank.

> **Important:** We use WebSphere MQ clustering, and this queue will be a cluster queue defined on the WebSphere Business Integration Server Foundation machine.
>
> If you are not using clustering, the queue name should be a remote queue definition or its alias. It is bad practice to explicitly define a queue manager name because it reduces locational transparency.

5. Select **JMS-compliant XML** for the Message Format and click **OK.**

Create the Program for RequestExternalReports

This is the program to be used by the RequestExternalReports activity, it must have the same name as the name of the program that we are going to create in WebSphere Studio Application Development Integration Edition to start the RequestExternalReports process.

1. On the **BuildTime** Tab → **Implementations** tab → right-click **Programs** and select **New Program.**

2. On the General Tab:

   a. In the Name field, type `RequestExternalReportsProxy.`

3. On the Data Tab:

   a. Check **Program requires these data structures** → **Input** → browse for **requestAssessor** → **OK.**

   b. In the **Output** field → browse for **requestAssessorResponse** → **OK.**

   c. Check **Program can run unattended** .

   See Figure 11-5 on page 464.
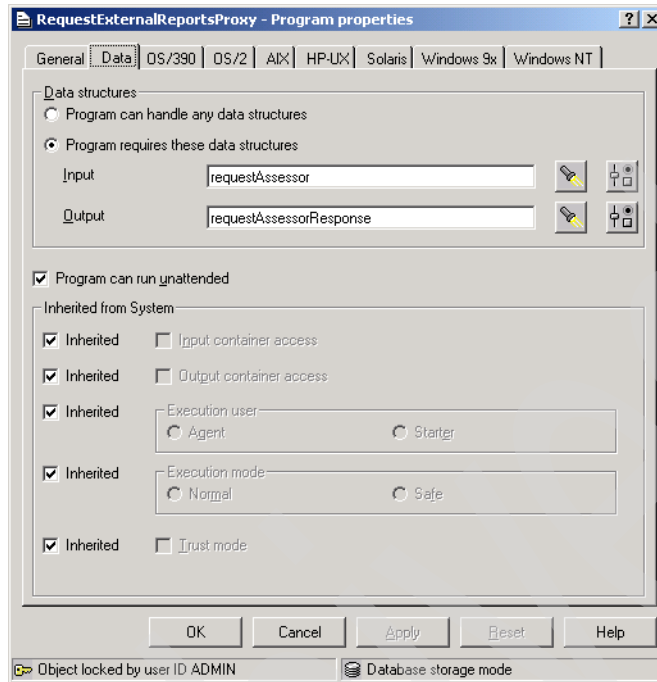
Figure 11-5   Defining the Data properties of RequestExternalReportsProxy

4. On the Windows NT® Tab:

    a. In the Path and file name field, type `RequestExternalReportsProxy.exe.`

       We could have typed any value in this field, because this executable name will never be called and does not need to be existing in the file system. It is only required because WebSphere MQ Workflow runtime requires that at least one platform definition of the program exists, otherwise the definition is considered to be invalid.

5. Click **OK.**

Configure the RequestExternalReports activity

Now we need to adjust the activity and the data mappings in which the activity is involved:

1. In **BuildTime,** select the **Process Models** → **Claim** → **Processes** tab → **ClaimInvestigation** → **RequestExternalReports.**

2. On the General Tab:

    a. Change Name to `RequestExternalReportsProxy.`

3. On the Execution Tab:

   a. *Deselect* **User program execution agent.**
   b. Check **Server** → Browse for **UPESSVR** → **OK.**

4. On the Start Tab:

   a. Check **Automatic** as the start option.

5. On the *Exit Tab*

   a. Ensure that **Automatic** is also checked

6. On the Data Tab:

   a. In the **Input** field, browse for **requestAssessor** → **OK.**

   b. In the **Output** field, browse for **requestAssessorResponse** → **OK**

7. Click **OK.**

Configure the data flow mappings

The data structures of the input and output of the RequestExternalReports activity have changed. The data to and from the activity needs to be mapped to and from the data container to the new data structures.

1. Double-click the green data connector (Figure 11-6) between **SelectReports** and **RequestExternalReports** → click **OK** to the two pop-ups warning that the data structures are invalid. We are about to fix them.
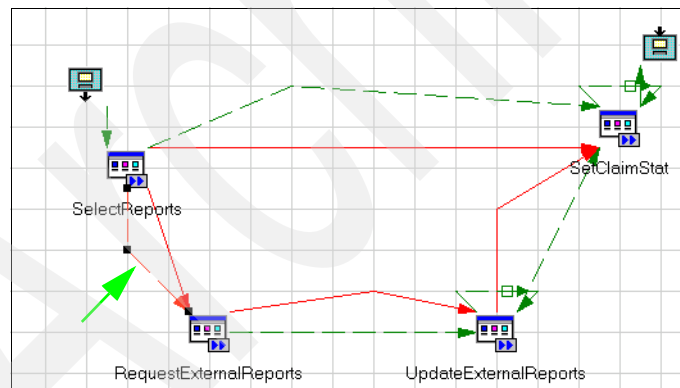
Figure 11-6 Select the data flow connector to RequestExternalReports

2. Apply the following mappings by dragging the field in the Member column of the **Origin Data Structure** to the field in the Member column of the **Target Data Structure** then click **OK.** See Figure 11-7 on page 466.

   a. **Investigate_DataInput.Claim_DataInput.**ClaimID → **claimID**
   b. **Investigate_DataInput.PolicyID** → **policyID**
   c. **Report_Claim.AssReqDate** → **requiredDate**

Chapter 11. Modify the Claim Investigation process   **465**

d. **Investigate_DataInput.LocVehicle** → **location**

e. **Investigate_DataInput.MakeOfCar** → **makeOfCar**

Figure 11-7   Mapping RequestExternalReports input data structures

3. Repeat the procedure for the data connector between **RequestExternalReports** and **UpdateExternalReports**. Ignore the po-up warning again. Map the following fields and then click **OK.** See Figure 11-8.

a. **claimID** → **Investigate_DataInput.Claim_DataInput.ClaimID**

b. **policyID** → **Investigate_DataInput.PolicyID**

c. **assCompDate** → **Report_Claim.AssActDate**

d. **reportLocal** → **Investigate_DataInput.AppEstRep**

Figure 11-8   Output mappings from RequestExternalReports

Export the FDL

The next step is to export the FDL so we can define the message structure for the RequestExternalReports process running in WebSphere Business Integration Server Foundation.

1. Select **ClaimInvestigation** in the Process Tab → **Right click** → **Save.**

2. Click the **BuildTime** menu → **Export** → **Export single objects.**

3. In the **Show objects** frame, check all the boxes → **Refresh** to be able to see all the model elements.

4. Select the **ClaimInvestigation process** (by holding down the shift key and clicking it, the icon will turn to a check). Check the **Export deep** option so that the referenced data structures and programs are also exported.

5. Select **(**shift+left mouse button) the UPES server **UPESSVR** by browsing **Network** → **OK.**

6. Chose a target folder and save as `Proxy(1).fdl` → **OK.**

11.4 Deploying the workflow process

The modified claim investigation process is now ready to be deployed to the WebSphere MQ Workflow server. All the integration work that impacts the workflow has been completed. As required by the original business requirements the impact is minimal.

To deploy the flow onto the runtime the FDL has to be imported into the runtime. The import process can either be done directly on the runtime server or by using a client machine that has the administrative components of WebSphere MQ Workflow installed.

To import the process model into the runtime, we use the *fmcibie* utility with the following options:

► The -i option to point to the file that we want to import.

► The -y parameter points to the name of the configuration.

► The -u and -p parameters are used to provide the user ID and password of the workflow administrator.

► The -to parameter tells the utility to overwrite any existing objects if required and to translate the process model into a process template, which can be executed in the runtime environment.

Working in the interop 5.1.1\bin directory where the Proxy(1).fdl was saved the results of running fmcibie are shown in Figure 11-9 on page 468.

```
D:\mqworkflow\SMP\InterOp5.1.1\bin>fmcibie -i proxy(1).fdl -y FMC -u A|
FMC24500I fmcibie is starting.
FMC24510I Import uses options
System Group name : FMCGRP
import file      : proxy(1).fdl
log file         : cerr
FMC20500I Start parsing proxy(1).fdl.
FMC25100I UPDATE STRUCTURE 'Default Data Structure' finished.
FMC25100I REPLACE STRUCTURE 'Claim_DataInput' finished.
FMC25100I REPLACE STRUCTURE 'Investigate_DataInput' finished.
FMC25100I REPLACE STRUCTURE 'Report_Claim' finished.
FMC25100I REPLACE STRUCTURE 'InvestigateClaim' finished.
FMC25100I REPLACE STRUCTURE 'requestAssessor' finished.
FMC25100I REPLACE STRUCTURE 'requestAssessorResponse' finished.
FMC25100I REPLACE PROCESS_CATEGORY 'Claim' finished.
FMC25100I REPLACE PROGRAM 'UPES_Program' finished.
FMC25100I REPLACE PROGRAM 'selectReports' finished.
FMC25100I REPLACE PROGRAM 'RequestExternalReportsProxy' finished.
FMC25100I REPLACE SERVER 'UPESSVR.FMCSYS.FMCGRP' finished.
FMC25100I CREATE SERVER 'UPES1.FMCSYS.FMCGRP' finished.
FMC25100I REPLACE PROCESS 'ClaimInvestigation' finished.
FMC20510I Finished parsing proxy(1).fdl.
FMC21500I Begin verification of process 'ClaimInvestigation'.
FMC21510I End verification of process 'ClaimInvestigation' (0 errors, (
FMC26500I Begin translation of process 'ClaimInvestigation'.
FMC26510I End translation of process 'ClaimInvestigation' (0 errors, 0
FMC24560I fmcibie finished and found 0 errors 0 warnings. RC = 0
```

Figure 11-9   Running fmcibie to deploy the workflow

> **Tip:** You might not be this lucky. Because the runtime is not in synchronization with the build time, the command to deploy the process might use the wrong verbs, such as UPDATE instead of REPLACE. You will need to edit any problematic UPDATE commands in Proxy(1).fdl that have resulted in an error. Simply delete the keyword UPDATE and the deployment will try CREATE instead, which should work.

