WebSphere software

IBM

An End-to-End SOA Integration Scenario using IBM WebSphere Process Server on z/OS

- Development and deployment of a business process
- Reuse of CICS and DB2 assets as services
- Using WebSphere Message Broker as ESB

Alex Louwe Kooijmans
G Michael Connolly
Loraine Arnold
Mikhail Egorov
Marianne Menå Heltborg
Fatima Otori
Brian Paskin

Redpaper

ibm.com/redbooks

International Technical Support Organization

An End-to-End SOA Integration Scenario using IBM WebSphere Process Server on z/OS

March 2009

REDP-4391-00

Note: Before using this information and the product it supports, read the information in "Notices" on page vii.

First Edition (March 2009)

This edition applies to the following products:

- WebSphere Process Server Version 6.1.0.0 for z/OS
- WebSphere Application Server Version 6.1.0.15 for z/OS (for WPS)
- WebSphere Message Broker Version 6.0 for z/OS
- WebSphere Portal Version 6.0 for z/OS
- CICS TS Version 3.1
- WebSphere Integration Developer Version 6.1.2
- Rational Application Developer Version 7.0
- WebSphere Message Broker Toolkit Version 6.0

This document created or updated on March 5, 2009.

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

Trademarks

Preface

This paper demonstrates an end-to-end SOA integration scenario on z/OS® following a simple business scenario. The paper discusses the architecture, introduces the products used, and describes how the scenario was developed.

Key products used are WebSphere® Portal, WebSphere Process Server, WebSphere Message broker and CICS®.

The team that wrote this paper

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

Alex Louwe Kooijmans is a project leader with the International Technical Support Organization (ITSO) in Poughkeepsie, NY, and specializes in WebSphere, Java™ and SOA on System z® with a focus on integration, security, high availability and application development. Previously he worked as a Client IT Architect in the Financial Services sector with IBM® in The Netherlands, advising financial services companies on IT issues such as software and hardware strategy and on demand. Alex has also worked at the Technical Marketing Competence Center for zSeries® and Linux® in Boeblingen, Germany, providing support to customers getting starting with Java and WebSphere on zSeries. From 1997 to 2002, Alex completed a previous assignment with the ITSO, managing various IBM Redbooks® projects and delivering workshops around the world from Poughkeepsie.

G Michael Connolly is an IT consultant at the ITSO, Poughkeepsie Center. He has more than 30 years of IBM software development experience in both distributed systems and the mainframe zSeries. He holds a BA in Humanities from Villanova University. His areas of expertise include TCP/IP communications, UNIX® System Services, EWLM, and WebSphere for z/OS.

Loraine Arnold is an Advisory Programmer who joined IBM in 1982. She has worked primarily on IBM mainframe systems as a systems programmer supporting MVS, IMS, DB2®, WebSphere MQ, and WebSphere Application Server. She has also worked in the software development area as a developer, tester, and change team support person. Currently Loraine works in the z/OS Integration Test area supporting application development for I/T workloads using WPS, WAS, MQ, and IMS.

Mikhail Egorov is an IT Specialist at IBM Software Group, Russia. He holds a Bachelor and Master's Degree in Computer Science from Moscow State Bauman University. In 2007 he worked at the ITSO developing and delivering various SOA on System z workshop materials and labs. Currently Mikhail works as a Technical Sales Specialist covering the WebSphere for z/OS product family. His area of expertise also includes Rational® Developer for System z, Rational Business Developer, and other enterprise modernization tools.

Marianne Menå Heltborg is an IBM certified IT specialist in IBM Software Group, Denmark. She joined IBM in 1986 and originally worked as a CICS Systems Programmer. Her areas of expertise include the CICS TS, the WebSphere product family, Rational Developer for System z, and Web enabling applications on z/OS

Fatima Otori is an IBM System z WebSphere IT Specialist based in Chicago, IL. She holds a Bachelor of Science Degree from Tulane University and a Master's Degree from the University of Michigan Ann Arbor, both in Electrical Engineering. In her current role, Fatima is actively engaged in implementing critical success paths to SOA for several major clients, particularly in the areas of Application Modernization, Application Integration, and overall Business Process Improvement and Management. Her areas of concentration include WebSphere Application Server, WebSphere Process Server, WebSphere Integration Developer, WebSphere Enterprise Service Bus, WebSphere Message Broker, WebSphere Services Registry and Repository, and WebSphere Business Services Fabric.

Brian Paskin is a Senior IT Specialist for IBM Software Services for WebSphere for IBM USA. He has 15 years of experience with IBM, predominantly with WebSphere AS, WebSphere MQ, CICS, and aggregated system design. He currently specializes in performance tuning WebSphere Application Server and WebSphere MQ on both distributed and System z platforms. He previously worked for IBM Germany and IBM Italy.

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

Rich Conway
International Technical Support Organization, Poughkeepsie Center

SeungKeon Chung
IBM South Korea, Software Group

Mike Poirier, Dave Bonaccorsi
IBM Middletown, RI

Don Bagwell, John Hutchinson
IBM Washington Systems Center

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will 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 papers to be as helpful as possible. Send us your comments about this paper or other IBM Redbooks publications in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

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

1

Introduction

This chapter is a quick introduction to service-oriented architecture (SOA), a positioning of the building blocks in an SOA and the specific value of having those building blocks on IBM System z and z/OS in particular.

We begin by illustrating how the view of application integration has developed over time. We discuss IBM's SOA Reference Architecture, which we then refer to throughout the remainder of this paper. We conclude this chapter with a summary of required deployment characteristics, which are the starting point for explaining the strengths and value of the z/OS platform.

The ultimate goal and essence of this document, however, is not to have a theoretical discussion of SOA and its components, but to show an example of an SOA scenario developed for and running on z/OS. The rest of this document evolves as follows:

- In Chapter 2, "Building an SOA scenario on z/OS" on page 15 we describe our example.
- In the following chapters we describe how we enabled existing assets for reuse in our SOA scenario:
  - Chapter 3, "Customer inquiry service" on page 33 discusses service-enablement of an existing CICS Customer inquiry program.
  - Chapter 4, "Credit check service" on page 55 discusses service enablement of an existing DB2 stored procedure by using an ESB (WebSphere Message Broker).

  - Chapter 5, “Order placement service” on page 69 discusses service enablement of an existing CICS Order application (CICS Catalog Manager application).
- In Chapter 6, “Creating the Business Process” on page 85 we explain step by step how to create a business process reusing various services.
- In Chapter 7, “Deploying and running the process in WPS on z/OS” on page 185 we describe deployment of the business process to z/OS.
- Finally, in Chapter 8, “Adding a Portal user interface” on page 203, we show how you can add a Portal interface to the business process.

1.1 Defining SOA

Service-oriented architecture (SOA) is a system architecture in which application components are built as *services*. These services have well-defined interfaces and are loosely coupled, which provides flexibility in inter operability and reuse. Service definitions can be based on a direct mapping of business processes to IT systems. In that way, an SOA can offer closer business and IT alignment than earlier integration architectures.

From a business perspective, an SOA is a set of flexible IT components that can be used to support composable business processes. From a technical perspective, an SOA is a set of IT services that can be called to perform a specified operation.

1.2 Evolution of Enterprise Application Integration

There are many views of how programming paradigms have evolved towards SOA as we know it nowadays; the view depicted in Figure 1-1 on page 3 is just one.

SOA has come from two somewhat independent but related developments:

- Evolution of the programming paradigm towards increased reusability.
- Evolution of program interoperability through a shift towards increased integration of application.

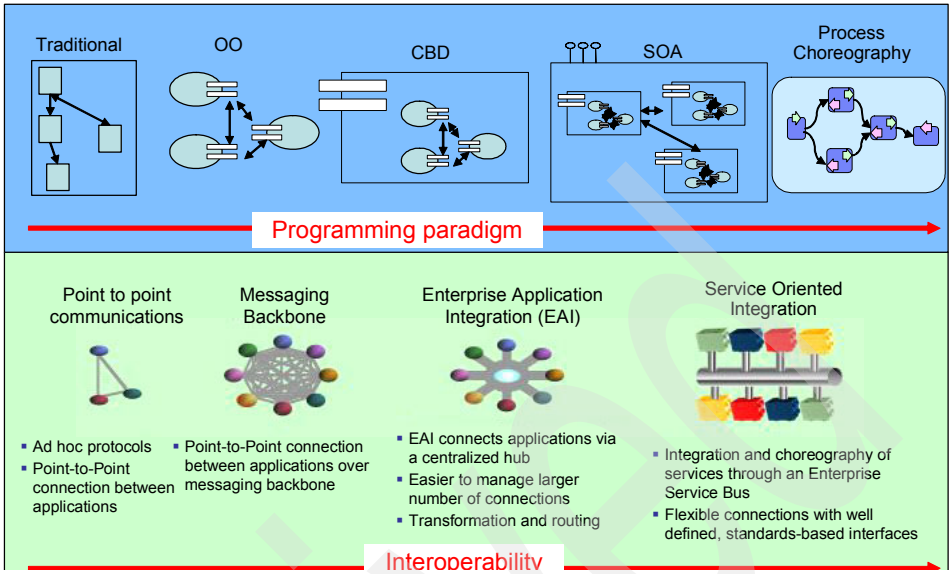

Figure 1-1 Evolution towards SOA

1.2.1 Programming paradigm shift

In the early days programmers were mainly concerned with structuring programs into logical procedures, which were tightly coupled and rarely reused. Communication and resource access was often more complex than the actual business logic implemented.

Modularizing brought further program structuring and reuse into play, and the object-oriented programming paradigm provided new capabilities for reuse. Component-based development (CBD) has become the general term for design and programming techniques concerned with reuse and separation of concerns.

Recently IBM has announced *Service Data Objects (SDO)* and *Services Component Architecture (SCA)* as their next steps, and has thereby introduced an SOA programming model that simplifies program construction and enables composite application development.

1.2.2 Interoperability shift

At the same time, program interoperability became an issue as programs increasingly needed to communicate with one another. *Messaging middleware* relieved the programmer of a lot of communication worries and provided a further de coupling of application components. *Message brokering* products were the

next step in application integration, providing tools for message transformation and aggregation facilities.

The SOA paradigm provides application designers with standardized intercommunication options and reuse of services at all levels, up to the business service level.

What this development means to application designers is that where they used to spend a lot of time on low-level aspects of system operation and inter operability, they now can focus primarily on the business problem space.

Business process choreography brings a further development into the integration picture, allowing a business modeler to assemble business processes from automated and human services.

1.3 Technologies underpinning an SOA

Before jumping into the IBM SOA Reference Architecture, we highlight the key technologies that can be used today to implement an SOA.

1.3.1 Web services and related technologies

Web services are a set of specifications that define a standard for system-to-system communication. Through Web services, applications can cooperate by invoking well-defined interfaces. The interfaces hide implementation details of the service, making it possible for applications based on different technologies, running on different platforms, to participate in higher level composite applications.

Web services standards use other technologies, such as XML and HTTP, and offer the de facto standards for the implementation of an SOA. The most important Web services standards are discussed now.

The *Web Services Description Language (WSDL)* is a series of XML standards that define the interface of each service. The WSDL of a service is highly reusable and should be kept in a place easily accessible for application developers.

The messaging protocol used as part of the Web services standards is called the *Simple Object Access Protocol (SOAP)*. It is a standard for defining the format of a request/reply that is sent to and returned from a service. Note that SOAP does not provide the transport infrastructure. You still need to use a transport protocol underneath. Currently, the SOAP protocol can be used over HTTP or JMS.

There are some protocols that deal with security for Web services. The Web services security specification, commonly named *WS-Security*, is based on a token architecture for secure communications. Built on this base we can find specifications like *WS-Policy*, which defines the rules on how services interact, and *WS-Trust,* defining the trust model of a secure exchange. There are others as well.

The Web services specification also addresses quality of services (QoS) issues. For transactionality there is *WS-Coordination*, *WS-BusinessActivity*, and *WS-AtomicTransaction*. Reliable messaging protocols have been defined in *WS-ReliableMessaging*.

1.3.2 Enterprise Service Bus (ESB)

The communication between Web services in an SOA environment is built on the *Enterprise Service Bus (ESB)*, which is a common distributing network for services to work with. The ESB, as the heart of the SOA environment, has to be very reliable, available, and secure.

An ESB enables applications to create service interfaces for new or existing application functions, either directly, or indirectly through so-called *adapters*.

Besides that, the ESB can transport and route messages between service requesters and providers.

The ESB typically provides the following:

- Support for transport of messages between services over several protocols such as JMS messaging and HTTP.
- Transformation and routing of service requests.
- Event handling.
- (Web) services standards support.
- Support for new applications that are based on services and through that for technologies such as J2EE™, .Net, and so on.
- Support for all required existing applications, programming models, and data formats.

1.4 IBM SOA Reference Architecture

The IBM SOA Reference Architecture presented in Figure 1-2 includes the building blocks we described in the previous section. In addition to the primary functional building blocks, the architecture includes:

- Supporting building blocks that allow integration with existing and partner systems
- Building blocks providing additional management and monitoring functions
- Infrastructure services to address other non-functional requirements

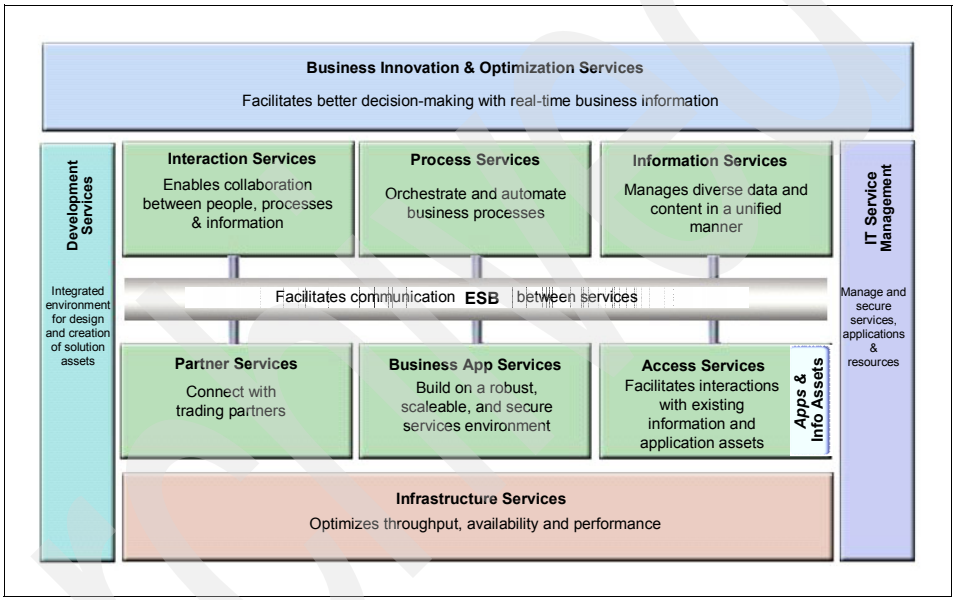

Figure 1-2 IBM SOA Reference Architecture

The IBM SOA Reference Architecture describes the key capabilities that are required for comprehensive, enterprise-wide SOA solutions:

- Development Services
- Enterprise Service Bus (ESB)
- Interaction Services
- Process Services
- Information Services
- Partner Services
- Business Application Services

- Access Services
- IT Service management Services
- Infrastructure Services

Our scenario, further described in 2.1, “Our example” on page 16, touches various areas, circled in Figure 1-3.

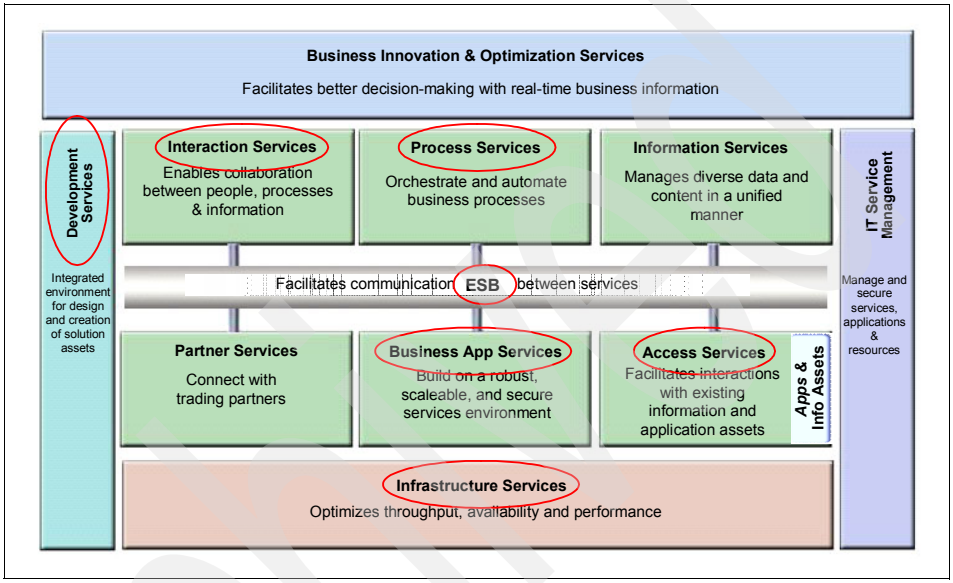

Figure 1-3 IBM SOA Reference Architecture with solutions for z/OS

1.4.1 How the IBM SOA Reference Architecture addresses IT requirements

The translation of building blocks into technical components and real hardware and software is primarily guided by functional and non-functional requirements. The IBM SOA Reference Architecture addresses this through the definition of building blocks that cover functional and non-functional requirements.

Business and IT alignment

Process services provide control over the flow of business processes composed of IT services and human interactions. Process services provide as close a translation of business processes to IT processes as possible, including integration with human processes, and also provide the means to monitor and adjust these processes. Partner services allow integration of business processes with partner processes.

The infrastructure supports this by providing appropriate IT service management services that allow monitoring of the IT services making up these business processes at the right level of granularity. The infrastructure must also provide for scalability, including dynamic resource allocation to processes that need more resources for business reasons.

By providing a close business process to IT service translation, and in addition by providing the means to integrally measure and manage these services and processes both at the business and IT level, an SOA allows for tighter business and IT alignment than we have ever seen before.