11.5 Summary

In this chapter we have briefly reviewed WebSphere MQ Workflow and then seen how with a minimum of changes the existing claim investigation process can be modified to call RequestExternalReports as an automatic activity using a UPES server as the execution agent for the activity. The UPES server is actually just a WebSphere MQ queue. The task of making the integration with WebSphere Business Integration Server Foundation work is addressed in Chapter 12, "Integrate and test the business processes" on page 469, after all the other parts of the scenario are completed. The integration does not require any other changes to the workflow process, and is accomplished by installing a supportpac which provides the integration tools and run time processes that execute on WebSphere Business Integration Server Foundation.

Integrate and test the business processes

In Chapter 11, "Modify the Claim Investigation process" on page 453, we describe how to modify the claim investigation process to enable it to use the automated RequestExternalReports process. However, we did not explain how the request message from the claim investigation process is going to invoke the RequestExternalReports process and get a reply back, perhaps after several weeks of waiting. In this chapter we explain how to integrate the claim investigation process to use the new automated RequestExternalReports activity.

The solution requires a new supportpac, WA0D, and fixes to both WebSphere Application Server taking it to version 5.1.1.7 and WebSphere Business Integration Server Foundation to version 5.1.1.3.

12.1 Integrating WebSphere MQ Workflow and WebSphere BI Server Foundation

There are many approaches to invoke automatically business processes deployed to WebSphere Business Integration Server Foundation from WebSphere MQ Workflow process, For example we can write our own program that internally invokes the process using its WSDL file and interacts with the WebSphere MQ Workflow using the WebSphere MQ Workflow APIs to handle the required data transformations. There are also some WebSphere MQ Workflow SupportPacs offered by IBM, these SupportPacs are very useful for our situation, they can eliminate the need to write any code at all.

These SupportPacs can be easily downloaded from IBM Web site through the following URL

```
http://www.ibm.com/software/integration/support/supportpacs/product.html#wm
qwf
```

12.1.1 Candidate SupportPacs

There are two SupportPacs that fit our requirement to invoke a business processes deployed to WebSphere Business Integration Server Foundation from WebSphere MQ Workflow:

1. WA07: WebSphere MQ Workflow Web services Process Management Toolkit

   Using this toolkit, WebSphere MQ Workflow process designers can define activities that are implemented by Web services and combine them with other activities, human-facing for example, into a business process.

   In addition, the toolkit can be used to deploy any WebSphere MQ Workflow business process as a *Flow Service*, a Web service that exposes the life cycle protocol of the business process.

2. WA0D: WebSphere MQ Workflow and WAS Enterprise Process Choreographer Inter-Operability

   This SupportPac enables interoperability between WebSphere MQ Workflow and WebSphere Business Integration Server Foundation. Being able to have processes that span across both products, combines the strengths of each product and allows a smooth and partial migration of existing processes from an existing WebSphere MQ Workflow base to the Process Choreographer.

   Using this SupportPac, the user can invoke a Process Choreographer process from an WebSphere MQ Workflow process and vice versa.

Both SupportPacs use the UPES feature of WebSphere MQ Workflow. We use WA0D because it is easier and faster to implement the solution using it and also because it is specific for our requirement. In contrast, WA07 requires more customization of the existing WebSphere MQ Workflow ClaimInvestigation process.

12.2 SupportPac WA0D overview

This SupportPac provides the capability of invoking a choreographer process from WebSphere MQ Workflow and vise versa. This requires us to understand the interfaces to WebSphere MQ Workflow and Process Choreographer.

12.2.1 WebSphere MQ Workflow interfaces

WebSphere MQ Workflow consists of several server components that communicate using WebSphere MQ queues. The most important server component for interoperability is the execution server. It is responsible for starting and navigating WebSphere MQ Workflow processes.

Information about WebSphere MQ Workflow processes, the infrastructure, and participating people (staff) are made known to the execution server by populating the runtime database with the corresponding information. This is done by importing a flow-definition-language (FDL) file that is generated using WebSphere MQ Workflow buildtime or any other tool similar to WebSphere Business Integration Modeler.

The execution server uses two message formats for message exchange through WebSphere MQ queues:

► SDDS is an internal WebSphere MQ Workflow format.
► XML is a subset of the SDDS messages in XML format, and it can also include JMS headers for JMS support.

12.2.2 Process Choreographer interfaces

Choreographer components can be configured to use WebSphere MQ as a JMS provider, so that it can send or receive messages through WebSphere MQ queues.

Process choreographer provides two main interfaces to the outside world:

► API and facade message driven beans

The base functionality of choreographer is externalized as an API to work with process templates, process instances, activity instances, and work items. In

addition, for every process a process-specific EJB, an optional process-specific MDB façade is generated.

The façade MDB represents the process according to the façade design pattern. For more information about design patterns go to:

http://www.research.ibm.com/designpatterns/

A façade MDB allows sending JMS messages in a custom format into a dedicated queue, reception of such message by the MDB causes an implicit call of the associated process. On completion of the process, the response is sent back to the reply-to queue as specified in the request.

► Choreographer plug-ins

Process choreographer implements a general-purpose process engine that can be plugged into a variety of environments. It also delegates functionality to other plug-ins to extend its base functionality with additional features. Plug-ins are the Process choreographer engine's way to delegate tasks to the external world, in areas where more than one implementation is potentially possible, and where different implementations should not affect the behavior of the process engine itself. The following functions are delegated to plug-ins:

– Handling process data (messages), including evaluating conditions and executing mapping activities.

– Invocation of elemental operations (activities)

– Dealing with people (authentication, authorization and staff resolution).

– Reacting to state changes of processes or activities, for example, logging changes to an audit trail or to publish events.

12.2.3 The SupportPac Architecture

The SupportPac is composed of two main components:

► Tools installed on the MQ WorkFlow server

These tools provide the ability to export a WSDL file for an existing BuildTime process from the BuildTime user interface and it is used when we want to invoke a WebSphere MQ Workflow process from WPC. Also there is another utility that runs from the command prompt to generate a WSDL file from and FDL file and it is used when we want to invoke a WPC process from WebSphere MQ Workflow and is explained in details in the following sections.

► Message-driven bean (MDB) installed on WebSphere BI Server Foundation

In addition to the tools, a MDB is also included in the SupportPac as an EAR (Enterprise Archive) file. Deploy it to WebSphere BI Server Foundation, where this MDB listens for the invocation messages coming over an MQ

queue (the UPES queue), interprets this message, and invoke the proper WPC process accordingly.

Details of how these components contribute to the SupportPac functionality is described in the next section.

12.2.4 How the SupportPac works

This SupportPac has two scenarios to describe how this interoperability works:

- ► How a WebSphere MQ Workflow process invokes a choreographer process.
- ► How a choreographer process invokes an WebSphere MQ Workflow process.

We focus only on the first scenario that we implement for our solution.

How WebSphere MQ Workflow invokes Choreographer

Assume that we already have:

- ► The interface of the Choreographer process with its name
- ► Input and output
- ► A WebSphere MQ Workflow process where one of its activities is responsible for invoking the Choreographer process.

This activity is defined as an automated unattended activity with a UPES definition, which points to a WebSphere MQ queue. In addition, we must define a program that has the same name, input and output data structures of the process to be invoked in choreographer. We use this program as the program for this activity.

After we complete these modeling steps, we export a FDL that contains these modifications, and we use the command line tool (`runfdl2wsdl.bat`) included in the SupportPac to generate a WSDL file representing the interface to the WPC process. It defines the input and output message structures of the Process choreographer business process. The generated WSDL file contains the interface definitions of every process and every UPES activity in the input FDL file.

The WSDL file must be imported in WebSphere Studio Application Development Integration Edition to generate the corresponding objects that represent the messages. After this, we create the process in WebSphere Studio Application Development Integration Edition and use the WSDL file as partner link for its receive and reply activities and model the internals of the process in WebSphere Studio Application Development Integration Edition. Figure 12-1 on page 474 illustrates these steps.
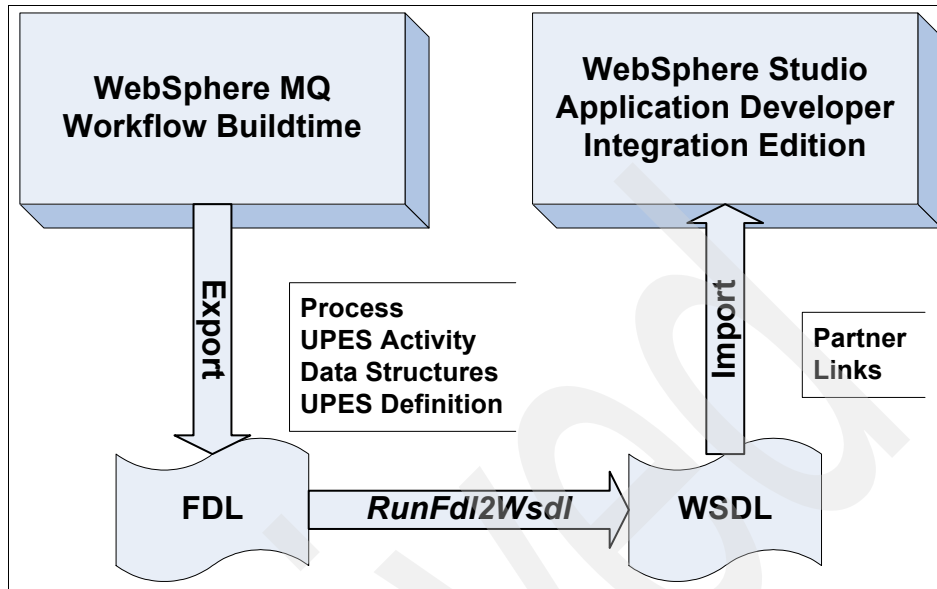
Figure 12-1   Steps to invoke a choreographer process from WebSphere MQ Workflow

We always need a unique choreographer process for each integration. To allow WebSphere MQ Workflow processes to communicate with an existing choreographer process, we can create a new process activity that initiates the existing choreographer process. This is a proxy design pattern, because the new choreographer process is a placeholder or proxy for the existing choreographer process. It controls access to the existing process in a way that conforms to the interoperability interface.

Execution

When the MQ Workflow process is executed and the UPES activity is started, the execution server sends a WebSphere MQ Workflow XML message of type ActivityImplInvoke message to the specified UPES queue. The MDB of the SupportPac reads the message, gets the program name from it, searches in choreographer for the process that has the same name as the program, and invokes it giving it the input data included in the message. When the process completes, the MDB converts the response of the process to a WebSphere MQ Workflow XML message of type ActivityImplInvokeResponse and puts it on the input queue for the WebSphere MQ Workflow. The default WebSphere MQ Workflow input queue for XML messages of the execution server is EXEXMLINPUTQ.

Figure 12-2 on page 475 shows the execution steps when an WebSphere MQ Workflow process invokes a choreographer process.
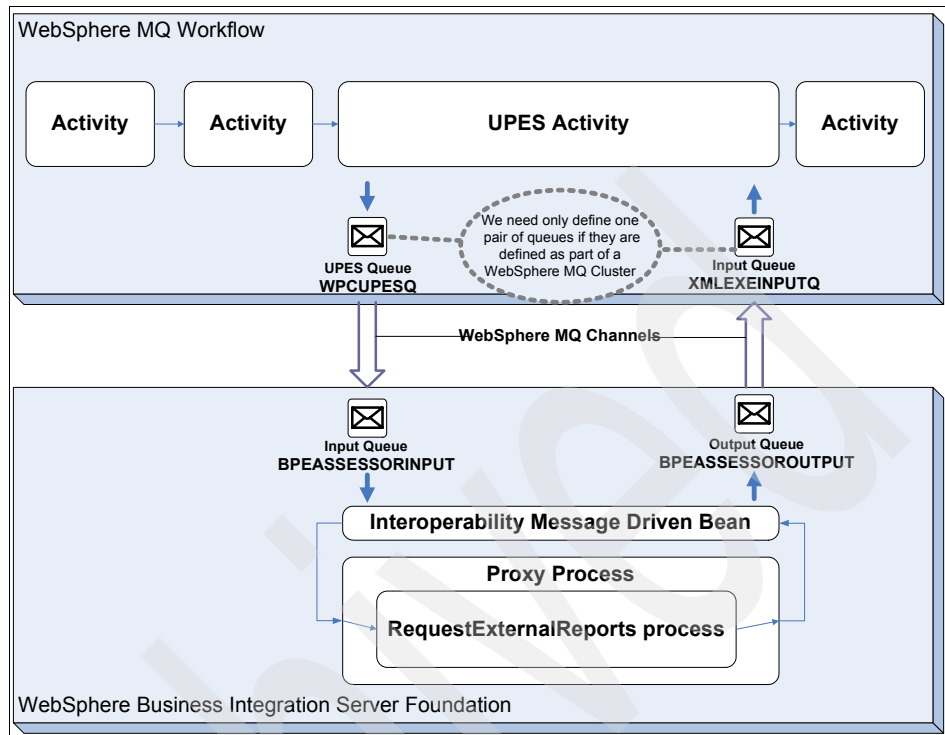
Figure 12-2 Process integration

12.3 Installing and configuring the WA0D SupportPac

Installing and configuring the WA0D SupportPac includes the following steps illustrated in Figure 12-3 on page 476. These steps are described in the corresponding numbered sections below.

1. Upgrade the WebSphere Application Server and WebSphere Business Integration Server Foundation installations to cumulative fix pack 3 and 7 respectively if they are not already at that level.

2. WPCUPESQ is the (clustered) queue which receives requests from the workflow system to be sent to Process Choreographer.

3. WA0D is installed on the workflow system, which must also have WebSphere Application Server installed.
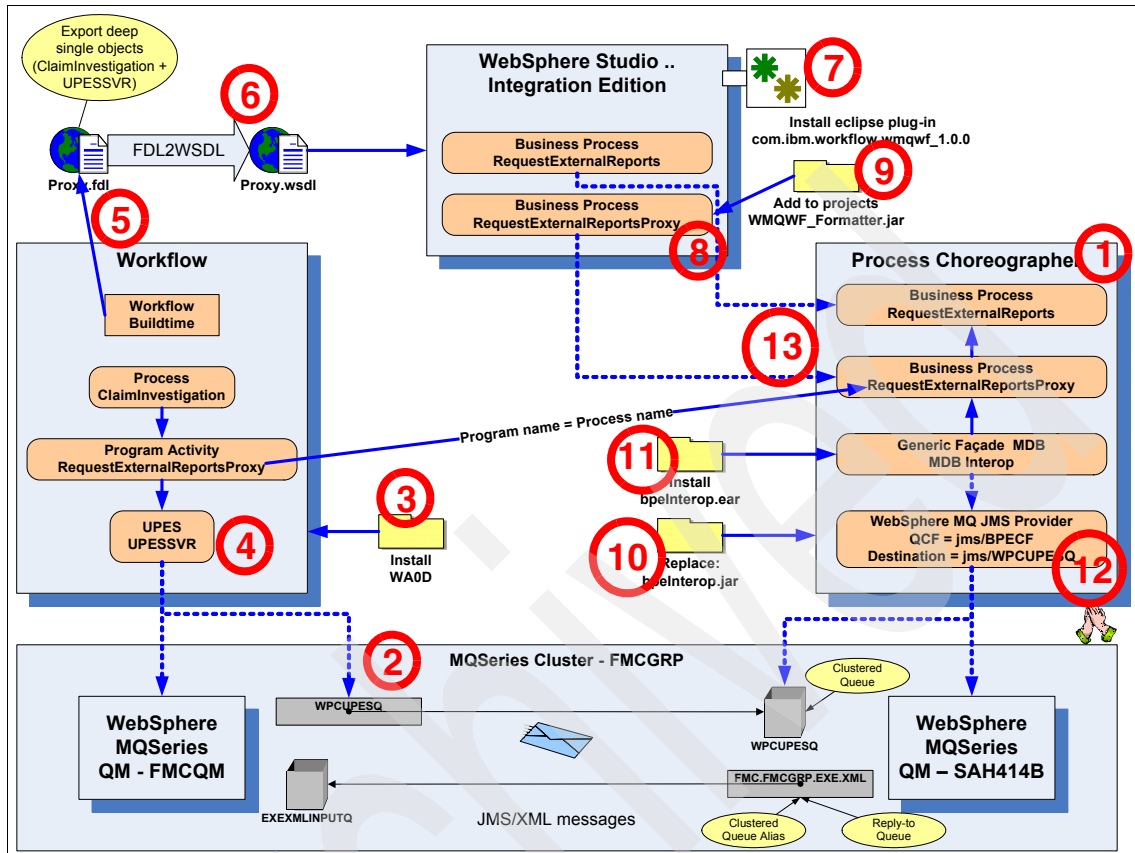
Figure 12-3   Installation and configuration of WA0D

4. The workflow process uses a program activity and the UPES server is configured to place its WPC requests on the WPCUPESQ. We do this in 11.3.3, "Define the interface to RequestExternalReports" on page 462.

5. The FDL for the process and the UPES definition are exported as FDL. See , "Export the FDL" on page 467.

6. The `runfdl2wsdl` command shipped in WA0D converts the FDL to WSDL to be consumed by WebSphere Studio Application Development Integration Edition to define the proxyRequestExternalReports process.

7. Install the eclipse plug-in `com.ibm.workflow.wmqwf_1.0.0` into WebSphere Studio Application Development Integration Edition.