Ease of integration

Many existing integration projects are still based on proprietary integration technologies. In order to provide for vendor independence and smooth integration with new or existing platforms, new IT systems and communication protocols must be based on open standards.

Ease of integration is supplied by a standards-based integration platform. This integration platform allows integration of the existing set of diverse IT assets in the organization with newly built applications and business processes. The standards it is based on, like Web services, allow smooth integration with partners and future systems.

The ESB is the backbone for allowing cooperation among IT services through standards. The other building blocks provide access to that services backbone using the same standards, or can rely on conversion services provided by the access services building block, or on transformation and conversion services provided by the ESB.

Corporate IT view and governance

The current call for a corporate view of IT and governance arises from the need to align the IT and business value chains. Governance provides the structure to prioritize business objectives and set out the IT strategy.

The need for IT governance is not new in an SOA, but it has become increasingly important since the need for integration of applications threatened to turn the IT landscape into “spaghetti.” However, an SOA can only provide its unique values if sufficient attention is paid to governance; otherwise, SOA will only be a new way to integrate applications.

An SOA provides a number of facilities that support a business-level view of automated business processes. The concept of an SOA and the business process management focus it provides in the process services building block allows business analysts to model business processes and decompose them to automated and manual components. Business processes can be designed and

implemented starting from the business perspective and driven by the business. Process services and business process modeling services allow for reuse of existing IT assets in new business processes. Armed with these services, the business manager has the ability to make decisions on a corporate level with regard to reuse of services, and as a consequence, reuse of IT systems supporting those services. Business process monitoring functions allow monitoring of the business processes designed, as well as any consequent changes of the processes due to new or changed business requirements.

The IT service management services in the infrastructure provide support for this enterprise model in the form of management and monitoring services of the IT systems in relation to the business processes they support.

IT asset reuse

Existing IT assets, whether existing programs or data, can be reused and therefore be made accessible from standards-based technologies to the composable business process implementation provided by process services. Reuse of the investments made in existing IT assets is therefore provided by access services.

These services must be made as reliable and efficient as possible through platform optimization on the platform where the assets are hosted because reuse of these assets will in practice lead to intensified use of them, thereby increasing the requirements with regards to quality of service.

1.5 QoS and other infrastructure requirements

In the previous section we focused on how the SOA reference architecture addresses the functional requirements. A number of non-functional requirements (NFRs) are imposed on the architecture that are less explicitly addressed.

Cost effectiveness and transparency can only be delivered if the infrastructure and service management environment provide the means to measure cost of IT on a sufficiently granular level in all different components making up the infrastructure. This imposes special requirements on infrastructure services and IT service management services. These building blocks must provide the means to measure the demand of individual services and components and assemble these figures into business-level reports.

The business process demands a certain *reliability*. The IT systems supporting the business processes must be able to provide sufficient reliability to support the business.

Reliability is a quality that is provided by services in the infrastructure in the form of:

- Predictable and consistent *performance* in accordance with business process requirements. This is especially important in an SOA because the services provided in an SOA can increasingly be accessed by different consumers, and have different characteristics with regard to fluctuations in demand.
- *Capacity* is related to performance consistency. The IT infrastructure must be scalable when demand requires this, in stressing conditions such as peak transaction volumes, but also in support of, for example, marketing campaigns that might attract a large number of potential customers and might dramatically increase capacity in both front-end systems and back-end systems throughout the SOA.
- Operating environment *reliability and stability*. Here we mean technical reliability in the form of high Mean-Time-Between-Failure, and the ability to recover from failures without interrupting the IT operation. In an SOA the availability of a service is dependent on all the components making up that service. This will increase the availability requirements for the operating environments on which these services reside.
- *Integrity*. The loose coupling that characterizes an SOA may have the consequence of loosening control over the atomicity of transactions spanning several services making up a business process. Techniques must be provided to support atomic transaction capability.
- *Security*. The security implications of an SOA put special requirements on the technology the SOA is deployed on. The IT systems and infrastructure must provide the level of security required by the business process it supports. This comprises all security services at levels in accordance with what the business process requires, such as authentication, authorization, non-repudiation, confidentiality and encryption, privacy, and so on.

1.6 SOA and z/OS

SOA is an architecture model, independent of platform, technology, and vendor, but when we come to the decision where to deploy SOA applications and the required infrastructure, we have to consider the different aspects and Quality of Services (QoS) provided by each platform and by each vendor.

In this section we discuss the particular characteristics and QoS provided by the IBM System z platform and the z/OS operating system when deploying the SOA model.

In the previous sections we identified the requirements that an SOA must address. In this section we show how the required components are mapped to technologies, and how the building blocks can be deployed on z/OS, thereby addressing the non-functional requirements in a unique way.

1.6.1 How z/OS addresses SOA non-functional requirements

In this section we describe the traditional strengths of the mainframe, focusing on the qualities the mainframe provides, thereby addressing the most important non-functional requirements of most mission-critical IT systems.

Background

The IBM System z has for more than 40 years successfully run the core IT systems of many successful businesses, from medium size to very large. During these years, IBM has consistently invested in the evolution of the mainframe's unparalleled technology. Mainframes have incorporated all the new technologies and computing models that have come around in the marketplace since they were first delivered, from the Centralized model to the Web model, from the Assembler and COBOL languages to Java. Today the mainframe provides all the necessary capabilities to form the backbone in an enterprise service-oriented architecture, and more than ever offers a unique proposition for the heart of an enterprise SOA deployment.

The IBM mainframe has become the computing industry benchmark. Mainframes are in use by thousand of enterprises worldwide, with trillions of dollars invested in applications and skills. The platform hosts a large portion of the total business transactions and data in the world, providing 24x7 availability and other unique qualities.

When we consider the new paradigms and value propositions of the SOA, and at the same time realize the value of existing IT assets and the qualities of the mainframe, the potential of the combination of the mainframe strengths and the SOA concepts becomes very clear. The combination can bring a fast return on investment when transitioning to an SOA, while building the SOA on a platform that provides the high QoS that an SOA requires. Although the IBM mainframe runs five different operating systems, z/OS, z/VM®, z/VSE™, z/TPF and z/Linux, in the following section we focus specifically on the z/OS operating system, the flagship mainframe operating system.

Traditional z/OS strengths

The generally recognized z/OS strengths fall into a number of broad categories:

- Centralized computing model
- Security

- Manageability
- Virtualization and workload management
- Reliability
- Scalability
- Availability
- Transaction processing
- Batch processing

The z/OS operating system provides a comprehensive set of capabilities and tools out-of-the-box that provide these qualities. The z/OS operating system is able to provide its unique quality of service in combination with System z hardware and Parallel Sysplex® capabilities.

Deploying SOA on z/OS not only brings the SOA functionality to the platform and the platform qualities to the SOA, but what is just as important, z/OS extends the functionality of the SOA components like WebSphere Application Server, WebSphere Process Server, CICS DB2, and WebSphere MQ, and brings a level of quality that no other platform is able to provide.

1.6.2 Positioning of specific components on z/OS

In this section we summarize the advantages of deploying the most important components in the SOA on the z/OS platform.

ESB on z/OS as the backbone for SOA

The ESB is the key infrastructure component in the SOA architecture. It is the intermediary through which all service communication runs. It therefore requires the highest level of availability, scalability, security, and performance.

When existing core application functions on the z/OS platform are integrated using the ESB, it will need to provide at least the same level of Quality of Service as the back-end services it accesses. This is one reason to position the ESB on z/OS. Co-locating the ESB on the same platform as the back-end services provides additional levels of security, availability, scalability, and integrity. This is a second compelling reason to deploy SOA on z/OS. The cost is another factor that will influence the decision where to deploy the ESB. Much of the ESB workload is Java workload that can be off-loaded to zAAP processors, which will lead to a very cost-effective configuration.

We must, however, emphasize that positioning the ESB is not a question of one platform or the other. Although it is possible to let the ESB on z/OS handle all requests, a logical ESB could be spread over multiple platforms. An ESB could

be positioned on a distributed server for those services that do not required the high QoS of the mainframe.

z/OS as the Process Engine for core business processes

In most companies the core business applications run on the z/OS platform because these application require the qualities this platform provides. What is more logical than to also run the Business Process Engine component of the SOA, materialized by the WebSphere Process Server product, on the same platform where the services it depends on are deployed?

In fact, the same arguments that apply for the ESB, also apply for the WebSphere Process Server positioning.

z/OS as the platform for core application services

New applications built on the J2EE platform that require the mainframe Quality of Service can be run in WebSphere Application Server for z/OS, thereby taking advantage of the cost-effectiveness of zAAP.

The traditional transaction managers IMS and CICS are also very well positioned for the development of new functionality. Both platforms can participate fully in an SOA by exposing their functionality as Web services, and CICS can also act as a consumer of Web services. This is a very attractive option for the large number of customers that have deep investments in mainframe applications, need the mainframe for its qualities and facilities like batch processing, and want to reuse these assets.

Leverage existing platform skills

SOA brings new opportunities to the table for existing z/OS skills. As Java grew more and more popular over time, COBOL programmers were more or less challenged to acquire this new skill in order to continue to be competitive in the marketplace. In the SOA model, however, it is not that important anymore in which language a service is written, but much more how easily the service can be integrated. Java still has the most natural support for the standards being used for SOA (such as XML and SOAP), but the gap is closing and COBOL, PL/1, or any other program can be as easily integrated into an SOA as any Java program.

2

Building an SOA scenario on z/OS

The purpose of this chapter is to provide an overview of our solution scenario and the technology used.

- In “Our example” on page 16, we introduce our sample scenario.
- In “Our environment” on page 19 we introduce the software products involved.

Subsequent chapters of this redpaper will further explore the usage of each software product in more detail.

2.1 Our example

In our example we designed a simplified order process. Our goal was to demonstrate the various SOA core middleware solutions on System z, and at the same time to demonstrate the reuse of existing back-end programs and data. Our business process follows these steps:

1. Enter data for the process, such as Customer number, item number, and quantity. Data entry is performed using a Portlet running in WebSphere Portal.
2. Start a process in WebSphere Process Server. The process performs the following steps:
   a. Check whether customer number is valid. This check is performed by invoking an existing CICS program which accesses a VSAM data set containing customer information.
   b. If the customer number is valid, a credit check is performed. The credit check is performed by a DB2 stored procedure accessing DB2 data. The stored procedure passes back an OK or not OK. The stored procedure in DB2 is accessed through an ESB, WebSphere Message Broker.
   c. If the credit check was positive, a request to place an order is sent to another existing CICS program. In fact, this activity consists of multiple CICS programs, but we used the Service Flow Feature in CICS to build one composite service out of those programs.
3. Pass the result back to the portlet in WebSphere Portal.

2.1.1 Overall software architecture

Figure 2-1 on page 17 shows the overall software architecture used for our scenario.

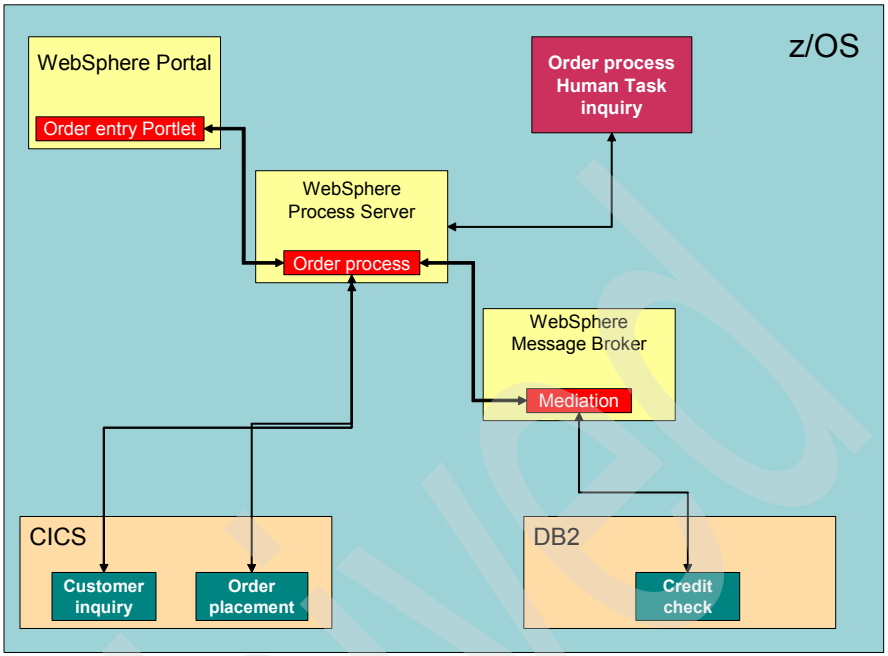

Figure 2-1 Overall systems architecture

The solution illustrated encompasses the following components:

- Order entry Portlet.
  The Order entry Portlet acts as the primary user interface for the Order process. The end user uses this Portlet to enter all information necessary to perform the Business process. The Order entry Portlet runs in WebSphere Portal on z/OS. Upon successful entry of all the information the Portlet will invoke the Order process in WPS using a Web Services call.
- Order process.
  The Order process is a business process modeled using BPEL and implemented in WebSphere Process Server on z/OS. The process includes a human task to resolve the situation of an unknown customer number. Several other services are invoked from this flow:
  - The Customer inquiry service in CICS
  - The Order placement service in CICS
  - A mediation in WebSphere Message Broker to call a DB2 Stored Procedure to perform a credit check
- Order process Human Task inquiry.
  There are several ways for the end user to inquire about outstanding Human

Tasks. One of them is the standard Business User Client application that comes with WPS.

- Customer inquiry service in CICS.
  This is a Web-service enabled version of a Customer inquiry program in CICS.
- Credit check service in DB2, accessed through WebSphere Message Broker. For this purpose a small mediation (message flow) has been implemented in WebSphere Message Broker.
- Order placement service in CICS.
  This is a composite Web service running in CICS, modelled using CICS Service Flow Feature.

2.1.2 Our approach

At a high level, the steps we performed to implement the processes shown in Figure 2-1 were as follows:

1. Web service enable the existing CICS Customer inquiry program using Rational Developer for System z.
2. Build and deploy a message flow for WebSphere Message Broker with a node accessing the DB2 stored procedure with the Credit check functionality.
3. Build and deploy a composite service with the Order placement functionality in CICS using Rational Developer for System z.
4. Develop the business process flow using WebSphere Integration Developer and reuse the services created in the previous steps.
5. Create and deploy a Portlet with a user interface for the business process.

In Chapter 3, “Customer inquiry service” on page 33 through Chapter 8, “Adding a Portal user interface” on page 203 we discuss these steps in more detail.

2.1.3 Products used

This section identifies the products we used to implement our scenario.

Development products used

- Rational Application Developer Version 7.0
  For developing the Portlet.
- Rational Developer for System z Version 7.1
  For developing the CICS service flow and enabling the back-end CICS programs as Web Services.

- WebSphere Integration Developer Version 6.1
  For developing the business process running in WebSphere Process Server.
- WebSphere Message Broker Toolkit Version 6.02
  For developing the mediation flow running in WebSphere Message Broker that accesses the stored procedure in DB2.

Runtime products used

- WebSphere Portal for z/OS Version 6.02
  Runs the Portlet that acts as the user interface for the business process.
- CICS TS Version 3.1.0
  Runs the customer inquiry transaction and the service flow that places the order.
- DB2 for z/OS Version 8.1.0
  Runs the Credit check stored procedure.
- WebSphere MQ for z/OS Version 6.0.0
  The underlying messaging backbone for WebSphere Message Broker on z/OS.
- WebSphere Process Server for z/OS Version 6.1
  WPS hosts the business process.
- WebSphere Message Broker Version 6.00
  Executes the mediation flow picking up the request from WPS, invokes the Credit check stored procedure in DB2, and passes back the result to WPS again.
- WebSphere Application Server for z/OS Version 6.0.2.19
  Acts as the underlying server for WebSphere Portal.
- WebSphere Application Server for z/OS Version 6.1.0.15
  Acts as the underlying server for WebSphere Process Server.

2.1.4 Sample material

Workspaces of the components developed and z/OS source components are available from the Web. Refer to Appendix A, “Additional material” on page 229 for download instructions.

2.2 Our environment

In the following sections we briefly discuss the products used in our scenario.

The core of our development environment and process consists of WebSphere Integration Developer (WID) Version 6.1.2. WID can be used for developing both a business process to run in WPS and mediation flows to run in WESB. In our scenario we use WID for developing a business process.

Our business process does not stand by itself. We are using a Portal front-end, an Enterprise Service Bus and CICS and DB2 back-end systems. The Enterprise Service Bus is implemented using WebSphere Message Broker Version 6.0.0 for z/OS.

2.2.1 Implementing a business process using WPS and WID

In this section we talk about the two products needed to develop and run business processes: WebSphere Integration Developer (WID) and WebSphere Process Server (WPS).

WebSphere Process Server on z/OS

IBM WebSphere Process Server (WPS) is the next generation business process integration server that has evolved from proven business integration concepts, application server technologies, and the latest open standards.

WebSphere Process Server enables deployment of standards-based process integration solutions in an SOA. The SOA framework is a conceptual description of the structure of a software system in terms of its components and the services they provide, without regard for the underlying implementation of these components, services, and connections between components. This means that a well defined set of business-level interfaces for the components can be created and maintained, and shielded from lower-level technology changes. Loosely coupled integration applications that are based on SOA provide flexibility and agility. You can implement integration solutions independent of platform, protocols, and products.

WebSphere Process Server is built on top of WebSphere Application Server and contains the best of the product features previously found in WebSphere MQ Workflow, WebSphere InterChange Server, and WebSphere Business Integration Server Foundation. By building on top of WebSphere Application Server Network Deployment, WebSphere Process Server can take advantage of all the mature capabilities it provides, such as clustering, high availability, embedded messaging and transaction management. WebSphere Process Server also includes WebSphere Enterprise Service Bus (WESB) to provide ESB functionality; however, this doesn't preclude using WebSphere Message Broker (WMB) as an ESB, a combination of the two, or extending the ESB with additional products.

WebSphere Process Server includes three layers as depicted in Figure 2-2.

The SOA core consists of Service Component Architecture (SCA), Business Objects, and the Common Event Infrastructure (CEI).