   Extract com.ibm.workflow.wmqwf_1.0.0.zip from the directory to which you installed WA0D, into your WebSphere Studio Application Development

Integration Edition installation directory\eclipse\plugins and restart WebSphere Studio Application Development Integration Edition.

8. Create a new proxy process RequestExternalReportsProxy in WebSphere Studio Application Development Integration Edition to receive the service requests from the Workflow ClaimInvestigation RequestExternalReports Proxy program activity. It is the equivalence of these names that is responsible for routing the request to the correct Process Choreographer process.

9. Add `WMQ_Formatter.jar` to the new process, and add it to the java build path. The .jar file parses the incoming JMS/XML message.

10. (Replace `bpeInterop.jar` in the WebSphere Business Integration Server Foundation library with the one shipped in the SupportPac.) This step is not necessary if you are working with WebSphere Business Integration Server Foundation cumulative Fix Pack 3 and WebSphere Application Server cumulative Fix Pack 7.

11. Install and start the supplied `bpeInterop.ear` file in WebSphere Business Integration Server Foundation. This is the generic façade MDB that routes and starts the RequestExternalReportsProxy process.

12. Using the WebSphere Application Server Administration console connected to WebSphere Business Integration Server Foundation configure the WebSphere MQ messaging provider. This is described in 12.3.1, "Upgrade WebSphere Business Integration Server Foundation" on page 477 that follows.

13. Install and start the RequestExternalReportsProxy .ear files on WebSphere Business Integration Server Foundation.

12.3.1 Upgrade WebSphere Business Integration Server Foundation

The WA0D support pack requires fixes to WebSphere Business Integration Server Foundation which in turn pre-reqs an upgrade to WebSphere Application Server. There is guidance in section 7.3.3, "WebSphere Business Integration Server Foundation" on page 237 how to apply the updates.

12.3.2 Define WPCUPESQ

In the example we define a new local queue, WPCUPESQ, on SAH414B using WebSphere MQ Explorer, and make it a member of the cluster FMCGRP.

12.3.3 Install WA0D

The interoperability SupportPac needs to be installed. it is best to do this on the WebSphere MQ Workflow server, which in our example is SAH414A.
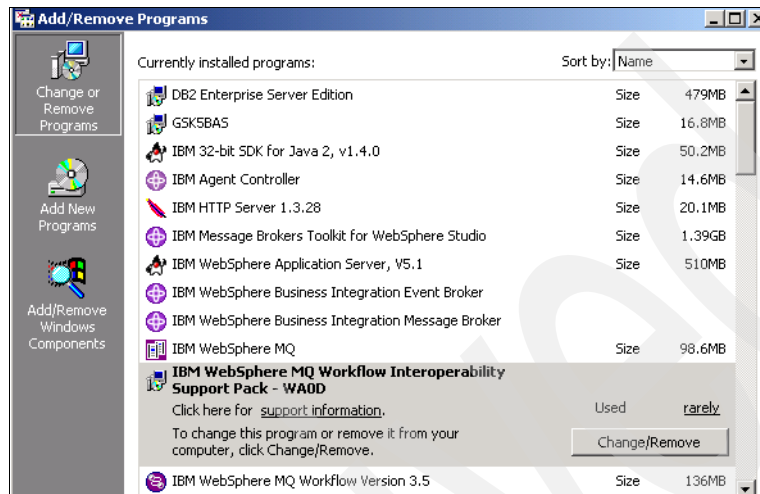
Figure 12-4   Confirming Interop supportpac is installed on SAH414A

1. Create a subdirectory under the Workflow installation directory, such as WA0D, copy the `interop.exe` installation file and run it.

2. Respond to the prompt asking for the location of WebSphere Application Server.

3. Respond to the prompt asking you for the location of the Java runtime with the location of the WebSphere Application Server java runtime.

12.3.4 Configure the claim investigation process

Verify the RequestExternalReports activity has been modified to call a UPES correctly as described in Section 11.3.3, "Define the interface to RequestExternalReports" on page 462.

1. The program activity is named RequestExternalReportsProxy.

2. The UPES is called UPESSVR, which points at the queue manager FMCQM and the queue WPCUPESQ.

3. The program implementation is called RequestExternalReportsProxy.

4. The input and output data containers have been mapped.

12.3.5 Generate FDL

We described how to export the modified FDL from the claim investigation process in the section, "Export the FDL" on page 467.

12.3.6 Generate WSDL for the proxy process

To general the WSDL, follow these steps:

1. Open a command prompt and change the current folder to the **bin** folder under your InterOp installation folder, for example type in the below command:

   ```
   cd D:\mqworkflow\SMP\InterOp5.1.1\bin
   ```

2. Execute setenv.bat by typing in the below command:

   ```
   setenv.bat
   ```

3. Generate the WSDL file by typing in this command to get the file generated and saved as Proxy.wsdl:

   ```
   runfdl2wsdl Proxy.fdl Proxy.wsdl
   ```

Examine the WSDL with your preferred editor. You can find the definitions for the RequestExternalReports interface embedded within larger workflow structures.

12.3.7 Install the Supportpac Eclipse plug-in
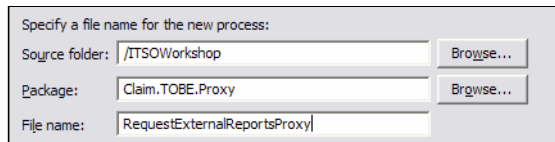
Extract `com.ibm.workflow.wmqwf_1.0.0.zip` from the directory you installed WA0D to, into your `WebSphere Studio Application Development Integration Edition installation directory\eclipse\plugins` and restart WebSphere Studio Application Development Integration Edition.

12.3.8 Create the RequestExternalReportsProxy process

Follow these steps:

1. Make sure you have saved a project interchange file for the RequestExternalReports projects.

2. Open a new workspace and import the ITSOLGI service project from the RequestExternalReports project interchange zip file.

3. Perform step 12.3.9, "Add WMQ_Formatter.jar to the process" on page 481 now, and then return to the next step.

4. Create a new business process. See Figure 12-5 on page 480. Use the create business process wizard in the pop-up menu.

5. Set the long running attribute on the business process on the Server tab of the RequestExternalReportProxy process

Specify a file name for the new process:		
Source folder:	/ITSOWorkshop	Browse...
Package:	Claim.TOBE.Proxy	Browse...
File name:	RequestExternalReportsProxy	

Figure 12-5   Adding the long running proxy process

6. Add a flow to the process, and move the request and reply activities inside the flow.

7. Import the proxy.wsdl into the new package and drop the WSDL file on the canvas to create a new partner link.

8. Create a new OutputVariable, and wire up the receive and reply activities. Remember to set the process role.

9. Temporarily connect the request and reply activities, save the flow, and check that there are no compilation errors.

10. Drag and drop the RequestExternalReports process into the flow and wire it up to an Invoke activity, creating two new input and output variables called `RequestExternalReportsInputVariable` and `RequestExternalReportsOutputVariable`. Map the input and output messages from the ExternalClaimsAssessorInterface.wsdl file. Set the Operation for the Invoke activity.

11. Wire up the flow again, save it and check for no compilation errors.

12. Now you need to add two transformers: a simple transformer for the input of the RequestExternalReports process, and an aggregation transformer for the output. You have to pass some of the input back to workflow. Make sure you are pointing at the Claim.TOBE.Proxy package.

   a. Use the simple transformation wizard to create TransformRequestExternalReports and drop the transform on the canvas and wire it up.

   b. For the aggregation transform TransformReplyRequestExternalReportProxy, call the aggregation wsdl AggregateReplyRequestExternalReportProxy.

13. Save the flow, shown in Figure 12-6 on page 481.
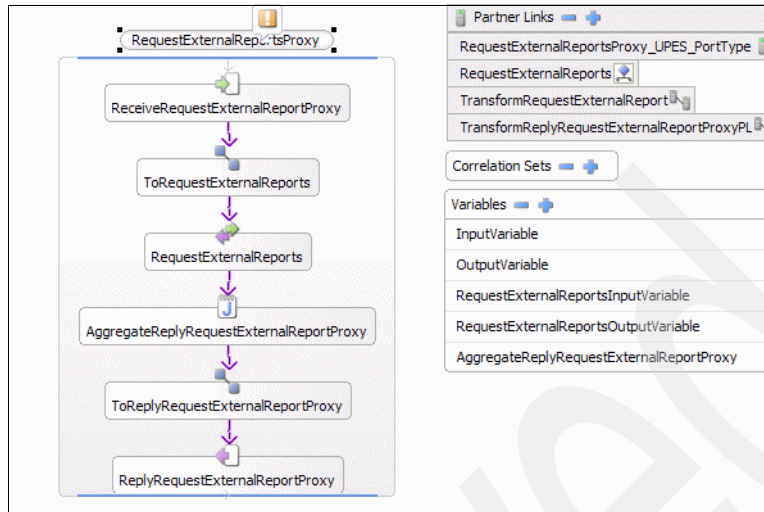
Figure 12-6   Long running proxy calling RequestExternalReports

14. Generate the deploy code for the RequestExternalReports process.

15. Create the Test Server in WebSphere Studio Application Development Integration Edition, add the ITSOLGI project, create the database tables, and unit test the flow.