SCA presents all elements of business transactions—access to Web services, Enterprise Information System (EIS) service assets, business rules, workflows, databases, and so on—as service components. SCA separates business logic from implementation, so that you can focus on assembling an integrated application without knowing the implementation details. Service components can be assembled graphically in WebSphere Integration Developer (WID), and the implementation can be added later.

Business Objects define the data flowing between SCA components. Business Objects provide an abstraction for data access and are based on a data access technology called Service Data Objects (SDO). SDOs provide a universal means of describing disparate data. Business Objects provide rich features to map, manage and transform data to underlying IT and are described through standards-based XML schema.

The Common Event Infrastructure (CEI) allows service components to create events that can be captured by business monitors such as WebSphere Business Monitor for real-time monitoring of business processes.

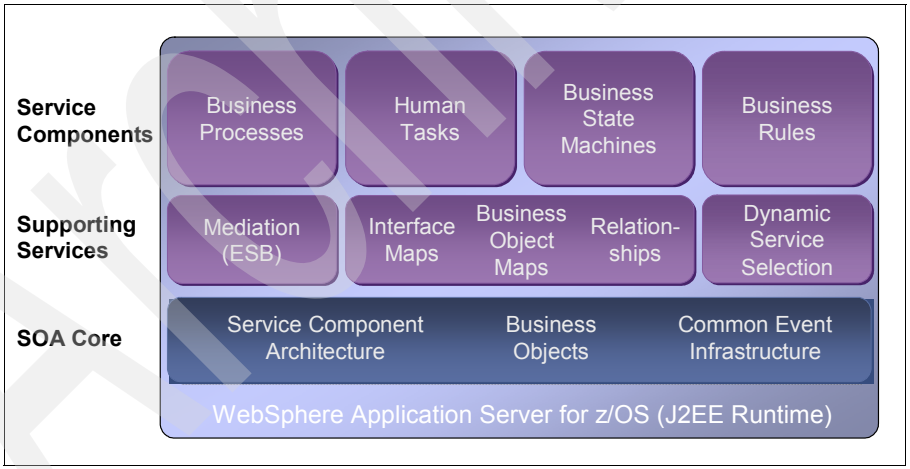

Figure 2-2 Architectural model of WebSphere Process Server

Supporting services are components that are needed in any integration solution, including data transformation and synchronization services.

WebSphere Enterprise Service Bus provides the ESB functionality, with mediation flows that can operate on messages to perform XML-based data

transformation, protocol transformation between various transports, custom Java operations, and routing.

Interface maps let you invoke components by translating calls to them. It is possible for interfaces of existing components to match semantically but not syntactically. This is especially true for components that already exist and services that need to be accessed.

Business object maps let you translate one type of Business Object into another type of Business Object. You can use these maps in a variety of ways, for example, in an interface map to convert one type of parameter data into another.

Relationships can be used to establish relationship instances between object representations of the same logical entity in disparate back-end systems. You may want to access the same logical entity within business integration scenarios, for example, the same customer's address might need to be updated in various back-end systems, such as an ERP system and a CRM system. These relationships can be established and managed automatically using the Relationships service component. These relationships are typically accessed from a business object map when translating one business object format into another.

Selectors can be used for dynamic selection and invocation of different services, which all share the same interface. WebSphere Process Server offers a Web-based interface to enable dynamic updates to the selection criteria and target services, which means that a module that has been deployed at a later time can still be called by this selector component, enabling dynamic changes to the integration solution.

WebSphere Process Server provides business processes, human tasks, business states machine, and business rules service components.

A business process component implements a Business Process Execution Language (BPEL) compliant process. You can develop and deploy business processes that support long and short running business processes and a compensation model within a scalable infrastructure. You can create BPEL models in WebSphere Integration Developer or import from a business model that you have created in WebSphere Business Modeler or any other modeling tool that supports the BPEL standard.

Human tasks are stand-alone components that you can use to assign work to employees. Additionally, the human task manager supports the ad hoc creation and tracking of tasks. WebSphere Process Server also supports multi-level escalation for human tasks, including e-mail notification and priority aging.

Business state machines provide another way of modeling a business process. Some processes are easily described as a sequential flow of activities, and they can be modeled as business processes. However, some processes are driven by events rather than a sequence, and in this case the business state machine is a better fit for modeling the process. One example would be an ordering process where you can modify or cancel the order at any time during the order process until the order is actually fulfilled.

Business rules are a means of implementing and enforcing business policy through externalization of business function. This enables dynamic changes of a business process for a more responsive business environment.

Using WID to develop a Business Process solution

This section provides more information on development of a business process using WID.

For examples of building business processes from beginning to end, see *Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6*, SG24-7205.

IBM WebSphere Integration Developer is the integration tool you use to author SOA-based services and choreograph them into business processes that you can deploy on IBM WebSphere Process Server.

Using the WebSphere Integration Developer tool set, business integration solutions can be created using simplified integration mechanisms, such as the Service Component Architecture (SCA) programming model and the Service Data Objects (SDO) data model. SDO business objects can be defined, transformed, routed, and mapped using SCA components. WebSphere Adapters supply connectivity to back-end Enterprise Information Systems (EIS).

WebSphere Integration Developer provides an *assembly editor* where the developer can group service components into *modules* and specify which service *interfaces* are exposed by the module to outside consumers. It also defines common *libraries* to store artifacts that will be shared across modules, such as interfaces or data maps. Modules expose their functionality by way of *exports*, and use the functionality of other modules using *imports*. Modules can be connected using *wires* to form complete integration solutions or composite applications. You will see examples of some of these elements as you go through the hands-on scenarios later in this book.

The following is a typical development flow for building a business process:

1. Create a new *business integration module* (or simply *module*) for the project. A module is a composite of service components, imports, and exports. A module can also contain the implementations and interfaces referenced by its

components, imports and exports, or these can be placed in other projects, such as a library project. Each module has a *module assembly*. The module assembly contains a diagram (referred to as the assembly diagram) that shows the components of the module and how their interfaces and references are wired together. The module is the basic unit of deployment for WebSphere Process Server.

> **Tip:** Artifacts like business objects and interfaces should be created within a library module and referenced by the business module. A library, like a module, contains resources and code for your applications.

Libraries allow you to share those resources and code between modules. However, unlike modules, libraries cannot be deployed by themselves.

2. Create *Business Objects* that describe how the components communicate with each other. Business Objects are containers for application data that represent business functions or elements, for example, customer or order data.
3. Create the *interface* for each component. The interface determines what data can be passed from one component to another. The interface consists of one or more operations, each operation defining the input and output to the component. An operation can be a request-response operation or a one-way operation.
4. Create the *service component*. A service component can be implemented as one of the following:
   - Java component
   - Business process (BPEL process)
   - State machine
   - Business rule
   - Selector
   - Human task

   In this publication we are focused on the business process. When you create the business process component, an inbound interface (*interface partner*) is assigned to the component. You can create the interface when you create the business process or select an existing interface. The process editor is used to build the business process:

   a. Create reference partners for the business process. Reference partners are outbound interfaces of the business process.

   b. Add activities to the business process.

Activities are the individual business tasks within the process that compose the larger business goal. For example:

- Add an Invoke activity for each call to a partner.
- Add a Human Task activity to send a process-related task out to a human for completion.
- Add a Choice activity to evaluate conditions and select a processing path.

5. Create exports and imports and bind them to a protocol.

   Import and export components define a module's external interfaces or access points. Imports identify services outside of a module so they can be called from within the module.

   An *import* is created in the assembly diagram by adding the export or interface of the external module to the assembly diagram.

   Exports allow components to provide their services to external clients. An export component is created in the assembly diagram for a component so that its business service can be used by other modules. Imports and exports require binding information that defines the means of transporting the data from the modules. You can change this binding later to use a different transport.

6. Test the business integration module in the integrated test environment.

7. Deploy the business module to WebSphere Process Server.

When you deploy a module to the test environment or to WebSphere Process Server, WebSphere Integration Developer packages the module as a J2EE EAR file. For any given module project, there will be up to three J2EE staging projects generated with naming conventions that are based off of the module project's name:

- An Enterprise Application project
- An EJB™ project
  This project contains the generated EJBs that represent the runtime artifacts that make components into reality.
- A Dynamic Web project
  This project contains artifacts that represent Web components, for example, servlets and JSPs. A Dynamic Web project is generated when needed.

You will not be able to see these projects in the Business Integration view. To view these projects, you need to change perspectives, for example, to the Web perspective.

2.2.2 Implementing a mediation flow using WebSphere Message Broker

The primary capabilities of WebSphere Message Broker (WMB) are message routing, message transformation, message enrichment, and publish/subscribe.

WebSphere Message Broker is a powerful information broker that allows both business data and information, in the form of messages, to flow between disparate applications and across multiple hardware and software platforms. Business rules can be applied to the data that is flowing through the message broker in order to route, store, retrieve, and transform the information.

WebSphere Message Broker is comprised of two principle parts:

- WebSphere Message Broker Toolkit, which is a *development environment* for the creation of message flows, message sets, and other message flow application resources
- WebSphere Message Broker, which is the *runtime environment*, containing the components for running those message flow applications that are created in the development environment

Development environment

The development environment is where the message flow applications that provide the logic to the broker are developed. The broker uses the logic in the message flow applications to process messages from business applications at run time. In the Message Broker Toolkit you can develop both message flows and message sets for message flow applications.

Message flows

Message flows are programs that provide the logic that the broker uses to process messages from business applications. Message flows are created by connecting nodes together, with each node providing some basic logic. A selection of built-in nodes is provided with WebSphere Message Broker for performing particular tasks. The nodes in the message flows define the source and the target transports of the message, any transformations and manipulations based on the business data, and any interactions with other systems such as databases and files.

Message sets

A *message set* is a definition of the structure of the messages that are processed by the message flows in the broker. In order for a message flow to be able to manipulate or transform a message, the broker must know the structure of the message. The definition of a message can be used to verify message structure and to assist with the construction of message flows and mappings in the

Message Broker Toolkit. Message sets are compiled for deployment to a broker as a message dictionary, which provides a reference for the message flows to verify the structure of messages as they flow through the broker.

Broker Application Development perspective

The Broker Application Development perspective is the part of the Message Broker Toolkit that is used to design and develop message flows and message sets. It contains editors that create message flows, create transformation code such as ESQL, and define messages.

Runtime environment

The runtime environment is the set of components that are required to deploy and run message flow applications in the broker.

Broker

The *broker* is a set of application processes that host and run message flows. When a message arrives at the broker from a business application, the broker processes the message before passing it on to one or more other business applications. The broker routes, transforms, and manipulates messages according to the logic that is defined in their message flow applications. A broker uses WebSphere MQ as the transport mechanism both to communicate with the Configuration Manager, from which it receives configuration information, and to communicate with any other brokers to which it is associated. Each broker has a database in which it stores the information that it needs to process messages at run time.

Execution groups

Execution groups enable message flows within the broker to be grouped together. Each broker contains a default execution group. Additional execution groups can be created as long as they are given unique names within the broker. Each execution group is a separate operating system process; therefore, the contents of an execution group remain separate from the contents of other execution groups within the same broker. This can be useful for isolating pieces of information for security because the message flows execute in separate address spaces or as unique processes. Message flow applications are deployed to a specific execution group. To enhance performance, the same message flows and message sets can be running in different execution groups.

Configuration Manager

The *Configuration Manager* is the interface between the Message Broker Toolkit and the brokers in the broker domain. The Configuration Manager stores configuration details for the broker domain in an internal repository, providing a central store for resources in the broker domain. The Configuration Manager is responsible for deploying message flow applications to the brokers. The

Configuration Manager also reports back on the progress of the deployment and on the status of the broker. When the Message Broker Toolkit connects to the Configuration Manager, the status of the brokers in the domain is derived from the configuration information stored in the Configuration Manager's internal repository.

Broker domain

Brokers are grouped together in *broker domains*. The brokers in a single broker domain share a common configuration that is defined in the Configuration Manager. A broker domain contains one or more brokers and a single Configuration Manager. It can also contain a User Name Server. The components in a broker domain can exist on multiple machines and operating systems, and are connected together with WebSphere MQ channels. A broker belongs to only one broker domain.

User Name Server

A *User Name Server* is an optional component that is required only when publish/subscribe message flow applications are running, and where extra security is required for applications to be able to publish or subscribe to topics. The User Name Server provides authentication for topic-level security for users and groups that are performing publish/subscribe operations.

Broker Administration perspective

The *Broker Administration perspective* is the part of the Message Broker Toolkit that is used for the administration of any broker domains that are defined to the Message Broker Toolkit. This perspective is also used for the deployment of message flows and message sets to brokers in the defined broker domains. The Broker Administration perspective also contains tools for creating message broker archive (.bar) files. Bar files are used to deploy message flow application resources such as message flows and message sets.

2.2.3 Implementing Web services in CICS

As of CICS TS Version 3.1, support exists for developing and implementing Web Services. At the same time, a feature has been introduced that supports the development and implementation of composite service flows in CICS. This feature is called the *Service Flow Feature (SFF)*.

Development environment

CICS Web Services can be developed by using JCL utilities on z/OS or by using plugins that are a standard part of Rational Developer for System z.

CICS service flows can only be developed using the Service Flow Modeler tooling inside Rational Developer for System z.

CICS Web Services Assistant

The CICS *Web Services Assistant* is a set of batch utilities that can help you to transform existing CICS applications into Web services and to enable CICS applications to use Web services provided by external providers. It contains two utility programs:

- DFHLS2WS
  Generates a Web service binding file from a language structure. This utility also generates a Web service description.
- DFHWS2LS
  Generates a Web service binding file from a Web service description. This utility also generates a language structure that you can use in your application programs.

The assistant supports rapid deployment of CICS applications for use in service providers and service requesters, with a minimum of programming effort. When you use the Web Services Assistant for CICS, you do not have to write your own code for parsing inbound messages and for constructing outbound messages: CICS maps data between the body of a SOAP message and the application program's data structure.

CICS will, for the most part, generate and install the resource definitions automatically. You do have to define PIPELINE resources, but in many cases you can use one of the pipeline configuration files that CICS provides. These are:

- basicsoap11provider.xml
  This file defines the pipeline configuration for a service provider that uses the SOAP 1.1 message handler supplied by CICS.
- basicsoap11requester.xml
  This file defines the pipeline configuration for a service requester that uses the SOAP 1.1 message handler supplied by CICS.

The assistant can create a WSDL document from a simple language structure or a language structure from an existing WSDL document, and supports COBOL, C/C++, and PL/I. However, the assistant cannot deal with every possibility, and there are times when you will need to take a different approach. For example, this is the case when:

- You do not want to use SOAP messages.

  If you prefer to use a non-SOAP protocol for your messages, you can do so. However, your application programs will be responsible for parsing inbound messages and constructing outbound messages.

- You want to use SOAP messages, but you do not want CICS to parse them.

  For an inbound message, the assistant maps the SOAP body to an application data structure. In some applications, you might want to parse the SOAP body yourself.

- You have an application written in an unsupported language.

  In this case you should either write a wrapper program in a supported language, or write a program to perform the mapping.

- The CICS Web Services Assistant does not support your application's data structure.

  Although the CICS Web Services Assistant supports the most common data types and structures, some exist that are not supported. That is, there may not always be a 1 to 1 mapping between XML data types and the data types in your language structure. In this situation, you should first consider providing a wrapper program that maps your application's data to a format that the assistant can support. If this is not possible, consider using Rational Developer for System z. As a last resort you might need to change your application's data structure.

If you decide not to use the CICS Web Services Assistant, you will have to:

- Provide your own code for parsing inbound messages and constructing outbound messages (unless you use Rational Developer).
- Provide your own pipeline configuration file.
- Define and install your own URIMAP and PIPELINE resources.

Rational Developer for z

IBM Rational Developer for zSeries V7.1 combines the power of service-oriented architectures that use Java 2 Enterprise Edition (J2EE), CICS, IMS, COBOL, and PL/I technologies with a Rapid Application Development (RAD) paradigm and teaming, and it brings these to diverse enterprise application development organizations. Today's applications are not developed by super technologists in a vacuum, but by teams of people with varying levels of technology backgrounds and the following in common:

- A firm understanding of the business
- A requirement to integrate across many business processes
- A desire to leverage currently executing (as well as new) Web-oriented technologies
- A desire to promote use of the Internet in meeting business needs
- A desire to leverage development standards and expertise to move the business forward

Two overriding goals drive these organizations:

- Meeting requirements of time to market
- Meeting high standards for quality

To help enterprise customers meet these goals, IBM delivered Rational Developer for System z. Rational Developer for System z supports a development process that includes:

- A visual Web-flow construction environment built on the Struts-based Model-View-Controller (MVC) paradigm. The elements of the MVC paradigm are as follows:
  - Model and Business Logic:
    Invoked by or implemented in Struts action classes. Business logic can be implemented in a variety of technologies, such as Java, COBOL, and PL/I.
  - View:
    Implemented using HTML and JavaServer™ Pages with special Struts tags.
  - Controller:
    Flow of Web applications, implemented as Struts actions or actions defined supporting various underlying business and technical processes.

  The MVC paradigm is recommended by J2EE experts and used by architects and developers to define, reuse, and debug application organization, flow, and actions.
- Editors and interactive development environments (IDEs) for Java, COBOL, and PL/I components including language understanding, syntax checking, and unit testing in WebSphere, CICS, and IMS transactional environments.
- Web services support including client generation, XML editors, COBOL and PL/I adapter generation, and support for Web services deployed to WebSphere, CICS, and IMS.

CICS Service Flow Feature

The CICS Service Flow Feature provides the ability to invoke a group of CICS programs as a *composite service*. There will be cases where the scope of a service entails multiple CICS programs, typically executed sequentially.

The CICS Service Flow Feature is a business service integration adapter for all CICS applications. It offers both tooling and runtime components. These components enable the creation of CICS business services for integration into an SOA and into business process collaborations.

The CICS Service Flow Feature provides the capability to implement CICS business services by composing a sequence of CICS application interactions using:

- A graphical modeling integrated development environment that enables the creation of CICS business services by composing a flow of CICS application interactions.
- A generation capability that transforms the composed flow of CICS application interactions to form a runtime application, highly optimized for the CICS environment, that retains the inherent qualities of service provided by the existing CICS application implementation.
- A run-time component, known as the CICS Integrator Adaptor, that extends the CICS Transaction Server for z/OS (CICS TS) environment. It offers adapters that exploit CICS interfaces to invoke the CICS terminal-oriented transactions and COMMAREA programs as required by the service flow.

CICS business services built with and hosted in CICS Service Flow Feature expose business function interfaces that can readily be published as Web services in an SOA by exploiting the Web services capabilities of CICS TS V3.1.

The CICS application assets orchestrated by the service flow do not have to be altered to support the CICS business service flow. Additionally, these business services can be integrated as process steps in a business process orchestrated by a business process engine such as WebSphere Process Server.

2.2.4 Implementing a front-end in WebSphere Portal

The user interaction layer for a business process or a service can be implemented in many ways. Most development tools provide wizards to generate a simple J2EE user interface component, such as a JavaServer Page (JSP™). This user interface can then be further customized using these same development tools. One way of implementing a user interface is to use WebSphere Portal. A portlet can be created or generated for any business process or service, based on the WSDL of the process or service. Once you have a portlet for your business process or service, you can incorporate that portlet in your enterprise portal.

Development environment

The development environment for portlets is IBM Rational Application Developer or could be any open source J2EE development environment supporting the portlet specification.

3

Customer inquiry service

This chapter describes the Customer inquiry service and how we implemented the Web Services interface to the CICS program.

This chapter covers the folowing topics:

- Overview of the Customer inquiry module
- DB2 and CICS resources for our scenario
- The process of creating a CICS Web service

Important: If you construct the scenario in your own environment, be aware that you may need to choose you own naming conventions, as well as use your own variables for TCP/IP addresses, port number, host names, and so on. Please pay special attention to keeping these consistent. The names and variables we use in this publication are only meant as examples.

3.1 Overview: Customer inquiry service

The Customer inquiry module is used to obtain information about a customer from a DB2 table based on the customer number provided. A CICS COBOL application, SOAPINQC, is used to extract a custoimer record from a DB2 table.

The original Customer inquiry application was built based on a traditional CICS BMS map. For our version of the program we are not interested in the BMS map because we built a new process with a new user interface residing in WebSphere Portal.

The input to this function is a three character string representing the customer number for the query. Once the user enters a three digit customer number, the detailed customer information is returned. If the customer number is not in the database an error message is returned indicating the customer number was not found[1].

The request and reply data are passed between the CICS application and the caller via the DFHCOMMAREA. In a receiving COBOL program, you must give the data area the name DFHCOMMAREA. This area is specific to CICS application programming.

3.1.1 Inputs and outputs

Figure 3-1 shows the input data for this example as seen in a WebSphere Integration Developer module test window. The value `001` was used to test the customer inquiry.

Name	Type	Value
⊟ DFHCOMMAREA	DFHCOMMAREA	
comm_CustNo	string	001

Figure 3-1 DFHCOMMAREA input data as seen in WID

Figure 3-2 on page 35 shows the output data that is returned to the client as seen in a WID test module window.

[1] Our Order process will later on use a Human Task to report this situation.

Name	Type	Value
DFHCOMMAREA	DFHCOMMAREA	
comm_CustNo	string	001
comm_LstName	string	Chung
comm_FstName	string	SeungKeon
comm_EMail	string	skchung@kr.ibm.com
comm_Department	string	ZSOA
comm_Address	string	MMAA Dogok GangNam
comm_City	string	Seoul
comm_State	string	n/a
comm_Country	string	Repub of Korea
comm_RetCode	string	000

Figure 3-2 DFHCOMMAREA output data as seen in WID

3.2 Environment

Our technical environment for this service consists of CICS programs and a DB2 table.

3.2.1 CICS application

The following modules play a role in the application:

- Transaction SOAC is used to invoke the SOAPCALC program.
- Program SOAPCALC is used to call the SOAPINQC program using a BMS interface. It returns one of the following codes:
  - 000 - NORMAL COMPLETION
  - 100 - NO CUSTOMER INFORMATION
  - ERR - SQL ERROR
  - NUM - CUSTOMER NUMBER IS INVALID
- BMS map SOAICUS is the map for the SOAPCALC program.
- SOAPINQC is the business logic program and the one we will be Web service-enabling. It performs the following functions:
  - Receives the customer number for the query.
  - Accesses DB2 to obtain the customer information.
  - Moves the requested query data to COMMAREA.

For the Web service enablement we focus on the SOAPINQC program, executing the actual business logic.

The source code for these components is included in the additional material.

3.2.2 DB2 configuration

A DB2 table is used to hold the customer information, CUSTOMER, to reflect typical data that would be used to track customers who order parts from a warehouse. The table has the following layout:

```
CUST_NO       CHAR(03) NOT NULL,
  CUST_LN       CHAR(25),
  CUST_FN       CHAR(15),
  CUST_EMAIL    CHAR(40),
  CUST_DEPT     CHAR(4),
  CUST_ADDR     CHAR(20),
  CUST_CITY     CHAR(20),
  CUST_ST       CHAR(5),
  CUST_CTRY     CHAR(15),
PRIMARY KEY (CUST_NO)
```

A sample job to define a table space and the table is included in the additional material. The name is @DB2CUST.

3.3 The process of creating a Web service

The Customer inquiry module was orignally not a Web service. In the following sections we describe how a Web service was created out of the existing Customer inquiry program.

3.3.1 CICS Web services overview

CICS supports two different approaches to deploying your CICS applications in a Web Services environment. One approach enables rapid deployment, and requires the least amount of programming effort. The other approach gives you complete flexibility and control over your Web Services applications, using code that you write to suit your particular needs. We chose to use the rapid deployment method to create our Web service. For more information on CICS Web Sevices see the CICS Trasaction Server 3.1 Information Center.

A Web service description contains abstract representations of the input and output messages used by the service. CICS uses the Web service description to

construct the data structures used by application programs. At run time, CICS performs a mapping between the application data structures and the messages.

The description of a Web service contains, among other things:

- The operations.
- For each operation, an input message and an optional output message.
- For each message, the message structure, defined in terms of XML data types. Complex data types used in the messages are defined in an XML schema which is contained in the <types> element within the Web service description. Simple messages can be described without using the <types> element.

Web Service Definition Language (WSDL) contains an abstract definition of an operation, and the associated messages; it cannot be used directly in an application program. To implement the operation, a service provider must do the following:

- Parse the WSDL, in order to understand the structure of the messages.
- Parse each input message and construct the output message.
- Perform the mappings between the contents of the input and output messages, and the data structures used in the application program.

A service requester must do the same in order to invoke the operation.

When you use the the CICS Web Services Assistant (WSA) or Rational Developer for System z (RDz), much of this is done for you, and you can write your application program without detailed understanding of WSDL, or of the way the input and output messages are constructed.

Asset discovery

According to our business process for the Catalog Order system, the first step is to check whether a customer is registered in our customer database.

Through the asset discovery process, we found that we are able to reuse the existing Customer Information Inquiry module described previously by enabling CICS Web Services on this program.

Note: For the purpose of the asset discovery process, you can consider using WebSphere Studio Asset Analyzer (WSAA).

3.3.2 Creating the CICS Web service

This application has been separated into two parts:

- BMS-based presentation logic (SOAPCALC)
- Business logic (SOAPINQC)

The two programs communicate using the CICS COMMAREA. This type of separation is a very good approach for easily applying CICS Web Services.

The Enterprise Service Tools (EST) functionality in Rational Developer for System z (RDz) provides the capability to transform COMMAREA-based applications into Web Services.

The EST has several tools to modernize legacy applications. One of the tools is XML Services for the Enterprise (XSE) that lets you easily adapt existing COBOL-based applications to both consume and produce XML messages. As part of XSE, we will use Web Services for CICS.

XSE provides of the following wizards:

- The Create New Service Interface (bottom-up) wizard, which generates a new Web Services interface for an existing COBOL program. Typically, this is called a “bottom-up” approach since the existing COBOL application is at the bottom of the new Web service creation process.
- The Create New Service Implementation (top-down) wizard, which generates a new Web service implementation template for a CICS Web Services runtime. Typically, this is called a “top-down” approach since the COBOL application is created top-down starting from the existing Web service definition (typically, a WSDL file).
- The XML-to-COBOL mapping wizard and tools that help map existing Web service interfaces or an XML data definition to an existing COBOL program.

Since we need to generate the Web Services from existing CICS COBOL program, we chose the bottom-up approach.

CICS Web Services utilities used

CICS provides two utilities as Java classes to assist in executing Web Services tasks (`com.ibm.cics.wsdl.ls2ws.ls2ws` and `com.ibm.cics.wsdl.ws2ls.ws2ls`) and JCL to perform this. The two utilities provided are:

- DFHLS2WS
  Generates a Web service description and a Web service binding file from a high level language data structure (bottom-up)

- DFHWS2LS
  Generates a high level language data structure and a Web service binding file from a Web service description (top-down)

There are two cataloged procedures, DFHLS2WS and DFHWS2LS in SDFHINST to invoke these utilities. These procedures set up the environment and run shell scripts. The related shell scripts and jar files that contain the utilites are located under <CICS_Home_directory>/lib/wsdl/

Through this process, the following artifacts are generated:

- Conversion program: SOAWINQD.cbl
  The conversion program includes three components:
  - An input converter that converts an inbound XML message into a COBOL language structure
  - An output converter that converts a COBOL structure to an outbound XML message
  - A Driver program that controls I/O converter programs
- WSBind file: SOAPINQC.wsbind
  This file tells CICS Web Services how conversion is to be performed and what the URI map looks like.
- WSDL file: SOAWINQ.wsdl
  This WSDL describes this CICS-based Web service. The consumer (or requester) of this Web service is able to invoke the service through this file. This file does not need to be deployed to the CICS runtime but rather, needs to be registered in UDDI or imported into WID or another tool to integrate this Web service.
- SOAWINQI.xsd and SOAWINQO.xsd
  These files are XML Schema Definition files that describe the input and output XML documents respectively and are used in RDz internally.

Steps to execute

To generate a Web service from the CICS Customer inquiry module we executed the following steps using Rational Developer for System z (RDz):

1. Start RDz and switch to the EST Perspective as shown in Figure 3-3. You may have to close the Welcome Window first.

   Click **Window** → **Open Perspective** → **Other** and select **Enterprise Service Tools**.

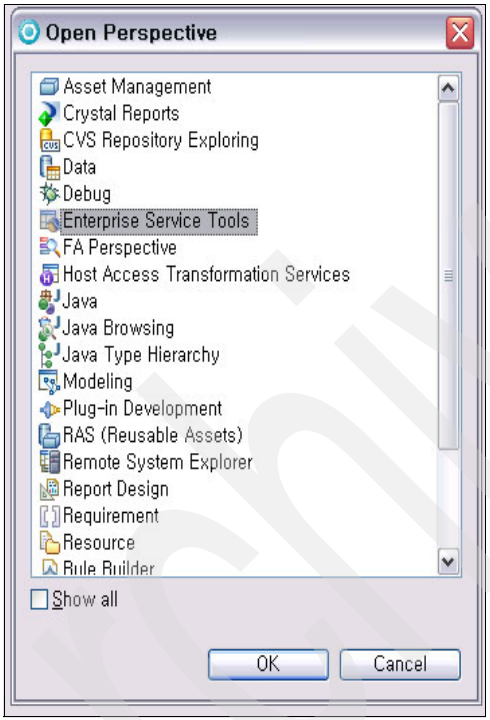

Figure 3-3 Opening the EST Perspective

2. Create a project for the CICS Web service as shown in Figure 3-4 by right-clicking in the EST Project Explorer pane and then selecting **New** → **Web Services for CICS Project**.

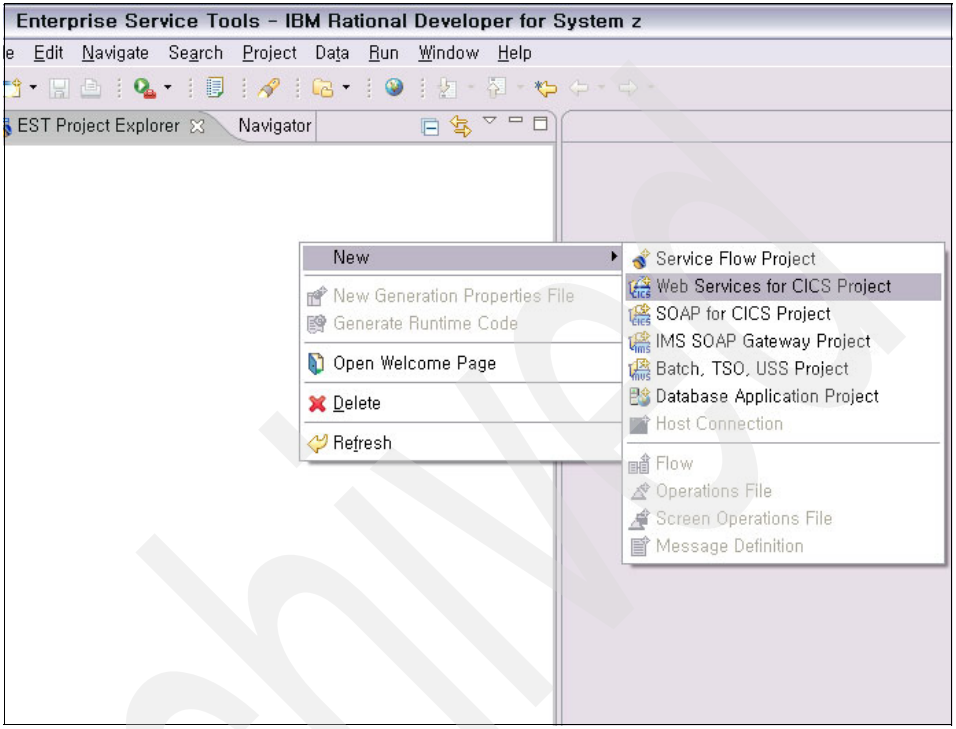

Figure 3-4 Selecting Web Services for CICS Project

3. Import the source files into the zSOA_CICS_WS project as shown in Figure 3-5 by right-clicking the project name and then clicking **Import → Source Files**.