12.3.9 Add WMQ_Formatter.jar to the process

To add the WMQ_Formatter.jar to the process, follow these steps:

1. Select the service project → **Import** → **File system** → and browse to `WMQWF_Formatter.jar` in the SupportPac → **Finish.**

2. Right-click the ShortRunningProcess folder → **Properties** → **Java Build Path** → select the **Libraries** tab → **Add JARs** → **WMQWF_Formatter.jar** → **OK** → **OK.**

12.3.10 Replace bpeInterop.jar in the server library

This step is no longer required if you are installing WA0D on top of WebSphere Business Integration Server Foundation 5.1.1.3.

12.3.11 Install bpeInterop.ear

To install itne bpeInterop.ear, follow these steps:

1. Use the address to open the Admin console for WebSphere Business Integration Server Foundation in a browser:

   ```
   http://SAH414B:9090/Admin
   ```

2. Under **Applications** → **Install** → **bpeInterop.ear**.

3. Check default bindings, set a short directory name such as `D:\Apps` as the install directory, and check **Deploy EJBs**.

4. In step 2, Figure 12-7, provide InteropMDBListenerPort as the listener port.

Install New Application

Allows installation of Enterprise Applications and Module

Step 1   Provide options to perform the installation

→ **Step 2 : Provide Listener Ports for Messaging Beans**

Each message driven enterprise bean in your application or module must be bound to a listener port name.

EJB Module	EJB	URI	Listener Port
MQWFInteropMDB	MqwfFacadeMDB	bpeinteropear.jar,META-INF/ejb-jar.xml	

Previous   Next   Cancel

Step 3   Map modules to application servers
Step 4   Map security roles to users/groups
Step 5   Summary

Figure 12-7   Configuring the interoperability MDB listener port

5. Save the configuration and start the application.

Configure the messaging resources

To configure the messaging resources, follow these steps:

1. Verify the queue connection factory jms/BPECF points to the queue manager on the WebSphere Business Integration Server Foundation server.

2. Add two new destinations, jms/WPCUPESQ inbound requests and FMC.FMCGRP.EXE.XML for outbound replies and requests.

   See Figure 12-8 on page 483.

Figure 12-8   Configuring jms/WPCUPESQ

The Queue manager field is left blank. SAH414B is the default queue manager in our example. See Figure 12-9.

Figure 12-9   Configuring FMC.FMCGRP.EXE.XML destination

3. Define a message listener port on the server for the bpeInterop MDB called `InterOpMDBListenerPort`. See Figure 12-10.

Figure 12-10   Configuring the bpeInterop listener port

a. The supportPac comes with a jacl script that can be modified to configure the interoperability settings, `InterOp_Sample1_config.jacl`. An example of setting the parameters in the script for the workshop is illustrated in Example 12-1:

Example 12-1   Workshop script parameters

```
####################################################################
# SETUP MQ Queue Connector Factory, MQ queue name and listener port #
# for IBM MQ Websphere Workflow Inter operability support pack      #
####################################################################
set MQINSTALLROOT      "D:\\WebSphere MQ"¹
set InterOp_HOME       "D:\\Interop5.1.1"
set serverName         "server1"
set FMCQM_QCF_Name     "FMCQM"
set EXEXMLINPUTQ_QName  "FMC.FMCGRP.EXE.XML"
set MYUPESQ_QName      "WPCUPESQ"
set ListenerPort_Name  "InterOpMDBListenerPort"
set JMSProviderName    "WebSphere MQ JMS Provider"
set nodeName           $env(local.node)
D:\Websphere\AppServer\bin\wsadmin -f InterOp_Sample_config.jacl
```

b. You will also need to modify the line that installs the sample business process. Change the lines below to point to the RequestExternalReportsProxy.ear

```
puts "Install sample 1 ear file: WMQWF2WPC.ear"
$AdminApp install $InterOp_HOME\\samples\\WMQWF2WPC\\WMQWF2WPC.ear
```

---

¹ On windows, use double backslashes instead of single ones, for example `C:\\InterOp5.1.1`

c. Run the script using the wsadmin command

```
D:\Websphere\AppServer\bin\wsadmin -f InterOp_Sample_config.jacl
```

12.3.12 Install the RequestExternalReportsProxy process

To install the RequestExternalReportsProxy process, follow these steps:

1. Follow the same installation procedure as before, but there is no listener port to identify this time.

2. Name the application `RequestExternalReportsProxy`.

3. If you need to reinstall the .ear, stop the application and the business process.

> **Tip:** If you need to reinstall the application, you need to stop it before uninstalling. Stop appears to work (the running icon changes to red and the stop is successful. You need to stop all long tasks, including long-running tasks before you can stop the process application. But when you attempt to uninstall, it still may fail to uninstall.
>
> This can be solved by manually stopping the business process container. Click **Enterprise Applications** → **RequestExternalReports** → **EJB Modules** (under the related items list) → **Business processes** → stop the process manually. This might solve the problem and allow the application to be uninstalled.

Everything is now configured to test the integration of Workflow and Business Process Choreographer.

12.4 Test the integration

We recommend testing the integration in three steps:

1. Unit-test the entire RequestExternalReportsProxy process using the Business process test client.

2. Unit-test the claim investigation process again.

3. Finally, test the integrated process by submitting a claim assessment using the Workflow test client.

12.5 Summary

In this chapter we have described how to install configure and use the WA0D support pack to integrate the existing claim investigation process with the new automated RequestExternalReports process.

13

Points to consider

The first part of this concluding chapter is a post mortem. What lessons did we learn from building the solution? Do these lessons have a bearing on integration projects in which you are involved?

In the second part of the chapter, we list some of the changes that have been made to the next version of the WebSphere platform. IBM System House, who, you might recall from the preface sponsored this Redbook, were responsible for working with developers to improve the integration capabilities of middleware from IBM. So it should come as no surprise to you, that there are many new features and detailed improvements in version 6 of the WebSphere platform that make building the External Claims Assessor scenario less time-consuming.

The most dramatic improvements can be found in the integration of the messaging and application servers' technologies around SOA. But there are also many changes in the underlying structure, such as:

► Improving the BPEL interface from WebSphere Business Integration Modeler,

► Improving the XSLT processing in the Assign activity in WebSphere Integration Developer to make it a real alternative to using transformation services

► Support for aggregation over non-MQ transports in WebSphere Business Integration Message Broker.

► Publication of a UML profile for software services written by Simon Johnson, at this Web site:

http://www.ibm.com/developerworks/rational/library/05/419_soa/

Details about applying this profile, and extending Rational Software Architect to support SOA can be found in *Patterns: Model Driven Development using Rational Software Architect*, SG24-7105.

These and other improvements should persuade you to base new integration projects on Version 6 rather than Version 5 of the WebSphere platform.

13.1 Lessons learned

The main lessons we learned largely concerned the way in which members of the team should work together. Primarily, these lessons revolved around the role of the architect, and the relationship of the architectect with the business analyst and IT specialist, and the policies and guidelines that are established for the project by the business analyst and IT architect. The way these roles plan to work together and how the tools support them, more than anything, will determine the outcome of your integration project.

13.1.1 Business Modeling and IT Architecture

We found WebSphere Business Integration Modeler to be a useful requirements and analysis tool:

► It helps to organize and document information about activities, roles, resources, costs and processes that can be exported to the IT architect in a formal model. This is a practical way to help the IT Architect and the Business Analyst to use common terms to describe the scenario, sharing common parts of their models of the solution.

► In the hands of an *experienced* business analyst, the model is an effective tool for making rapid return on investment and resource usage predictions.

We place the emphasis on *experienced* because, with the goal of making early predictions, detailed knowledge about process implementation will be lacking. An experienced business analyst will be able to select the significant cost elements and key activities without getting bogged down in trying to model the process implementation details.

Who defines the executable business process?

We found the business analyst's initial objectives of collecting requirements, exploring process improvements, estimating resource requirements, and return on investment is not congruent with the pressure from the rest of the team to define an executable process ready for implementation as soon as possible. There are a number of reasons for this, and these need to be understood by the project manager in building a development plan:

► A matter of timing

Process definition needs to proceed in parallel with detailed business analysis to get the project underway as a whole.

- A matter of the business analyst's perspective, skills and experience.
  - Analyzing a process from the *perspective* of looking for opportunities for improvement is different from designing a process that will function robustly in its use of a combination of manual and automated activities.

    The process model developed by the analyst needs to be elaborated to account for IT requirements and constraints that were not pertinent to value of the business process.
  - Analyzing and describing the process down to executable detail requires more IT architectural *skill* than a business analyst normally possesses.

    Be aware that the process model defined by the business analyst will most likely not be a wholly adequate process definition for the rest of the project.
  - IT *experience* is needed to recognize the impact and significance of process changes on the underlying IT infrastructure.

    In our project the analyst focused on the process improvements required by LGI, but not on the business and IT relationships that needed to be created between LGI and its assessors to make the automated process work. Because of lack of skill and experience in Business to Business (B2B) IT, the significance of replacing manual communication between LGI and its assessors with automatic communication was not sufficiently appreciated.

Here is a small example where additional analysis and use of best practice patterns for B2B interactions would have avoided a problem that arose later in our implementation.

In Section 10.4.7, "Configuring the flow to wait for responses from the assessors" on page 416, the asynchronous responses from the chosen assessor are implemented by being received directly into the RequestExternalReports process.