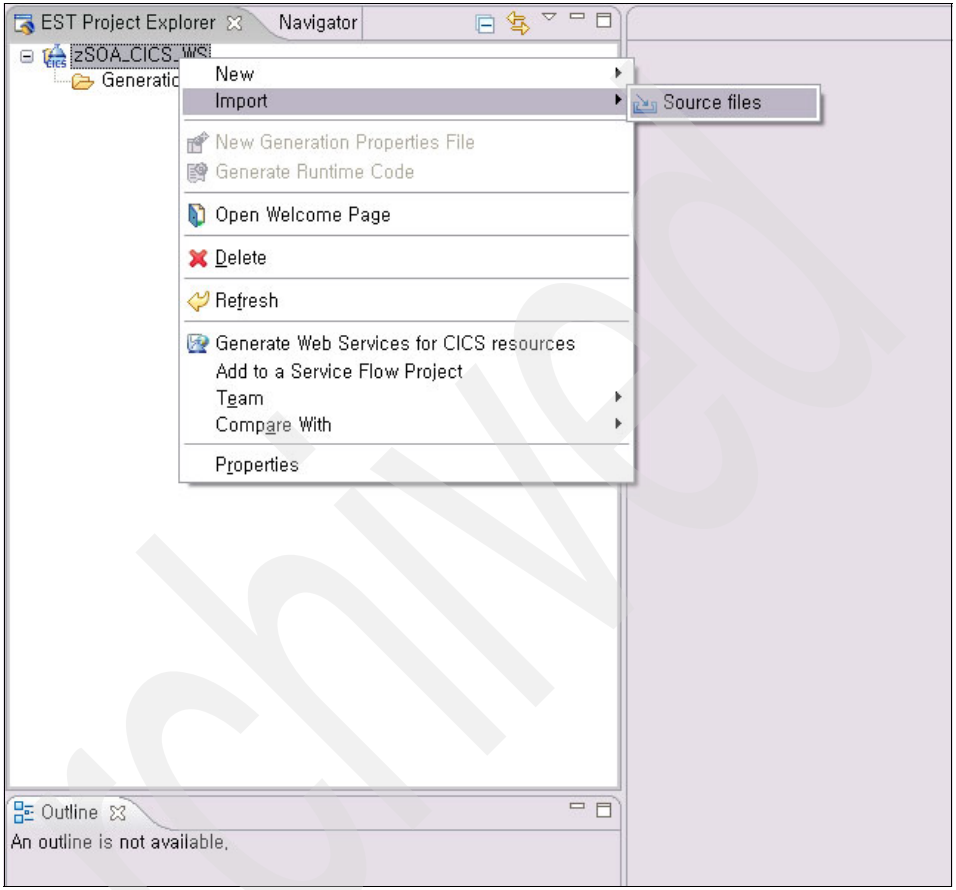

Figure 3-5 Importing the source files

4. Select the SOAPINQC COBOL program including the related Copybook in the project as shown in Figure 3-6.

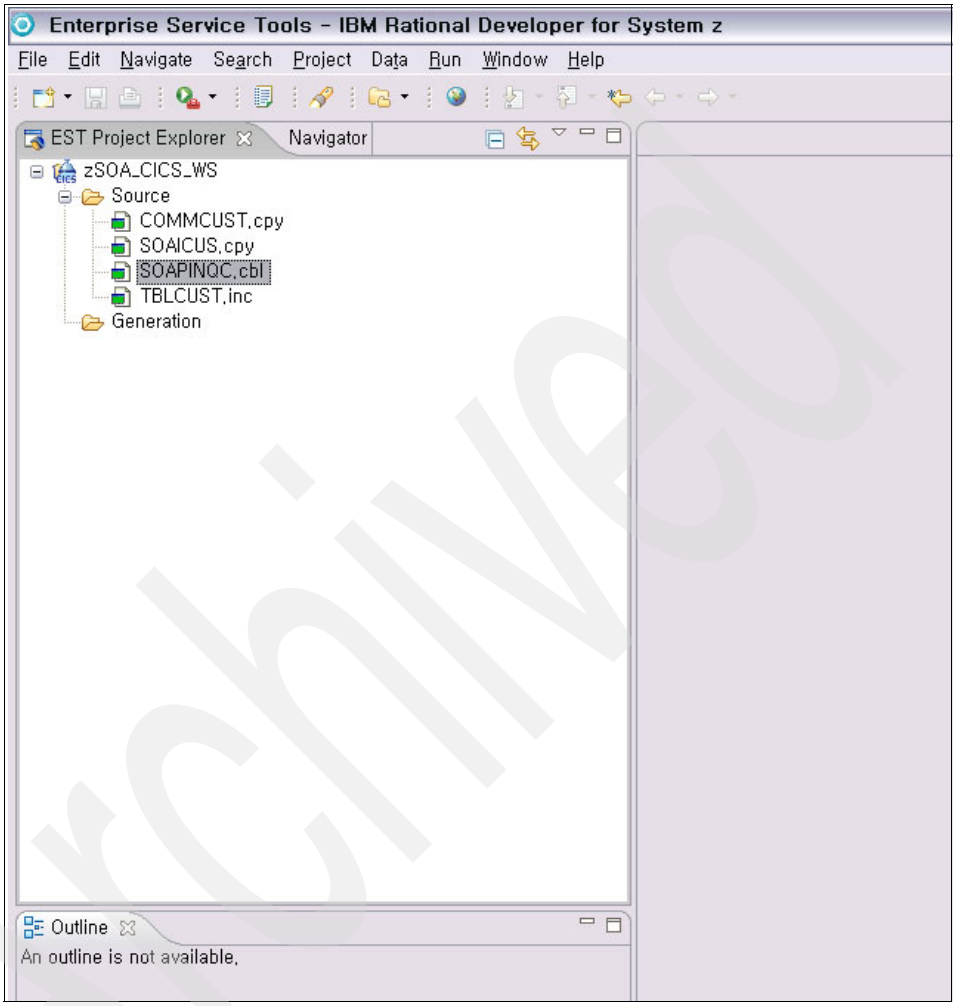

Figure 3-6 Selecting SOAPINQC COBOL source

5. Generate the artifacts to be installed into the CICS runtime as shown in Figure 3-7.

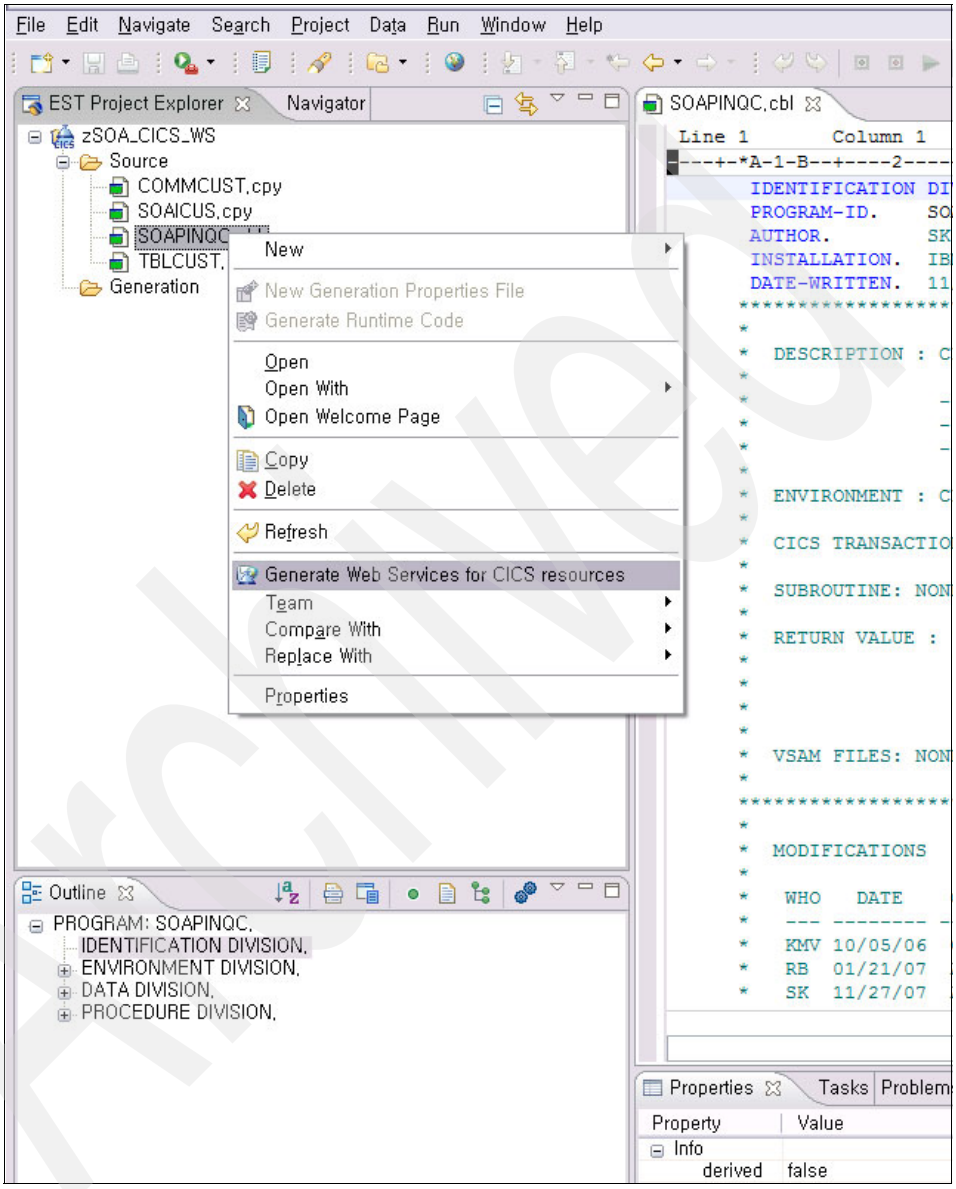

Figure 3-7 Generating the Web Services artifacts

6. Specify the options for the EST Wizard as shown in Figure 3-8.

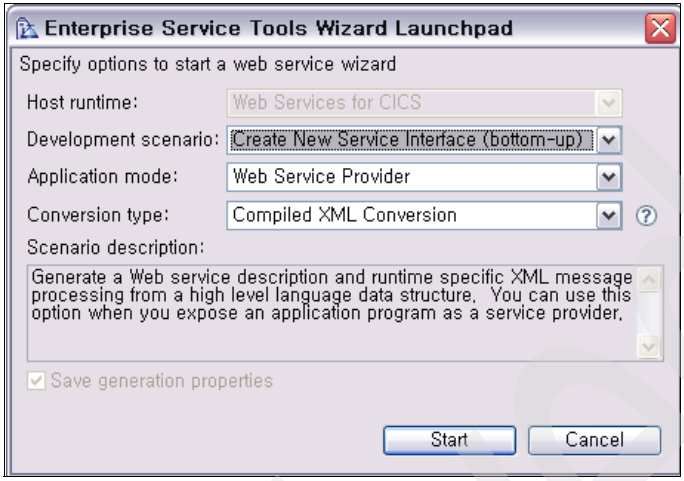

Figure 3-8 Specifying the CICS Web Services wizard options

7. Define the service interface as shown in Figure 3-9.

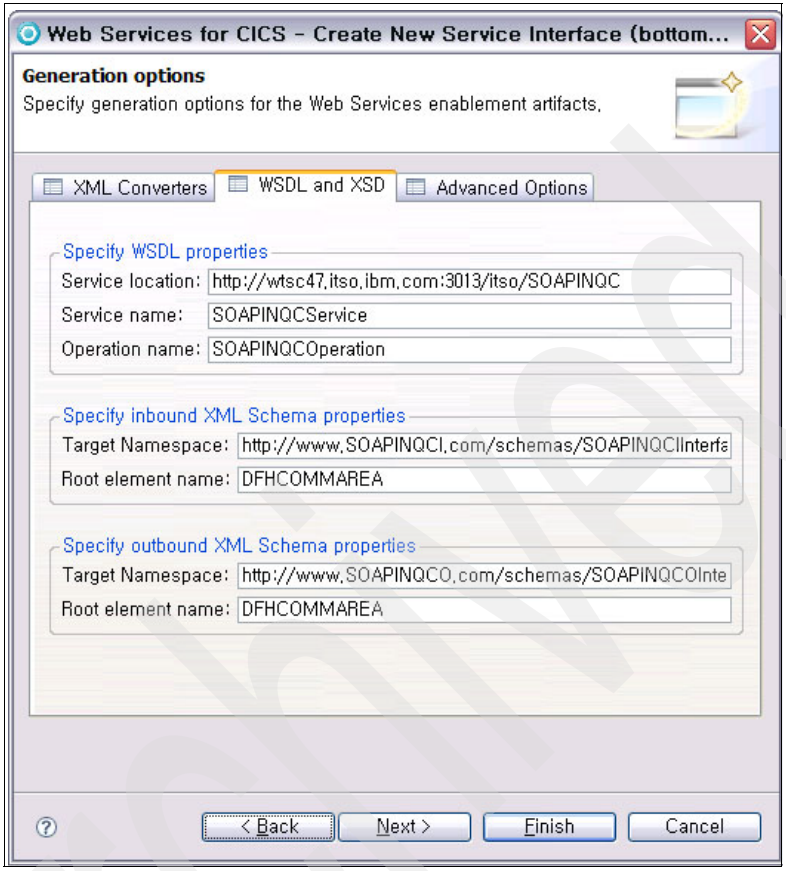

Figure 3-9 Creating CICS Web Services generation options

8. Once the wizards with all options have been completed, the Web service artifacts are generated. At this time certain validations are performed as well. If the generation is successful, you will find the new artifacts created in the workbench. The most interesting one is the WSDL file generated. See Figure 3-10.

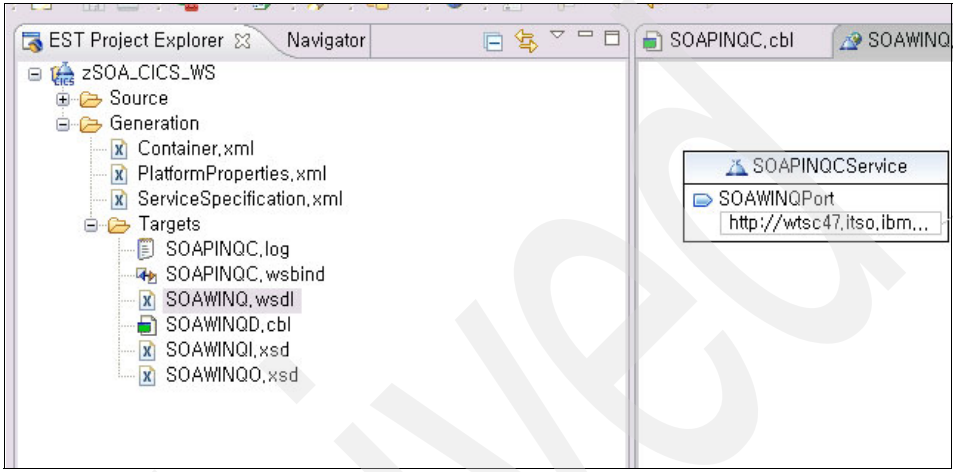

Figure 3-10 WSDL file generated

9. If this WSDL file is opened (by double-clicking it), a graphical view of the contents of the WSDL file is opened. This view can be edited. See Figure 3-11.

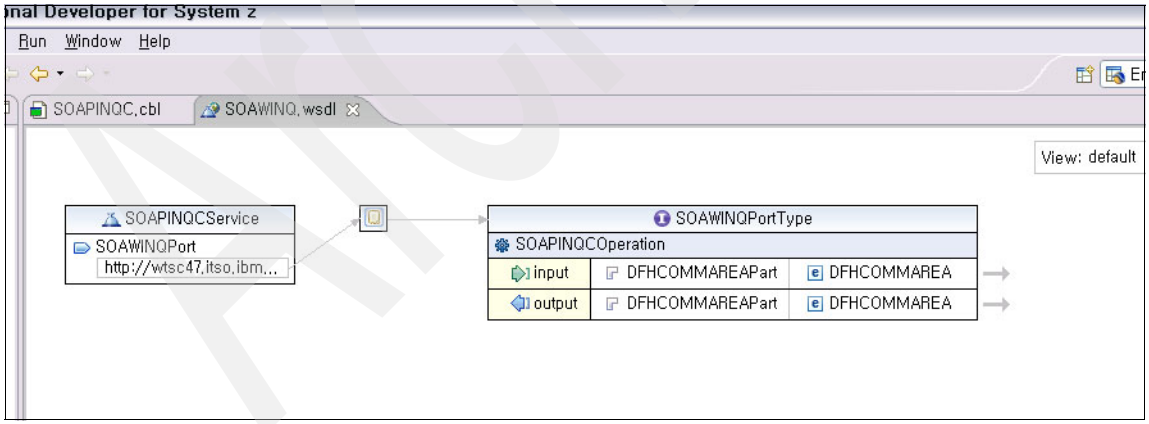

Figure 3-11 Graphical view of the WSDL file

10. Deploy and configure the CICS runtime for Web Services, as follows:

    a. Compile and link-edit the conversion program (SOAWINQD):

       - Upload the conversion program SOAWINQD.cbl to a PDS member named as SOAWINQD on a z/OS system.
       - Compile and link-edit this module into the library that is concatenated to DFHRPL in the CICS region startup JCL.

    b. Upload the WSBind file into the CICS Web Services pick up directory using binary mode. In our case this directory is:

       /u/skchung/itso/wspickup/provider/

    c. Work with RDO to define and activate the resources in CICS. Set the following definitions (using values of your choice):

       - CICS PIPELINE definition, as shown in Example 3-1.

Example 3-1 CICS PIPELINE definition

```
OVERTYPE TO MODIFY                          CICS RELEASE = 0640
  CEDA  ALter PIpeline( PIPESOA  )
   PIpeline       : PIPESOA
   Group          : ITSOTEST
   Description  ==> BASIC SOAP 1.1 PROVIDER PIPELINE FOR ITSO
   STatus       ==> Enabled          Enabled | Disabled
   Configfile   ==> /u/skchung/itso/cicsProvider.xml
   (Mixed Case) ==>
                ==>
                ==>
                ==>
   SHelf        ==> /u/skchung/itso/shelf/
   (Mixed Case) ==>
                ==>
                ==>
                ==>
   Wsdir          : /u/skchung/itso/wspickup/provider/
   (Mixed Case)   :
+                 :

                                      SYSID=WRKS APPLID=CITSO12

PF 1 HELP 2 COM 3 END 6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

When we install this PIPELINE, CICS scans the pickup directory and finds all of the *.wsbind files within that directory. Based on the contents of these files, CICS creates WEBSERVICE and URIMAP

resources automatically. Also, CICS copies *.wsbind file into the shelf directory.

The CICS log for the CEDA INSTALL command is shown in Example 3-2.

Example 3-2 CEDA install log

```
DFHPI0912 I 11/28/2007 14:00:24 CITS012 SKCHUNG WEBSERVICE SOAPINQC was
successfully discarded.
DFHPI0710 I 11/28/2007 14:00:24 CITS012 SKCHUNG PIPELINE PIPESOA was
successfully discarded.
DFHPI0701 I 11/28/2007 14:00:24 CITS012 SKCHUNG PIPELINE PIPESOA has
been created.
DFHRD0124 I 11/28/2007 14:00:24 CITS012 SC47TC08 SKCHUNG CEDA INSTALL
PIPELINE(PIPESOA)
TC08     CEDA SKCHUNG  11/28/07 14:00:24 INSTALL PIPELINE(PIPESOA)
GROUP(ITSOTEST)
DFHPI0703 I 11/28/2007 14:00:24 CITS012 ITS012 PIPELINE PIPESOA is
about to scan the WSDIR directory.
DFHPI0901 I 11/28/2007 14:00:24 CITS012 ITS012 New WEBSERVICE SOAPINQC
is being created during a scan against PIPELINE PIPESOA.
DFHPI0910 I 11/28/2007 14:00:24 CITS012 ITS012 WEBSERVICE SOAPINQC
within PIPELINE PIPESOA has been created.
DFHPI0915 I 11/28/2007 14:00:24 CITS012 ITS012 WEBSERVICE SOAPINQC is
now INSERVICE and is ready for use.
DFHPI0903 I 11/28/2007 14:00:24 CITS012 ITS012 New URIMAP $649320 is
being created during a scan against PIPELINE PIPESOA for
     WEBSERVICE SOAPINQC.
DFHPI0704 I 11/28/2007 14:00:24 CITS012 ITS012 PIPELINE PIPESOA
Implicit scan has completed.
Number of wsbind files found in the WSDIR directory: 000001.
Number of successful WEBSERVICE creates: 000001.
Number of failed WEBSERVICE creates: 000000.
```

Issue the command:

```
/u/skchung/itso/shelf/CITS012/PIPELINE/PIPESOA>ls -al
```

After this command is executed, the CICS shelf directory appears as shown in Example 3-3.

Example 3-3 CICS shefl directory after PIPELINE INSTALL

```
total 14
drwxr-xr-x   3 ITS012   SYS1         352 Nov 28 14:00 .
drwxr-xr-x   3 ITS012   SYS1         288 Nov 28 14:00 ..
-rwxr-xr-x   1 ITS012   SYS1        2984 Nov 28 14:00 SOAPINQC.wsbind
```

```
drwxr-xr-x   2 ITSO12   SYS1         256 Nov 28 14:00 c
-rwxr-xr-x   1 ITSO12   SYS1         427 Nov 28 14:00 cicsProvider.xml
```

When a new or updated *.wsbind file is created the following command will activate the modified version:

```
CEMT P PIPELINE(PIPESOA) SCAN
```

This command causes CICS to perform the same work as the INSTALL command described previously. Example 3-4 on page 50 shows the CICS log for the CEMT PERFORM command.

Example 3-4 CEMT P command output log

```
DFHPI0703 I 11/28/2007 15:38:25 CITSO12 SKCHUNG PIPELINE PIPESOA is
about to scan the WSDIR directory.
DFHPI0902 I 11/28/2007 15:38:25 CITSO12 SKCHUNG WEBSERVICE SOAPINQC is
being updated during a scan against PIPELINE PIPESOA.
DFHPI0912 I 11/28/2007 15:38:25 CITSO12 SKCHUNG WEBSERVICE SOAPINQC was
successfully discarded.
DFHPI0901 I 11/28/2007 15:38:25 CITSO12 SKCHUNG New WEBSERVICE SOAPINQC
is being created during a scan against PIPELINE PIPESOA.
DFHPI0910 I 11/28/2007 15:38:25 CITSO12 SKCHUNG WEBSERVICE SOAPINQC
within PIPELINE PIPESOA has been created.
DFHPI0915 I 11/28/2007 15:38:25 CITSO12 SKCHUNG WEBSERVICE SOAPINQC is
now INSERVICE and is ready for use.
DFHPI0903 I 11/28/2007 15:38:25 CITSO12 SKCHUNG New URIMAP $037440 is
being created during a scan against PIPELINE PIPESOA for
     WEBSERVICE SOAPINQC.
DFHPI0715 I 11/28/2007 15:38:25 CITSO12 SKCHUNG PIPELINE PIPESOA
explicit scan has completed.
Number of wsbind files found in the WSDIR directory: 000001.
Number of WEBSERVICEs created or updated: 000001.
Number of WEBSERVICEs not requiring an update: 000000.
Number of failed WEBSERVICE creates or updates: 000000.
```

A good reference is the CICS Web Service Runtime Configuration manual: *CICS Web Services Guide*, SC34-6458-05.

- Example 3-5 shows our TCPIPSERVICE.

Example 3-5

```
OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
  CEDA  ALter TCpipservice( SOAPORT  )
   TCpipservice   : SOAPORT
   GROup          : ITSOTEST
   DEscription  ==> TCPIPSERVICE TO RECEIVE THE INBOUND HTTP TRAFFIC SOA APPS
```

```
   Urm          ==> DFHWBAAX
   POrtnumber   ==> 03013              1-65535
   STatus       ==> Open               Open | Closed
   PROtocol     ==> Http               Iiop | Http | Eci | User
   TRansaction  ==> CWXN
   Backlog      ==> 00020              0-32767
   TSqprefix    ==>
   Ipaddress    ==>
   SOcketclose  ==> No                 No | 0-240000 (HHMMSS)
   Maxdatalen   ==> 000032             3-524288
  SECURITY
   SSl          ==> No                 Yes | No | Clientauth
   CErtificate  ==>
+  (Mixed Case)
                                                   SYSID=WRKS APPLID=CITS012

PF 1 HELP 2 COM 3 END            6 CRSR 7 SBH 8 SFH 9 MSG 10 SB 11 SF 12 CNCL
```

11. Test the CICS Web service using the Web Servcies Explorer in RDz, as shown in Figure 3-12.

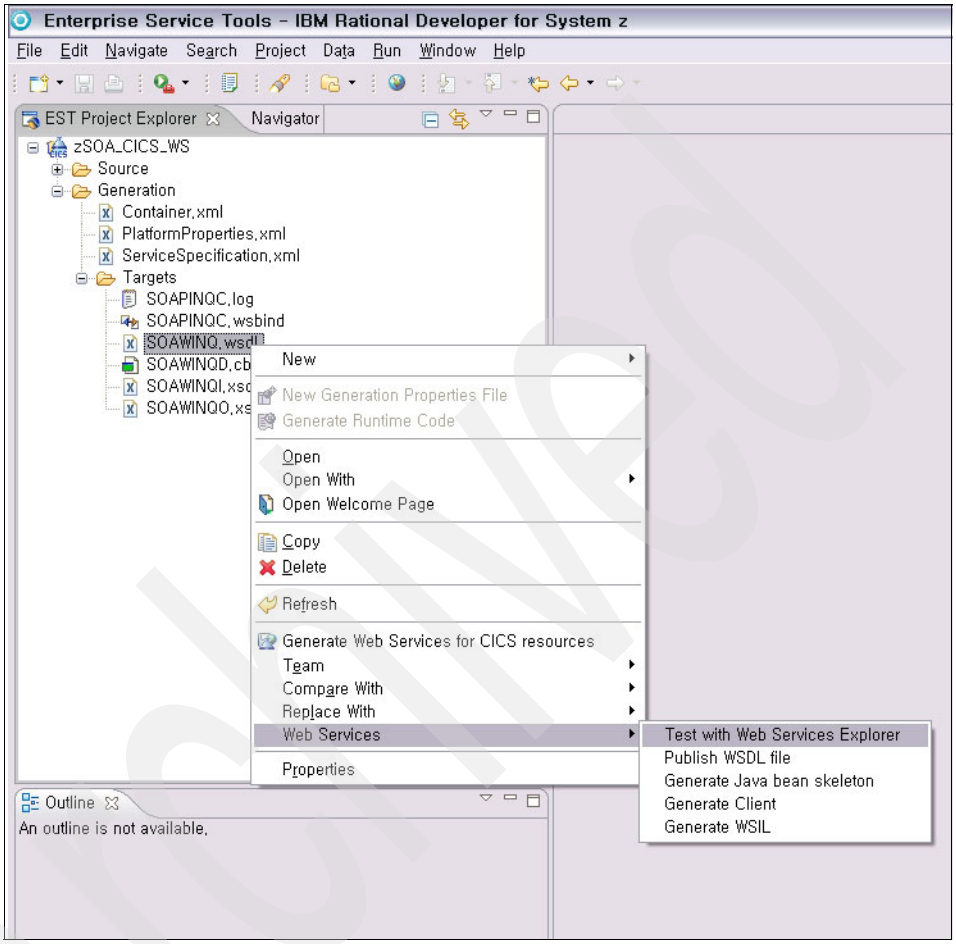

Figure 3-12 Selecting the Web Services Explorer in RDz

12. In the Web Services Explorer, fill in a value for Customer number as shown in Figure 3-13.

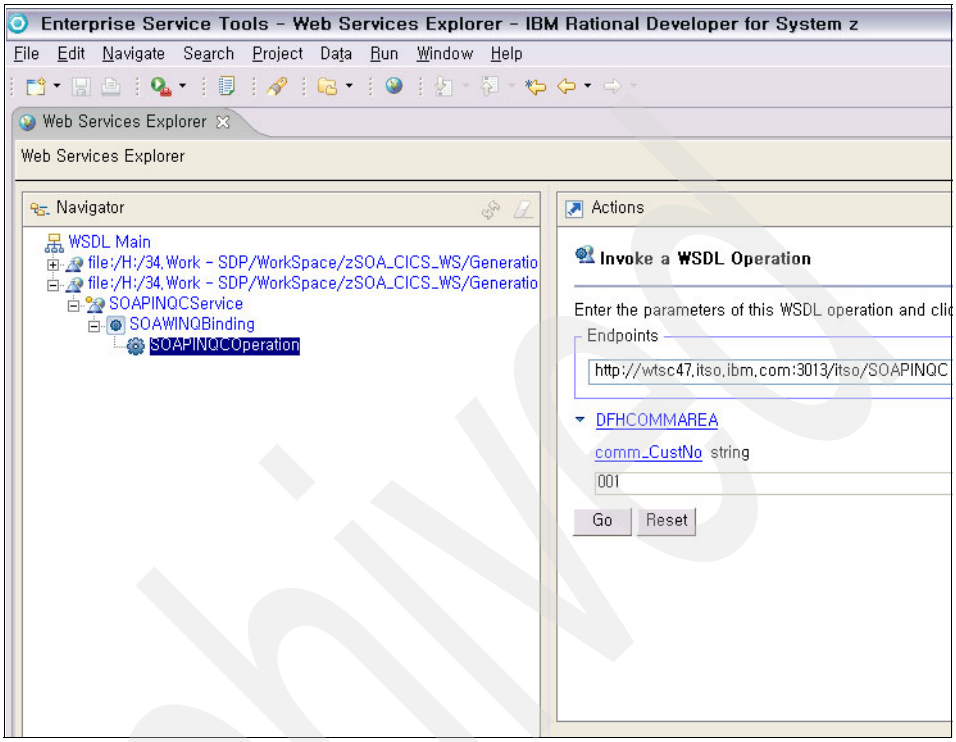

Figure 3-13 Testing the Web Service

3.4 Additional material

Refer to “Additional material” on page 229 for instructions on downloading the sample material.

4

Credit check service

This chapter describes the Credit check function and how we implemented the interface to DB2 using WebSphere Message Broker (WMB). Our objective is to illustrate how an application can call a Web service that executes a DB2 stored procedure through an Enterprise Service Bus (ESB).

WebSphere Message Broker enables information packaged as messages to flow between different business applications. It can perform transformation on messages so data sent in one format can be received in another format, thus eliminating the need for application changes when data formats change. More detailed information can be found in the WMB v6 Information Center at:

`http://publib.boulder.ibm.com/infocenter/wmbhelp/v6r0m0/index.jsp.`

There are several components in WebSphere Message Broker as shown in Figure 4-1. We mentioned the most important concepts in WMB in “Implementing a mediation flow using WebSphere Message Broker” on page 26; we do not go into more detail in this book.

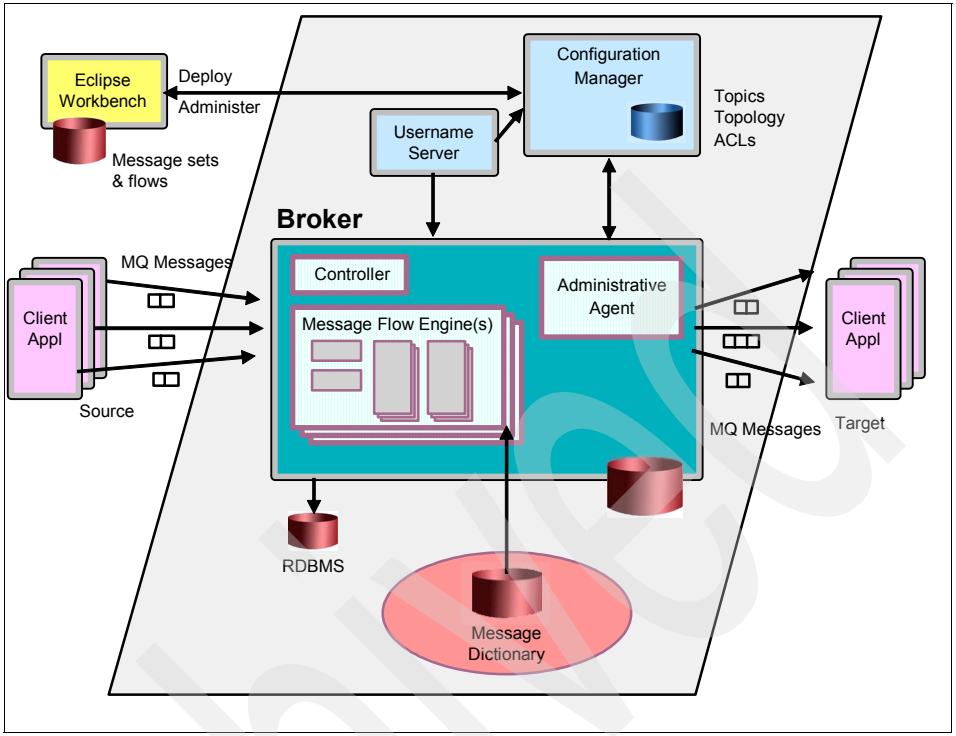

Figure 4-1 Structure of WebSphere Message Broker

The Message Broker Toolkit shown in Figure 4-1 and labeled “Eclipse Workbench” is an Eclipse-based tool that is used to define message flows, create execution groups, and deploy flows to WMB. We used the toolkit to create the message flow to invoke a DB2 stored procedure as a Web service through using WMB as the ESB.

In the following sections we describe the tasks required to create the mediation flow.

The topics covered in this chapter are:

- Overview of the Credit check service
- Environment
- Creating a WMB message flow

Important: If you construct the scenario in your own environment, be aware that you may need to choose you own naming conventions, as well as use your own variables for TCP/IP addresses, port number, host names, and so on. Please pay special attention to keeping these consistent. The names and variables we use in this publication are only meant as examples.

4.1 Overview of the Credit check service

The Credit check service is used to verify that a customer has good credit in order to approve the purchase. The data for each customer is stored in a DB2 table on z/OS. “DB2 configuration” on page 58 describes the structure of the DB2 table used for this scenario. A DB2 stored procedure is used to extract the data and return information about the customer. This DB2 stored procedure is invoked through a message flow running in WebSphere Message Broker. “Message flow” on page 59 describes what the message flow looks like.

4.1.1 Inputs and outputs

When we imported the WSDL for our Web service into WID the following input and output business objects were created automatically.

The input to this process component is a character string representing:

- Customer Number

The output data includes:

- Customer Number
- Last Name
- First Name
- Credit Expiration Date
- Credit Result

4.2 Environment

In this service we used the following components:

- DB2, running the stored procedure

- WebSphere Message Broker
  To run the mediation flow accessing the DB2 stored procedure
- WebSphere Message Broker Toolkit
  For developing the mediation flow

4.2.1 DB2 configuration

Our DB2 stored procedure is named CREDINQJ. The DDL used to create the procedure is shown in Example 4-1.

Example 4-1 DDL to create DB2 stored procedure

```
CREATE PROCEDURE CREDINQJ (INOUT CUSTNO CHAR(3),
      OUT CUSTLN CHAR(25),
      OUT CUSTFN CHAR(15),
      OUT CREDEXPIREDATE DATE,
      OUT CREDRESULT CHAR(3))
   EXTERNAL NAME 'com.ibm.itso.zsoa.db2sp.CredInqJ.credInqJ'
   LANGUAGE JAVA
   READS SQL DATA
   PARAMETER STYLE JAVA
   FENCED
   WLM ENVIRONMENT SPCREDQJ;

COMMENT ON PROCEDURE CREDINQJ IS
'Inquiry Customer Credit Status implemented by Java';
```

4.3 Creating a WMB message flow

Our mission now is to create a message flow that can be invoked from the business process as a Web service. This message flow must include a call to the DB2 stored procedure.

Our flow consists of an HTTP input node, several compute nodes, and an HTTP reply node. We used the HTTPInput node to receive our application Web services requests for processing by a message flow. By using the HTTPInput node with the HTTPReply node, the broker acts as an intermediary for Web Services, and Web Services requests can be transformed and routed in the same way as other message formats that are supported by WebSphere Message Broker. When including an HTTPInput node in a message flow, you must either include an HTTPReply node in the same flow, or pass the message

to another flow that includes an HTTPReply node. We included the HTTPReply node in the same flow.

HTTP messages are non-transactional. However, if the message flow interacts with a database or another external resource, such as a WebSphere MQ queue, these interactions are performed transactionally. The HTTPInput node provides commit or rollback, depending on how the message flow has ended. In our example the message flow interacts with DB2 by calling a stored procedure.

4.3.1 Message flow

Our Message Broker flow for this scenario, as seen in the WMB Toolkit, is shown in Figure 4-2. The HTTP input node receives data for the request from the client. Control passes from this node to three compute nodes, each of which performs some ESQL functions. Finally, control is passed to an HTTP reply node which returns results back to the client.

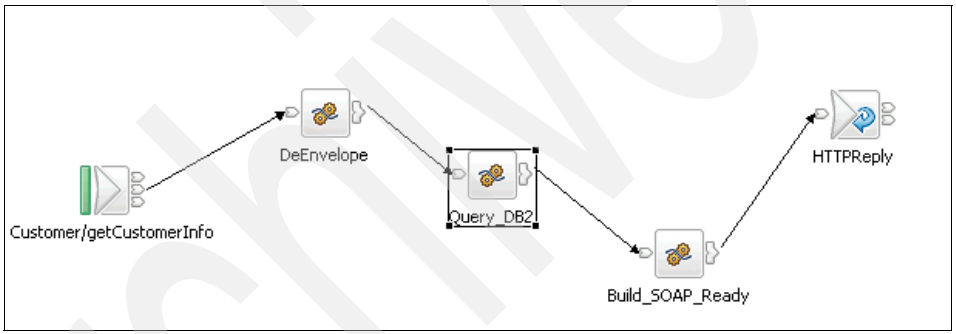

Figure 4-2 WMB Message flow

The nodes used in this scenario are described next.

Customer/getCustomerInfo HTTPInput node

In the HTTPInput node, we specified the request that the node will listen to. Our broker is listening on port 7080, so our request uses `http://wtsc47.itso.ibm.com:7080/Customer/getCustomerInfo`. The HTTP listener removes the HTTP address, leaving the request Customer/getCustomerInfo. It then matches the request with the information specified in the URL selector of HTTPInput node; this match is done from the most specific to the most generic data.

Compute nodes

We used the following three compute nodes:

- DeEnvelope
- Query_DB2
- Build_SOAP_Ready

Descriptions of these compute nodes follow.

DeEnvelope compute node

This node performs the following functions using ESQL:

- Takes the SOAP input message and parses out the customer ID. The SOAP message looks like this:

```
<?xml version="1.0" encoding="UTF-8" ?>
- <SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:q0="http://www.CustomerReq.com/schemas/CustomerReqInterface"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <SOAP-ENV:Body>
- <q0:CustomerReq>
  <q0:CustNo>001</q0:CustNo>
  </q0:CustomerReq>
  </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

- Creates procedures to copy the message headers and build the output SOAP message. The ESQL code looks as follows:

```
CREATE COMPUTE MODULE MsgFlow_Customer_DeEnvelope

   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      /* Remove the SOAP envelope and place customer ID in
"standard" plase */
      CALL CopyMessageHeaders();
      SET OutputRoot.XMLNS.CustomerReq.CustNo =
          InputRoot.XMLNS.*:Envelope.*:Body.*:CustomerReq.*:CustNo;
   END;

   CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER 1;
      DECLARE J INTEGER;
      SET J = CARDINALITY(InputRoot.*[]);
      WHILE I < J DO
         SET OutputRoot.*[I] = InputRoot.*[I];
         SET I = I + 1;
```

```
        END WHILE;
    END;

END MODULE;
```

The CREATE FUNCTION and CREATE PROCEDURE statements define a callable function or procedure, also known as a routine. In our example the function calls the procedure to parse the SOAP message header. Figure 4-3 shows the Compute node properties as displayed in the WMB Toolkit.