The receive activity was imported directly from the process definition in the business model. What was not fully appreciated until the IT specialist started testing the process, is this model has an implicit requirement that the responses from assessors are received in the right order, an acknowledgement followed by the report. Otherwise, the process throws a fault.

Relying upon these two conditons makes for a fragile implementation:
- Assessor producing both responses and in the right order
- Infrastructure not jumbling up the responses

A better architecture would have been to introduce a public/private process pattern common in B2B designs, and to model the partner interactions separately from LGI's RequestExternalReports process.

It is also widely thought that a better IT model for protocol-based interactions is a state machine-based model rather than a process model. WebSphere Process Server Version 6 provides a mechanism for implementing state machine models for business interactions.

Our conclusion is that the CIM must be the result of a *collaborative* engineering process conducted by both the Business Analyst and IT Architect. The IT architect, in building the IT model for the business process, discovers overlaps and gaps which lead to the aggregation or creation of new components. These modifications must be discussed with the analyst to understand the business requirements for these components and the choice of an appropriate IT model.

For example, in the case of the External Claim Assessor scenario, the business analyst would be invited to define the business requirements and business processes behind the relationship between LGI and the external assessors. This, in turn, would lead the IT architect to definite a state machine to implement the relationship between LGI and the Assessors that could be robustly implemented.

From a process perspective, we recommend:

► The CIM is owned by the business analyst and one of the approvers should be the IT Architect.

► The PIM is owned by the IT architect one of the approvers should be the business analyst.

13.1.2 Export BPEL from WebSphere Business Integration Modeler?

We decided to import the BPEL from Modeler and refine it in WebSphere Studio Application Development Integration Edition.

This approach had its genesis in our team following the development waterfall paradigm supported by the tool chain. The BPEL was exported by the business analyst to the process specialist.

There was a lot of interaction between the analyst and the architect in the development of the BPEL model in the spirit of collaborative development. The resulting BPEL was imported into WebSphere Studio Application Development Integration Edition by the process specialist. However, the process specialist found working with generated BPEL introduced difficulties because our development standards, created by the architect using Rational Software Architect, were not incorporated into the analyst's model. Consequently, the IT process specialist took the practical option of deleting a number of artifacts generated in the BPEL in WebSphere Studio Application Development Integration Edition.

Our recommendation, which is more practical with Version 6 of Modeler, is for the process specialist, as a precursor to exporting the BPEL from modeler, to refine the business process to make sure it meets all in-house standards, such as naming standards and package structures. The process specialist can also choose to model the data flows in the process model to generate BPEL more closely resembling the final executable process.

13.1.3 Naming

Extensive renaming and refactoring is difficult to do in all the tools without producing unwanted side-effects that are time consuming to fix. The issue is only made more difficult with using multiple tools loosely coupled by import/export mechanisms, and needing to work name and interface changes back into multiple artifacts. We introduced more bugs and lost more time through self-inflicted naming problems and interface changes than any other source of problem.

► Limited refactoring capabilities beyond changing file names makes changing the names of WSDL components particularly troublesome.

► Limited control over sharing artifacts between tools makes it all the more important for the solution team to have a systematic approach to naming objects and artifacts, a well-defined point of control such as a library system and ownership of artifacts, and to review interfaces thoroughly before proceeding to implementation.

► A particular problem with naming in this scenario is that there are a lot of related objects, all of which end up getting very similar names. Because interactions flow from one component to another, nothing really changes except the endpoints. The result was that attempting to give things meaningful names rapidly led to duplication and confusion. This was probably our second biggest source of problems. The choice of names must be analyzed in great depth by the solution architect.

Our suggestion is to give artifact names a short descriptive part and a simple unique name tag, such as a short number. Naming can be federated by prefixing the number with a letter for each department, AssessAvail_A1.wsdl, for example. *Avoid* special characters, such as (,[,{ and so on at all costs! And keep names and paths *really short* to avoid path length problems using Eclipse products on Windows. Establishing a naming convention based on components might help. We were continually asking: I*s that the availability request into the proxy or into the real assessor? Is that the availability response back to the choreographer, or the response to the proxy?*". However, we found the simple numbering of flows worked better than anything else.

13.1.4 Metadata

We had a choice of three forms of metadata to use to describe our solution interfaces:

- ► UML
- ► WSDL
- ► Mixture of WSDL and XSD

We chose to use WSDL. This led to some problems exchanging interface data between WebSphere Business Integration Message Broker and WebSphere Studio Application Development Integration Edition. These problems might have been easier to manage had we used imported rather than in-line schemas in our WSDL. There were a few occasions where interfaces diverged because the in-line schemas in the WSDL definitions diverged unnecessarily, simply because the schemas were developed in line at different times.

Now that WebSphere Business Integration Message Broker in version 6 supports WSDL import, our choice of using WSDL might prove less error-prone. But we ended up thinking the problems resulting from out of sync in-line schemas were so costly that using WSDL with imported schemas is the preferable method. Using a common repository of schemas imported in different WSDL files should encourage more reuse and less chance of spawning slightly different message formats. This solution requires a little extra work to extract the in-line schemas generated automatically from EJBs.

13.1.5 Service Bus

The implementation of the message flows in the broker took longer than expected. This was partly a result of the version 5 broker not implementing the SOAP 1.1 schema and not being able to import WSDL, and partly that by adopting SOAP/HTTP as a transport, because of integration difficulties with SOAP/JMS, we also introduced additional costs implementing distribution and aggregation.

We also never tackled the problems of dealing with transport level error-handling. Transport errors are better handled by the middleware when using SOAP/JMS than they are with SOAP/HTTP. If two-way SOAP interactions are used, then a good SOA architecture should address the issue of handling SOAP fault messages on behalf of SOAP clients.

Some of these problems are immediately resolved by moving to version 6 of WebSphere Business Integration Message Broker and using WebSphere Platform Messaging as well as WebSphere MQ where appropriate. Some other problems are resolved by studying patterns for SOA, such as in *Patterns: Model-Driven Development Using IBM Rational Software Architect*, SG24-7105.

The lessons we drew from our experience with version 5 were:

1. Separate the architecture and implementation of the Service Bus from the design and implementation of the proxyAssessorSystem components implemented on the service bus.

   The IT specialist, in implementing our broker components, was required to create the basic message transport as well as implementing the specific features required for the assessors. The components were not isolated from one another sufficiently. The design is not readily extensible.

2. The architectural involvement in designing the bus and its components was too light. The architect lacked the tools to describe the service bus down to the level of interaction design. The implementer found that the version 5 level of middleware left a lot of work to be done to create a service bus on which to build the proxyAssessorSystem.

   The root cause of these problems is we needed to revisit business requirements with the help of the business analyst when the IT architect identified the need for the proxyAssessorSystem. The result would have been a better separation of concerns between the ESB and the process layer.

Better tooling for the ESB in WebSphere Version 6, better patterns and models for SOA, and the implementation of state machines to implement business protocols in WebSphere Process Server Version 6 would improve the design and implementation of the service bus.

13.1.6 Conclusion

The business-driven development approach we adopted, creating a solution to a specific business problem, focused the solution team on how to model the solution and then how to refine the model into implementation using multiple tools and middleware components.

Compared to established IT driven development, there was less emphasis on the development of an IT infrastructure and more on driving the business requirements as directly as possible into a solution.

We needed to give more attention to understanding and defining IT issues: architectural issues were left to the IT specialists to solve in their implementation. The project would have proceeded more smoothly had these issues been resolved earlier. This was perhaps a manifestation of working in a laboratory, and not working with a real IT infrastructure. Had this been a real project, then the IT infrastructure would already have existed and more attention would have had to be given to any changes to it.

Nonetheless, there is a parallel to be drawn with real-world projects that follow a Business Driven Development approach. It is important to identify early on what changes and resources need to be allocated to the infrastructure team to build a service bus that will support the capabilities required by new SOA solutions that are deployed to it. This is part of a successful strategy for an On-Demand operating environment.

13.2 Tooling and middleware changes

Since building the scenario, much of the underlying software has moved from WebSphere Version 5 to Version 6. In some cases, this makes very little difference to the architecture and implementation, but in others there are significant improvements, both in opportunities to improve development productivity and in making the solution more robust.

13.2.1 WebSphere MQ

WebSphere MQ has moved from Version 5 to Version 6. The only changes that affect the project are:

► WebSphere MQ Version 6 uses an Eclipse rather than Microsoft Management Console based explorer for management.

► WebSphere MQ Workflow support level should be upgraded for the queue configuration to work smoothly with WebSphere MQ version 6.

13.2.2 WebSphere MQ Workflow

The support level for WebSphere MQ Workflow has increased from 3.1.4 to 3.1.7

13.2.3 WebSphere Application Server

WebSphere Application Server V6.0 includes WebSphere platform messaging built on top of JMS. WebSphere platform messaging differs significantly from the embedded messaging and optional plug-in WebSphere MQ messaging in version 5. From an architectural perspective, WebSphere Application Server messaging integrates with a WebSphere MQ server running on another server and at the same time is a an integral part of WebSphere Application Server. This should make it easier to build a messaging bus integrating the application server and WebSphere MQ parts of the solution, and should make it possible, for performance and reliability reasons, to build the one-way interactions between Process Choreographer and the application components as SOAP/JMS messages.