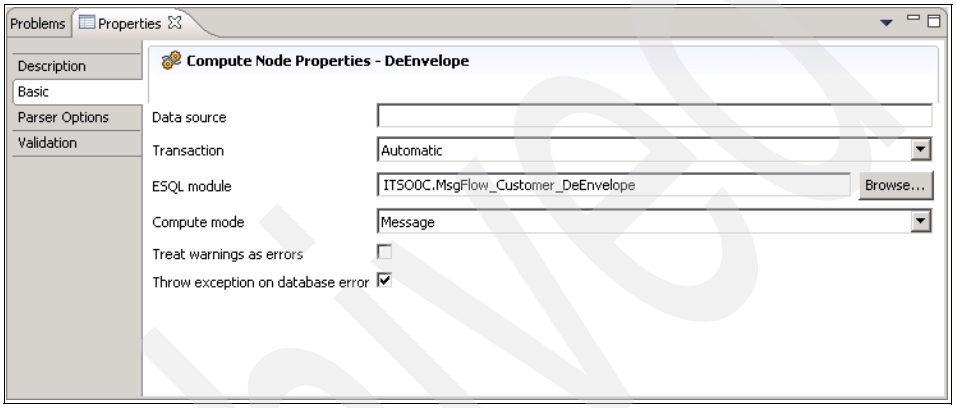

Figure 4-3 Compute node properties for DeEnvelope

Query_DB2 node

This node calls DB2 on z/OS to get the customer data for the credit check. Its objective is to call the DB2 stored procedure `ITSO0C.CREDINQJ` which resides on z/OS. This node has a Datasource property that is the DB2 location name of our z/OS DB2 subsystem. The DB2 location name can be found by looking in the DB2 master address space for the following message:

```
DSNL004I  -DB8K DDF START COMPLETE
          LOCATION  DB8K
          LU        USIBMSC.SCPDB8K
          GENERICLU -NONE
          DOMAIN    wtsc47.itso.ibm.com
          TCPPORT   38130
          RESPORT   38131
```

Thus, in our setup we used DB8K as the Datasource name in this compute node.

In order for the broker to access the DB2 defined as a data source in message flows it needs to have the connection defined in the broker's BIPDSNAO member

of the installation control file. Our DSNACLI as defined in BIPDSNAO shows the broker's connection to DB2 for its own table access as well as data sources defined for message flows. In our example the same DB2 subsystem is used for both but these may be different in your environment.

BIPDSNAO example

```
APPLTRACE=0
APPLTRACEfilename=/var/wmqi/MQBDBRK/output/traceodbc
CONNECTTYPE=2
DIAGTRACE=0
DIAGTRACE_NO_WRAP=0
MAXCONN=0
MULTICONTEXT=0
MVSDEFAULTSSID=DB8K
; SUBSYSTEM
DB8K
MVSATTACHTYPE=RRSAF
PLANNAME=DSNACLI

; DATASOURCES
DB8K
CURRENTSQLID=MQBDBRK
```

Here is our ESQL for the Query_DB2 compute node. The external name in the InquiryCustomerCredit procedure is the name of our z/SO DB2 stored procedure:

```
CREATE FUNCTION Main() RETURNS BOOLEAN
  BEGIN
      CALL CopyMessageHeaders();
      /* Call InquiryCustomerCredit */
      DECLARE CUSTLN CHAR;
      DECLARE CUSTFN CHAR;
      DECLARE CREDEXPIREDATE DATE;
      DECLARE CREDRESULT CHAR;
      CALL InquiryCustomerCredit(InputRoot.XMLNS.CustomerReq.CustNo,
                                 CUSTLN,
                                 CUSTFN,
                                 CREDEXPIREDATE,
                                 CREDRESULT
      );
      SET OutputRoot.XMLNS.CustomerCredit.CustNo           =
InputRoot.XMLNS.CustomerReq.CustNo;
      SET OutputRoot.XMLNS.CustomerCredit.LastName         = CUSTLN;
```

```
      SET OutputRoot.XMLNS.CustomerCredit.FirstName        = CUSTFN;
      SET OutputRoot.XMLNS.CustomerCredit.CreditExpireDate =
CREDEXPIREDATE;
      SET OutputRoot.XMLNS.CustomerCredit.CreditResult     =
CREDRESULT;
      RETURN TRUE;
   END;

   CREATE PROCEDURE InquiryCustomerCredit(INOUT CustomerNo CHAR,
                                          OUT   CustomerLn CHAR,
                                          OUT   CustomerFn CHAR,
                                          OUT   CreditExpireDate
DATE,
                                          OUT   CreditResult CHAR)
                     LANGUAGE DATABASE
                     EXTERNAL NAME "ITSOOC.CREDINQJ"
    ;

   CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER 1;
      DECLARE J INTEGER;
      SET J = CARDINALITY(InputRoot.*[]);
      WHILE I < J DO
         SET OutputRoot.*[I] = InputRoot.*[I];
         SET I = I + 1;
      END WHILE;
   END;

END MODULE;
```

Figure 4-4 shows how the Compute node properties look in the WMB Toolkit.

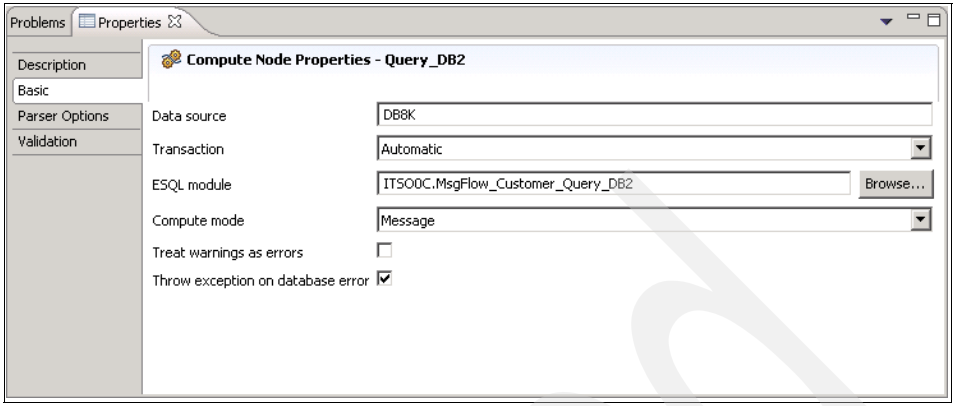

Figure 4-4 Compute node properties for DB2_Query

Build_SOAP_Ready node

This node builds the SOAP reply message and passes it to the HTTPReply node.

Here is our ESQL for the Build_SOAP_Ready compute node:

```
CREATE COMPUTE MODULE MsgFlow_Customer_Build_SOAP_Ready

   CREATE FUNCTION Main() RETURNS BOOLEAN
   BEGIN
      /* Declare the XML namespaces we will be using */
      /* DECLARE mySoapNS NAMESPACE
'http://schemas.xmlsoap.org/soap/envelope/';    */
      /* DECLARE myNS1 NAMESPACE 'http://itso.ibm.com';
*/
      /* DECLARE myXSI NAMESPACE
'http://www.w3.org/2001/XMLSchema-instance';       */
      /* DECLARE myXSD NAMESPACE 'http://www.w3.org/2001/XMLSchema';
*/

      CALL CopyMessageHeaders();

      /* Create a standard broker message Body for application data
with standard header */
      CREATE FIELD OutputRoot.XML;
      SET OutputRoot.XML.(XML.XmlDecl) = '';
      SET OutputRoot.XML.(XML.XmlDecl).(XML.Version) = '1.0';
      SET OutputRoot.XML.(XML.XmlDecl).(XML.Encoding) = 'UTF-8';
```

```
      /* Add the SOAP Envelope and Body and pointers  */
      CREATE FIELD OutputRoot.XML."soapenv:Envelope";
      Declare soapEnvelope REFERENCE TO
OutputRoot.XML."soapenv:Envelope";
      CREATE FIELD soapEnvelope."soapenv:Body";
      Declare soapBody REFERENCE TO soapEnvelope."soapenv:Body";

      /* Create the standard SOAP Envelope attributes */
      SET soapEnvelope.(XML.Attribute)"xmlns:soap" =
'http://schemas.xmlsoap.org/soap/envelope/' ;
      SET soapEnvelope.(XML.Attribute)"xmlns:soapenv" =
'http://schemas.xmlsoap.org/soap/envelope/' ;
      SET soapEnvelope.(XML.Attribute)"xmlns:q0" =
'http://www.CustomerCredit.com/schemas/CustomerCreditInterface' ;
      SET soapEnvelope.(XML.Attribute)"xmlns:xsd" =
'http://www.w3.org/2001/XMLSchema' ;
      SET soapEnvelope.(XML.Attribute)"xmlns:xsi" =
'http://www.w3.org/2001/XMLSchema-instance' ;

      /* Add the application-specific data */
      CREATE FIELD soapBody.CustomerCredit;
      Declare getCustomerCredit REFERENCE TO
soapBody.CustomerCredit;

      SET getCustomerCredit = InputRoot.XMLNS.CustomerCredit;
      SET getCustomerCredit.(XML.Attribute)"xmlns" =
'http://www.CustomerCredit.com/schemas/CustomerCreditInterface' ;

      RETURN TRUE;

   END;

   CREATE PROCEDURE CopyMessageHeaders() BEGIN
      DECLARE I INTEGER 1;
      DECLARE J INTEGER CARDINALITY(InputRoot.*[]);
      WHILE I < J DO
         SET OutputRoot.*[I] = InputRoot.*[I];
         SET I = I + 1;
      END WHILE;
   END;

END MODULE;
```

The SOAP output message that the compute node creates is shown below. You can see the response data for our example.

```
- <soapenv:Envelope
xmlns:q0="http://www.CustomerCredit.com/schemas/CustomerCreditInterf
ace" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
- <soapenv:Body>
- <CustomerCredit
xmlns="http://www.CustomerCredit.com/schemas/CustomerCreditInterface
">
  <CustNo>001</CustNo>
  <LastName>Chung</LastName>
<FirstName>SeungKeon</FirstName>
  <CreditExpireDate>2010-12-31</CreditExpireDate>
  <CreditResult>000</CreditResult>
  </CustomerCredit>
 </soapenv:Envelope>
```

Figure 4-5 shows how the compute note properties look in the WMB Toolkit.

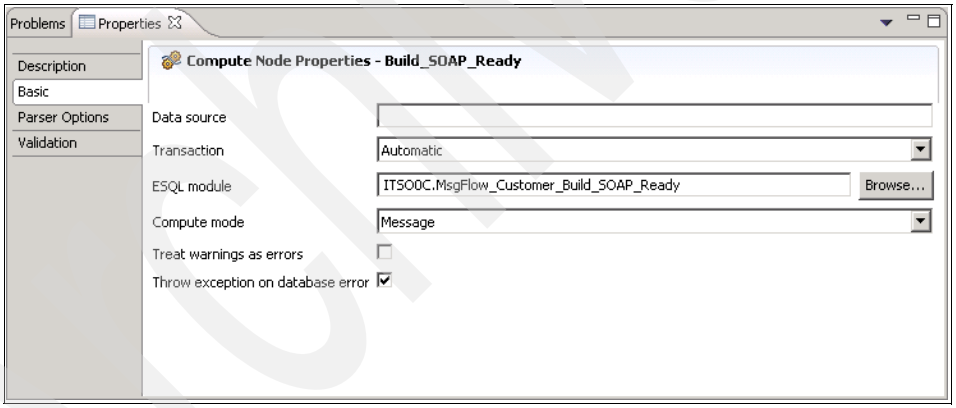

Figure 4-5 Compute node properties for Build_SOAP_Ready

HTTPReply node

The HTTP reply node passes the response data back to the requestor in a SOAP message.

4.3.2 Creating a Web Services interface for the message flow

In order to easily build a Web Services interface between the Order process in WPS and the message flow in WMB, we generated a WSDL document. This

WSDL document can then later be imported in WebSphere Integration Developer.

4.4 Additional material

Refer to Appendix A, “Additional material” on page 229 for instructions on downloading the sample material.

5

Order placement service

The CICS back-end order placement transactions used in our business process are based on existing CICS components in the CICS Catalog Manager application. We have modernized those existing components to make them easily accessible from our new SOA-based architecture. In this chapter we discuss the design approach behind the modernization of the CICS Catalog Manager application to make it suitable to integrate in our solution. We begin with a brief description of the Original CICS Catalog Manager application.

Important: If you construct the scenario in your own environment, be aware that you might need to choose you own naming conventions, as well as use your own variables for TCP/IP addresses, port number, host names, and so on. Please pay special attention to keeping these consistent. The names and variables we use in this publication are only meant as examples.

5.1 Original CICS Catalog Manager application

The CICS Catalog Manager sample application is a working COBOL application that is designed to illustrate best practices when connecting CICS applications to external clients and servers.

The example is constructed around a simple sales catalog and order processing application, in which the end user can perform these functions:

- List the items in a catalog
- Inquire on individual items in the catalog
- Order items from the catalog

The catalog is implemented as a VSAM file.

The base application, with its 3270 user interface, provides functions with which you can list the contents of a stored catalog, select an item from the list, and enter a quantity to order. The application has a modular design, which makes it simple to extend the application to support newer technology, such as Web services. Figure 2-1 shows the structure of the base application.

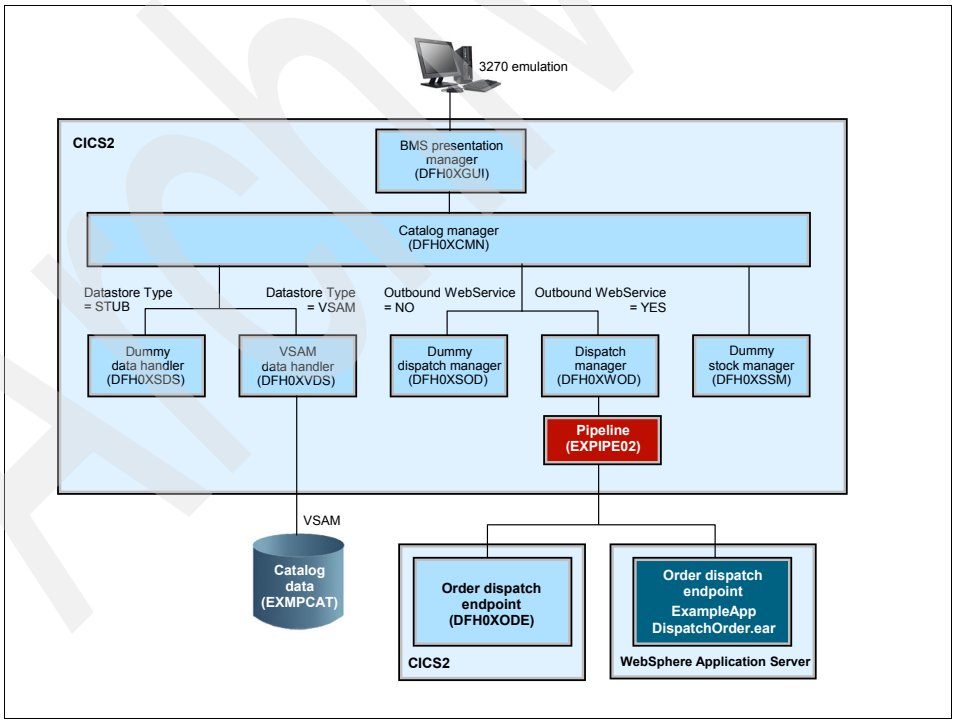

Figure 5-1 CICS Catalog Manager application design

The components of the base application are:

- A *BMS presentation manager (DFH0XGUI)* that supports a 3270 terminal or emulator, and that interacts with the main catalog manager program.
- A *catalog manager program (DFH0XCMN)* that is the core of the example application, and that interacts with several back-end components.
- The back-end components, which include:
  - A *data handler program* that provides the interface between the catalog manager program and the data store. The base application provides two versions of this program. They are the VSAM data handler program (DFH0XVDS), which stores data in a VSAM data set; and a dummy data handler (DFH0XSDS), which does not store data, but simply returns valid responses to its caller. Configuration options let you choose between the two programs.
  - A *dispatch manager program* that provides an interface for dispatching an order to a customer. Again, configuration options let you choose between the two versions of this program: DFHX0WOD is a Web service requester that invokes a remote order dispatch end point, and DFHX0SOD is a dummy program that simply returns valid responses to its caller. There are two equivalent order dispatch endpoints: DFH0XODE is a CICS service provider program; ExampleAppDispatchOrder.ear is an enterprise archive that can be deployed in WebSphere Application Server or similar environments.
  - A *dummy stock manager program (DFH0XSSM)* that returns valid responses to its caller, but takes no other action.

> **Note:** Visit the CICS Transaction Server Information Center at `http://publib.boulder.ibm.com/infocenter/cicsts/v3r1/index.jsp?topic=/com.ibm.cics.ts31.doc/dfhxa/topics/dfhxa_t100.htm` for a comprehensive review of the CICS Catalog Manager example application.

5.2 CICS Catalog Manager application modernization

As previously noted, the base application of the original CICS Catalog Manager example application has a modular design, which makes it possible to extend the application to support newer technology. We have taken advantage of this flexibility and modified, as well as extended, the base application beyond its core functionality.

The core functionality of the revised CICS application remains the same. The end user is still able to list the contents of a stored catalog, select an item from the list, and enter a quantity to order.

What has changed is the implementation of the back-end processes in CICS. We have implemented our CICS COBOL business logic as a reusable Web service via the CICS Service Flow Feature. CICS Service Flow Feature is a business service integration adapter for all CICS applications. It offers both tooling and run-time components. These components enable enterprise solution architects, enterprise integration developers, and CICS application specialists to create CICS business services for integration in service-oriented architectures, business process collaborations, or enterprise solutions that exploit a loose coupling approach. With the revised version of the CICS Catalog Manager Application, we have taken an IBM SOA approach to the application design and development.

5.2.1 CICS Service Flow feature

CICS Service Flow feature provides the capability to aggregate existing CICS programs into composite business services, which can in turn form part of a business process. The ability to aggregate several CICS-based screen flows into one complete business application service is useful in providing course grained services. CICS SFF fits in both a refacing and rejuvenating approach. CICS SFF allows you to eliminate 3270 screens, but also creates an opportunity to better componentize your application, as well as creating the proper service granularity. However, while aggregation of CICS programs into a service is simple and straightforward, decomposition of programs into smaller services cannot be done without a proper refactoring approach and tools.

Figure 5-2 and Figure 5-3 on page 73 illustrate the difference between a CICS application before and after using CICS SFF.