13.2.4 WebSphere Business Integration Message Broker

WebSphere Business Integration Message Broker includes a JMS node that makes integration with WebSphere Application Server using a SOAP/JMS bus much simpler.

WebSphere Business Integration Message Broker also has support for the SOAP schema, for importing WSDLs, and for aggregating SOAP/Http requests, which will greatly improve the productivity of the IT specialist responsible for building the broker component of the scenario.

WebSphere Enterprise Service Bus provides an alternative runtime mapping for the service bus component in the solution architecture. There will need to be further investigation of the solution to decide on the version 6 product mappings for the service bus. One area of improvement the IT architect will be looking for in the new WebSphere Enterprise Service Bus product is better integration with the tooling used to create services. The costs of deploying a service on a service bus need to be no greater than deploying a service in a point-to-point connection. The difference in deployment costs was one of the reasons why in version 5 the IT architect opted for a process-centric rather than a bus-centric product mapping.

13.2.5 WebSphere Business Integration Server Foundation

WebSphere Business Integration Server Foundation has been superseded by WebSphere Process Server version 6. This has some effect on the architecture of the scenario. WebSphere Process Server uses WebSphere Platform messaging, rather than having WebSphere MQ running as a messaging service. One implication is the point-to-point connection between WebSphere MQ Workflow and WebSphere Business Integration Server Foundation cannot be migrated automatically to WebSphere Process Server.

If you recall, in Section 5.5, "Step 3: Select and merge the runtime patterns" on page 176 we chose a process focused runtime pattern and this point-to-point connection between WebSphere MQ Workflow and WebSphere Business Integration Server Foundation was one of a number of point-to-point connections that should now be migrated. It makes sense to revisit the choice of runtime pattern, and decide to opt for the broker focussed (or ESB) pattern for version 6, rather than the process-focused pattern we used in version 5.

13.2.6 WebSphere Studio Application Development Integration Edition

WebSphere Integration Developer has replaced WebSphere Studio Application Development Integration Edition in version 6.

There are a lot of detailed improvements in the tooling which will make the organization of the development project easier, and development more productive. Three changes in particular are of note because they affect the way we built the scenario.

1. The Assign activity is now able to handle references to complex messages. This will eliminate the need to create transformations for the simple one-to-one message mappings in the process. The Assign activities in the BPEL exported from Modeler are more likely to be retained in the execution flow.

2. There is a new kind of project concept, rather like the idea of the Message set in the Broker, which enables WSDL files easily to be stored in a separate project and referenced from multiple flows. This encourages more modular flow development, and should reduce the opportunity for making deployment errors by mistakenly storing WSDLs in a folder that was not visible to the deployment wizard.

3. The integration of the test server, in WebSphere Integration Developer, uses the approach adopted in the Rational tooling products. There is close integration between the tooling and runtime server. This makes it simpler to test on the server runtime rather than creating a special embedded test server.

   One immediate benefit is that it is a easier to apply maintenance to the run time server than the embedded test server, and there are potentially fewer server environments to manage. A second benefit is for the developer a smoother transition from unit test into a full scale test environment.

A final area to look for improvement is in the wizards that create transforms and java snippets that aggregate multiple input messages to produce one output message.

13.2.7 WebSphere Business Integration Modeler

Modeler version 6 is an evolutionary change from version 5. The major area of enhancement, modeling business measures, comes outside the scope of this scenario.

There have been some minor changes that improve the integration of the Modeler with the rest of the solution, principally:

1. The BPEL that is generated to import into WebSphere Integration Developer is simpler to use.

2. The content of what is imported has been broken down into smaller pieces, so rather than import the whole model, it is possible, for example, to import the process and not the data model.

   This would immediately eliminate the first step that we took in the refinement of the Modeler BPEL in WebSphere Studio Application Development Integration Edition. We had to delete all the partner links and associated message structures as the interfaces were being imported from Rational Software Architect.

13.2.8 Rational Software Architect

We were already using Rational Software Architect version 6 in the scenario, and during the course of this Redbook, we upgraded to Fixpack 1, which improved the stability of the integration with the WebSphere Business Integration Modeler feature.

The functionality we were most in need of developing in Rational Software Architect was a transformation between UML and WSDL. It would have been possible for us to create our own transformation plug-in using the extensibility of Rational Software Architect that is described in *Patterns: Model Driven Development using Rational Software Architect*, SG24-7105.

Check whether IBM, or another vendor, has developed the plug-in. There are papers that discuss the issues of transforming between UML and WSDL. See for example, *IMS General Web services UML to WSDL Binding Auto-generation Guidelines*, found here:

```
http://www.imsglobal.org/gws/gwsv1p0pd/imsgws_transfv1p0pd.html
```

Part 4

Appendixes

There are two appendixes. Appendix A, "Additional material" on page 501 describes the resources provided for you to work through the examples. Appendix B, "Integration considerations" on page 505, is provided by Jim Amsden, the author of the WebSphere Business Integration Modeler integration with Rational Software Architect. It explains the reasons behind how Jim designed the integration.

499

Additional material

This redbook refers to additional material that can be downloaded from the Internet as described below.

Locating the Web material

The Web material associated with this redbook is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser to:

`ftp://www.redbooks.ibm.com/redbooks/SG246636`

Alternatively, you can go to the IBM Redbooks Web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the redbook form number, SG246366.

Using the Web material

The additional Web material that accompanies this redbook includes the following file:

File name   *Description*
SG24-6636.zip   Zipped materials

The instructions on using the Web material are in the body of the Redbook.

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space: 60GB
Operating System: Windows 2000 or later.
Processor: Intel Pentium M 2GHz or equivalent
Memory: 2GB minimum

The materials are organized around running the tooling in a native Windows machine and splitting the runtimes into two VMware images which can run either on the same machine or on an ethernet connected LAN. It is possible to run the configuration on a Thinkpad T42p with the resources described about, but it is tight as with one tool and two VMs running the memory demand is just over 2GB.

How to use the Web material

Unzipping SG24-6636 will create six top level folders containing materials for each tool.

► Broker
► Modeler
► RSA
► WAS
► Workflow
► WSADIE

> **Important:** Because of long path and filenames it is possible if you use the Windows built in Extract... wizard for compressed folders you will not reliably recreate the uncompressed contents. It will probably work.
>
> The compressed folders have been uncompressed and then tested using the WINZIP utility successfully. It is important that you unzip the folders into a short path, preferably a drive root directory, before moving the folders somewhere more convenient.
>
> If you are not familiar with the Windows **SUBST** command to create an abbreviated path, try searching for help on **SUBST** using the Windows help wizard to help you plan where and how to decompress the materials.

The Workspace subfolders in all the tools are empty because they would add hundreds of megabytes to the .zip file. All the workspaces have been compressed in a tool specific manner and the .zip files stored in project or zipped workspace directories.

With the exception of the Broker the utility to unzip a compressed workspace is part of the tool, though the terminology and procedure to reload the workspace varies. In Rational Software Architect and WebSphere Studio Application Development Integration Edition for example, the workspace is stored as a Project Interchange file.

In the case of the broker there is no utility in the workbench to export and import workspaces. However, if you use the WINZIP utility and uncompress the zipped workspaces to a root drive you will be able to reload the compressed workspaces successfully. The **Tip**: on page 295 describes how to invoke a broker workspace from a nondefault location in the file system.

B

Integration considerations

WebSphere Business Integration Modeler currently supports model export for import into XDE models that have the Business Modeler profile applied. This appendix describes the philosophy, requirements, design, and implementation of integration between WebSphere Business Integration Modeler / BOM (Business Object Models) , and Rational Software Architect / UML2 models.

Integrating WebSphere Business Integration Modeler andRational Software Architect

The integration of WebSphere Business Integration Modeler and Rational Software Architect / Rational Software Modeler is based on the translation of BOM models to UML2. This translation treats the BOM models as a contract specifying what implementations in Rational Software Architect need to do. The WebSphere Business Integration Modeler models are not treated as something that is elaborated or transformed into implementation, but rather something that specifies the contract for what an implementation must do, not how it does it.

The contract is made up of two parts. The static part, or usage contract, specifies what clients must know in order to use services implemented in UML2 using Rational Software Architect. These are the tasks that are invoked in the BPEL process generated from WebSphere Business Integration Modeler. The usage contract consists of the provided and required interfaces. From WebSphere Business Integration Modeler these are extracted from the requiredRoles assigned to tasks in the business processes. For each task in a process assigned to a role, we create an interface for that role and add an operation corresponding to the invoked task. Note that not all the business process will be realized, nor do all operations in all roles need to have direct implementation. However, if a process is automated (and therefore has a corresponding BPEL process that orchestrates the responsibilities of the required roles), all of the tasks in that process have to be invoked somehow. That is, they must have some realization in Rational Software Architect, even if that implementation is only sending an Email, SMS message, task item in an agenda, a temporary alarm pop-up, or an entry in a workflow inbasket. So the interfaces corresponding to the BOM roles form a nice way to capture the usage contract in Rational Software Architect.

The realization contract is the business process itself. The best way to model this in UML2 is to create a Collaboration containing an Activity that is the UML2 version of the business process. The roles in the collaboration correspond to the requiredRoles in the BOM model. These are the roles that are assigned to each task in the business process. In UML2, the roles correspond to an InputPin that is the target of a CallOperationAction. The name of the input pin is the name of the BOM resource requirement while its type is the Interface corresponding to the role containing the invoked operation. The CallOperationAction::operation is also set to the operation in the role interface.

Usecases corresponding to business processes from the BOM model are also created in order to provide a higher level view of the business requirements and a place to specify non-functional characteristics. These use cases are realized by the collaboration generated for the BOM business processes. There is no need

for additional, nonstandard dependencies, nor is there any missing traceability between the BOM and UML2 models.

The Collaboration can then have any number of Classifiers that contain CollaborationOccurrences which bind parts to the roles in the collaboration thereby providing an implementation that fulfills the specified contract. There are no constraints on how these classifiers are organized. They will generally have to address other system use case, architectural, and nonfunctional concerns over and above the functional requirements specified by the Collaboration contract. As a result, there is a large degree of freedom about how the implementation is modeled and transformed into an implementation. But, in the final implementation, the required interfaces specified in the contract, and generated from the required roles in the BOM model will be realized somehow. As a result, the BPEL generated from the BOM model will be able to easily invoke the required tasks.

This approach is more complicated than what was done for the import/export between WebSphere Business Integration Modeler and XDE. However, it provides a formal way of defining the integration between the products that leverages the capabilities of both while minimizing the overlap and duplication (and resulting reconciliation problems). The Rational Software Architect model will be able to directly refer to Collaboration, Interfaces, and Operations in the BOM model without copying. The BOM model can be read-only from the Rational Software Architect model since there is no need to change the specification contracts in order to build an implementation. Any changes in the BOM model by WebSphere Business Integration Modeler will be immediately visible in the Rational Software Architect model eliminating any transformation or reconciliation errors or latency. This might require refreshing the shortcut, or exiting Rational Software Architect and restarting. We're looking at how to include the WebSphere Business Integration Modeler models in the Rational Software Architect editor so that it is automatically notified of changes. But this is not likely.

Note also that this integration strategy is consistent with the Enterprise Application Architecture profile used for modeling complete business applications, including architectural concerns. As a result, WebSphere Business Integration Modeler can be used to create business models that define the contracts for business applications, while EAA can be used to capture those contracts in standard UML2 allowing the reuse of emerging transformations to WebSphere platforms.

The specific requirements are:

1. Configure Rational Software Architect to be able to load the WebSphere Business Integration Modeler 5.1.12 resources.XMI file and transform it into

UML2 with the UML BM profile applied. This eliminates the need for export and import operations and makes the tools better integrated.

2. Automatically apply the BM to imported models.

3. The loader will read WebSphere Business Integration Modeler files using a custom EMF resource, and convert them into UML2 so the WebSphere Business Integration Modeler / BOM models can be views from the Rational Software Architect model explorer and integrated with other UML2 models.

4. The translated BOM model may be saves as a standard Rational Software Architect .emx (XMI2.1) file extended with the Rational Software Architect implementation of the UML BM (Business Modeler) profile in plug-in com.ibm.xtools.buzmodeler.

5. Developers do not have to have WebSphere Business Integration Modeler installed. There must not be any additional coupling between Rational Software Architect and WebSphere Business Integration Modeler. Rational Software Architect cannot require the WebSphere Business Integration Modeler meta model to be installed.

6. Loads only the WebSphere Business Integration Modeler BOM model elements, not any diagrams.

7. SupportWebSphere Business Integration Modeler Advanced Version 5.1 and 5.1.1.

Business Process and Application Development Use Case

This use case is not intended to address all aspects of business process modeling, integration with object modeling, or application generation and deployment. It is intended to give a broad overview of those subjects in the context of WebSphere Business Integration Modeler and Rational Software Architect integration, and represent a simple, but typical end-to-end development process.

Use case Steps:

1. UseWebSphere Business Integration Modeler to discover and capture key business processes.

   – Capture business data items that are exchanged between processes and tasks.

   – Assign tasks to roles

   – Determine required resources

   – Organization

2. Simulate the business processes for validation and to determine optimal resource allocation.

3. Determine opportunities for software automation.

4. Use Rational Software Architect to create an implementation model.

5. Open the WebSphere Business Integration Modeler model in Rational Software Architect to view the business contract.

6. Realize the contract using facilities of Rational Software Architect.

7. Generate and deploy the Rational Software Architect implementation.

8. Generate the process choreography using WebSphere Business Integration Modeler.

9. Bind theWebSphere Business Integration Modeler tasks with their implementations produced by Rational Software Architect.

Abbreviations and acronyms

.Net	"dot Net" Web service platform on Windows (Microsoft)	**IBM**	International Business Machines Corporation
ABD	Asset Based Development	**IDE**	Integrated Development Environment
ANT	Another Neat Tool	**IDL**	Interface Definition Language
B2B	Business to Business	**IT**	Information Technology
B2C	Business to Consumer	**ITSO**	International Technical Support Organization
BDD	Business Driven Development	**J2EE**	Java 2 Enterprise Architecture
BM	Business Model	**JMS**	Java Messaging Service
BOM	Business Object Model	**JNDI**	Java Native Directory Interface
BPEL	Business Process Execution Language	**LAN**	Local Area Network
BPEL4WS	Business Process Execution Language for Web service	**MDA**	Model Driven Architecture
BPM	Business Process Management	**MDD**	Model Driven Development
		OAG	Open Applications Group
CICS	Customer Information and Control System	**OMG**	Object Management Group
CIM	Computer Independent Model	**PIM**	Platform Independent Model
CTG	CICS Transaction Gateway	**PSM**	Platform Specific Model
DB/2	Database (IBM)	**RMI-IIOP**	Remote Method Invocation-Internet Interoperability Protocol
DMZ	Demilitarized Zone		
EAI	Enterprise Application Integrations	**RSA**	Rational Software Architect
EDI	Electronic Data Interchange	**RUP**	Rational Unified Process
EIS	Enterprise Information System	**SOA**	Service Oriented Architecture
EJB	Enterprise Java Bean	**SOAP**	Simple Object Access Protocol
ESB	Enterprise Service Bus	**SOI**	Service Oriented Integration
ESB	Enterprise Service Bus	**SQL**	Structured Query Language
FDL	Flow description language	**SSL**	Secure Sockets Layer
Http	Hypertext Transfer Protocol	**UML**	Unified Modeling Language
Https	Secure Http	**WBI**	WebSphere Business Integration

511

WSDL	Web services Description Language

Related publications

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

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 514. Note that some of the documents referenced here may be available in softcopy only.

- ► *"WebSphere and .Net interoperability using Web services"*, SG24-6395
- ► *"Continuous business process management"*, SG24-6590
- ► *Using a Single Business Pattern with the Rational Unified Process (RUP)"*, REDP-3877-00
- ► *Patterns: Model Driven Development using Rational Software Architect*, SG24-7105
- ► *"On demand Operating Environment: Creating Business Flexibility"*, SG24-6633
- ► *"Continuous business process management"*, SG24-6590
- ► *"BPEL4WS Business processes with WebSphere Business Integration: Understanding, Modeling, Migrating"*, SG24-6381
- ► *"WebSphere Business Integration Server Foundation 5.1 Handbook"*, SG24-6318

Other publications

These publications are also relevant as further information sources:

- ► *"The inmates are running the Asylum"*, Alan Cooper, 1999. ISBN 0-672-31649-8, SAMS - Macmillan publishing.
- ► "The Unified Modeling language User Guide", Grady Booch, James Rumbaugh and Ivar Jacobson. Addison-Wesley, 1999 ISBN 0-201-57168-4

Online resources

These Web sites and URLs are also relevant as further information sources:

► Sue Bayliss and Larry Yusuf , *"Merging disparate IT systems, Part 1: Use business process management to create integrated solutions"*, IBM developerWorks, found at,

http://www-106.ibm.com/developerworks/ibm/library/i-merge1/

► Sue Bayliss and Larry Yusuf , *"Merging disparate IT systems, Part 2: Understand the claim system"*, IBM developerWorks, found at,

http://www-128.ibm.com/developerworks/library/i-merge2/

► Donald Light, *"Deriving insurance business value from business process management tools"*, Building an Edge, Vol. 5, No. 11, November 9, 2004,

http://www-1.ibm.com/industries/financialservices/doc/content/bin/bae_nov_2 004.pdf

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Build a Business Process Solution Using Rational and WebSphere Tools

IBM®

Build a Business Process Solution Using Rational and WebSphere Tools

Redbooks

Explore IBM On Demand Business and business-driven development

Learn to use modeling, UML, and BPEL

Study implementation and integration

This IBM Redbook is based on the experiences of a team in the IBM Hursley laboratory. They built an auto-claim insurance solution to put the WebSphere software platform through its paces. The team worked with WebSphere developers to use the experience of building the solution to improve the design of WebSphere version 6 platform products.

They thought it would be valuable to share their experiences with a wider audience. The result is a tour de force, showing how the team went about using IBMís software development platform to understand business requirements and then architect, design and build the solution.

Their experiences will help you plan, design and build a business driven development solution using products from IBMís WebSphere Business Integration portfolio.

This redbook is written from the perspective of three types of developer: the business analyst, the software architect, and the IT specialist. Individual chapters in the book show how each member of the team developed their part of the solution, and how the team integrated the solution together.