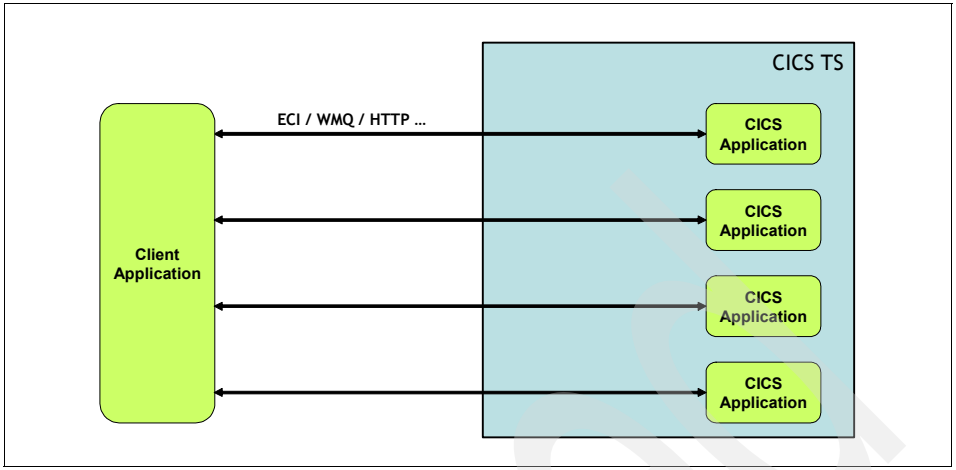

Figure 5-2 Traditional external access to CICS

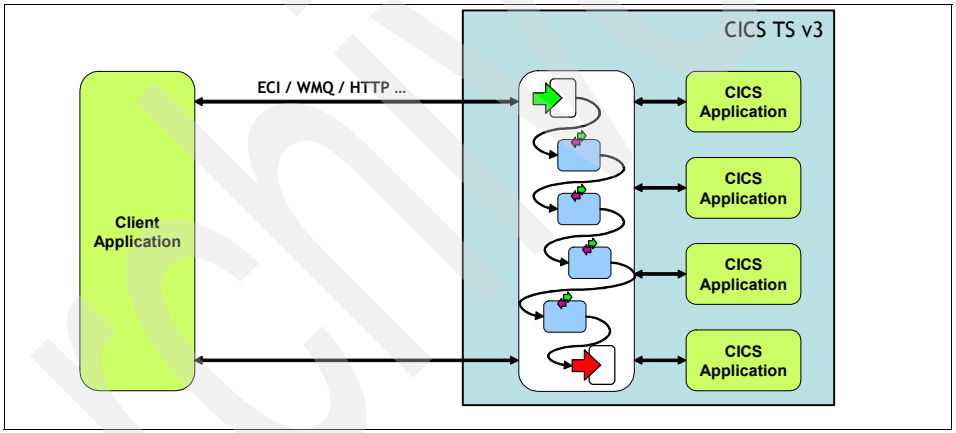

Figure 5-3 A composite CICS Web service after using CICS SFF

The aggregation of these programs into a usable service is necessary because it is unlikely that the functions contained within a single program would provide a complete business component. Even if the functions within a program are stateless, they are probably too fine grained to provide a complete service. Thus the need for aggregation.

You might find, once aggregation is complete, that the converse applies. This is due to the fact that the existing programs are likely to contain some embedded business process choreography resulting in a course grained service. A service might become so course grained that its flexibility, and therefore the likelihood of its possible reuse, is reduced or eliminated.

In cases where the luxury of functional decomposition and rejuvenation is not available, the CICS Service Flow Feature is an extremely useful tool in orchestrating functions across multiple CICS applications into usable business application components. An added benefit is that the Quality of Service (QoS) delivered by CICS and the System z platform is in no way compromised.

The logical structure of a flow designed with CICS SFF is shown in Figure 5-4.

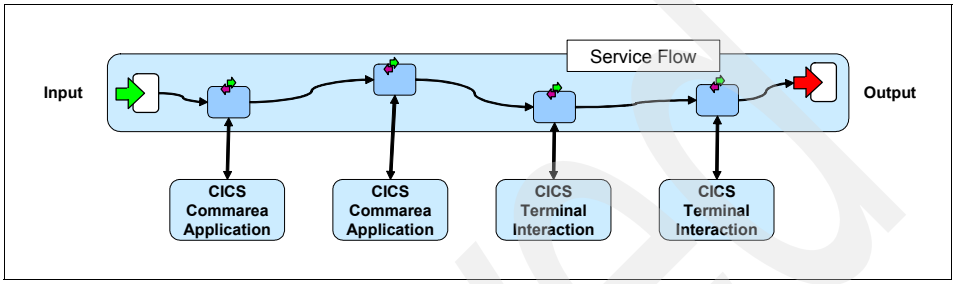

Figure 5-4 A flow in CICS SFF

CICS Service Flow Feature consists of two components:

- The *Service Flow Runtime*, which is a no-charge separately orderable feature of CICS TS 3.1 and later.
- The *Service Flow Modeler*, which is included in WebSphere Developer for System z Version 7 and Rational Developer for System z (RDz).

The overall deployment model of CICS SFF applications is shown in Figure 5-5.

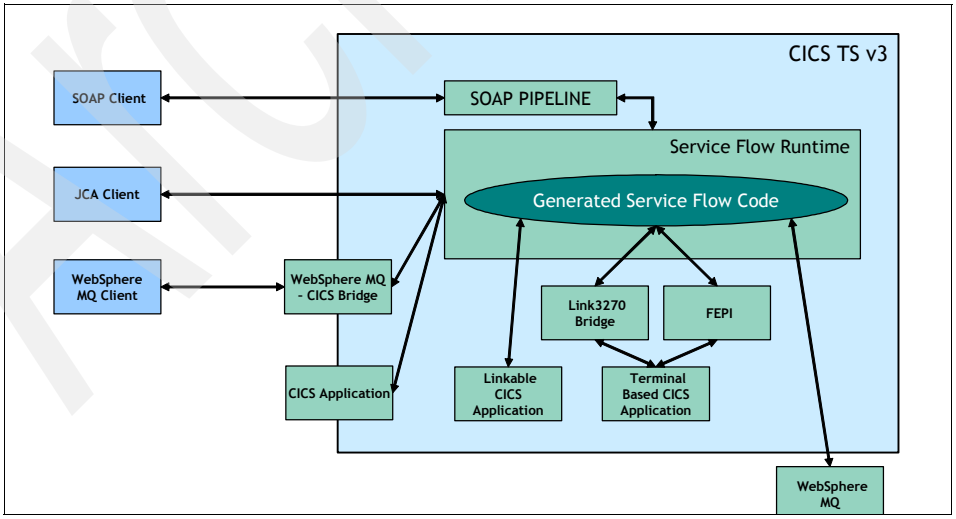

Figure 5-5 Overall deployment model of CICS SFF

The overall service flow and any relevant decision point must be described within the Service Flow Modeler (SFM). SFM is a component of RDz and allows a developer to define flows, nodes and message mappings graphically. CICS SFF allows both business logic and 3270 based programs to participate within a service flow.

5.3 Environment

The following sections describe our environment on z/OS.

5.3.1 CICS application

The following CICS modules are used in this service:

- SOA0XQRY accesses the DB2 ITEMS table to query the entire catalog. This code was modified to build this scenario from the CICS sample module DFH0XVDS.
- SOA0XQR1 accesses the DB2 ITEMS table to query a single catalog entry. This code was modified to build the SOA scenario from the CICS sample module DFH0XVDS.
- SOA0XORD updates the quantity of the item in stock in the ITEMS table at the placement of an order.
- SOA0XSSM is a stubbed version of the stock replenishment manager. This code was modified to build the SOA scenario from CICS sample module DFH0XSSM.
- SOA0XS1 defines the BMS map for the menu and order panels. This map is modified to build the SOA scenario from sample module DFH0XS1.
- SOA0XS2 defines the BMS map for the inquire panel. This map is modified to build the SOA scenario from sample module DFH0XS2.

5.3.2 DB2 tables

Two DB2 tables are used in this service:

- ITEMS table
- ORDERS table

DB2 ITEMS table

The ITEMS table has the following layout:

```
ITEM_NO                 CHAR(04) NOT NULL,
  ITEM_DESC               CHAR(40),
  ITEM_DEPT               CHAR(03),
  ITEM_COST               DECIMAL(5,2),
  ITEM_IN_STOCK           SMALLINT,
  ITEM_ON_ORDER           SMALLINT,
  ITEM_ON_PROCURE         SMALLINT,
PRIMARY KEY (ITEM_NO)
```

A sample job to define a table space and the table is included in the additional material. The name is @DB2ITEM.

DB2 ORDER table

The ORDERS table has the following layout:

```
ORDER_NO                CHAR(27) NOT NULL,
  ORDER_CUST_NO           CHAR(03),
  ORDER_CHARGE_DEPT       CHAR(04),
  ORDER_ITEM_NO           CHAR(04),
  ORDER_ITEM_QUANTITY     SMALLINT,
  ORDER_ITEM_UNIT_COST    DEC(5,2),
  ORDER_ITEM_TOTAL_COST   DEC(9,2),
PRIMARY KEY (ORDER_NO)
```

A sample job to define a table space and the table is included in the additional material. The name is @DB2ORDER.

5.4 The process of creating the service flow

A service flow is created by selecting the Service Flow Project wizard from the EST tools. The complete service flow of the Order placement service is shown in Figure 5-6 on page 77.

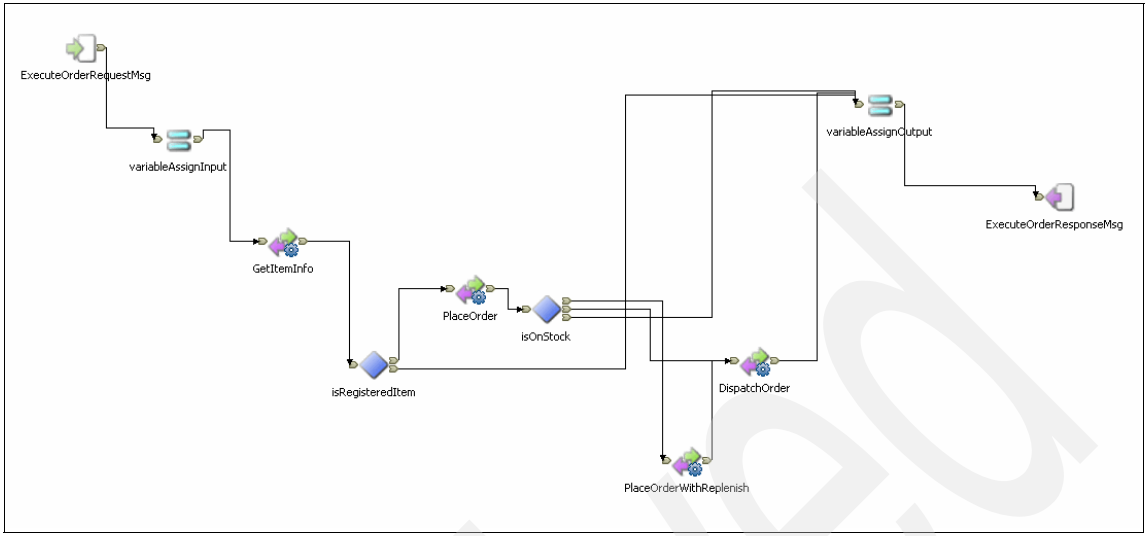

Figure 5-6 Order placement service flow

5.4.1 Nodes

In this section we provide some information on each of the nodes.

ExecuteOrderRequestMsg node

This node is the receive node, receiving the input message for the flow.

variableAssignInput

This node is an Assign node and is used to map and eventually transform variables before the first program is executed.

Figure 5-7 on page 78 shows the details of our mapping. The request contains four input fields:

- CustNo
- ChargeDept
- ItemNo
- Quantity

In the next CICS program, SOA0XQR1, represented in the GetItemInfo node, we run a query for the Item number on the DB2 ITEMS table. We are only interested in passing the ItemNo from the original request right now, because this is the only key field for the DB2 query. A small arrow to the left of a field indicates that a mapping exists between a source and a target field.

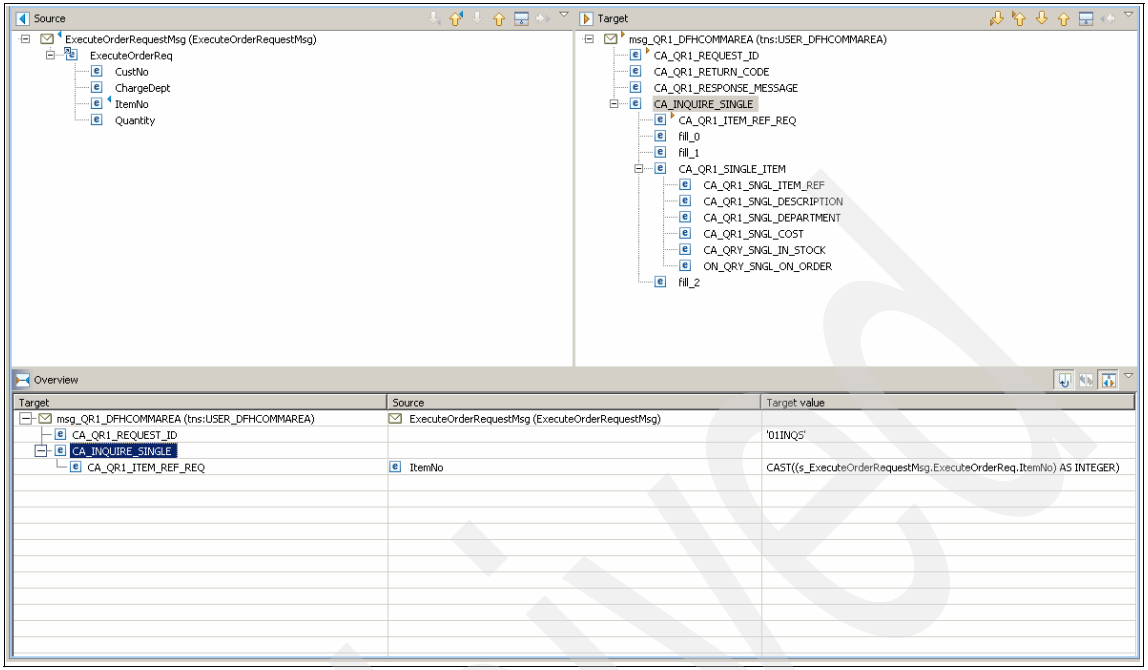

Figure 5-7 variableAssignInput node mappings

GetItemInfo

This node is an Invoke node representing an actual CICS program. The program executed in this node is the SOA0XQR1 program. This program runs a query on the DB2 ITEMS table based on the Item number. It returns a number of fields for the item queried, or a return code for certain error conditions. You can take a look at the COBOL source (included in the additional material) for more details on the exact functionality.

You can double-click the node to open the Operations Editor, which gives you a quick overview of the nodes and their inputs and outputs. See Figure 5-8 on page 79.

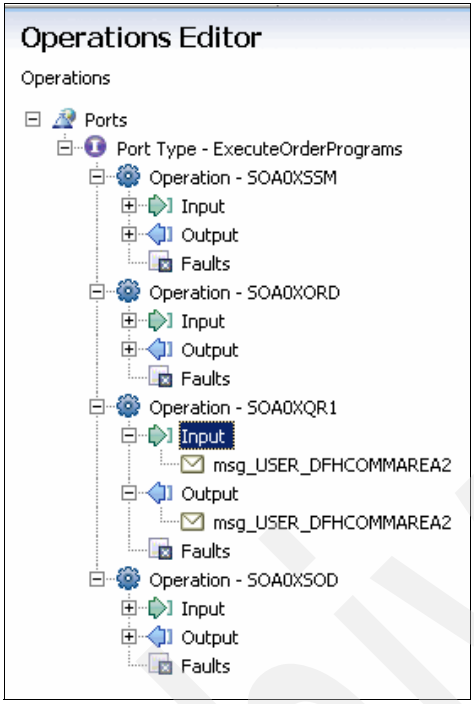

Figure 5-8 Operations Editor

Double-clicking on the small input and output icons of the node will open a window with details on the input and output message set.

isRegisteredItem

This node is a choice node. When the Item number exists, the return code is 00, if the Item number does not exists, the return code from the program is 20. If the Item does not exist, a response message is fed back to the calling service consumer. In that case only a variable mapping is done in node variableAssignOutput. If the Item does exist, the SOA0XORD program is invoked in node PlaceOrder.

PlaceOrder

This node is an Invoke node representing an actual CICS program. The program executed in this node is the SOA0XORD program. This program places the order by simply subtracting the ordered quantity from the quantity on stock in the ITEMS table. You can take a look at the COBOL source (included in the additional material) for more details on the exact functionality.

Again, you can double-click the node to open the Operations Editor, which gives you a quick overview of the nodes and their input and output.

The PlaceOrder Invoke node is followed by another choice node, isOnStock.

isOnStock

This choice node has 3 possibilities, depending on the return code from the SOA0XORD program:

- Return code 97
  The order is placed with replenishment and program SOA0XSSM is invoked in node PlaceOrderWithReplenish.
- Return code 00
  The order is placed normal and program SOA0XSOD is invoked in node DispatchOrder
- Return code other than 97 or 00
  A response message is fed back to the calling service consumer. In that case only a variable mapping is done in node variableAssignOutput.

PlaceOrderWithReplenish

This node is an Invoke node representing an actual CICS program. The program executed in this node is the SOA0XSSM program. This program adds the ordered quantity to the quantity on order in the ITEMS table. See the COBOL source (included in the additional material) for more details on the exact functionality.

Again, you can double-click the node to open the Operations Editor and view the nodes and their input and output.

After this action has taken place, the SOA0XSOD program is invoked in node DispatchOrder.

DispatchOrder

The Order dispatch program, SOA0XSOD wraps up the order by writing some statistics to the DB2 tables, calculating the Order total cost and writing a record to the ORDERS table. See the COBOL source (included in the additional material) for more details on the exact functionality.

Based on the return code a message is fed back to the variableAssignOutput node, which passes back the response message and the other fields to the service consumer.

variableAssignOutput

This node performs the variables mapping before sending the request back to the service consumer. Figure 5-9 shows the entire mapping in this node.

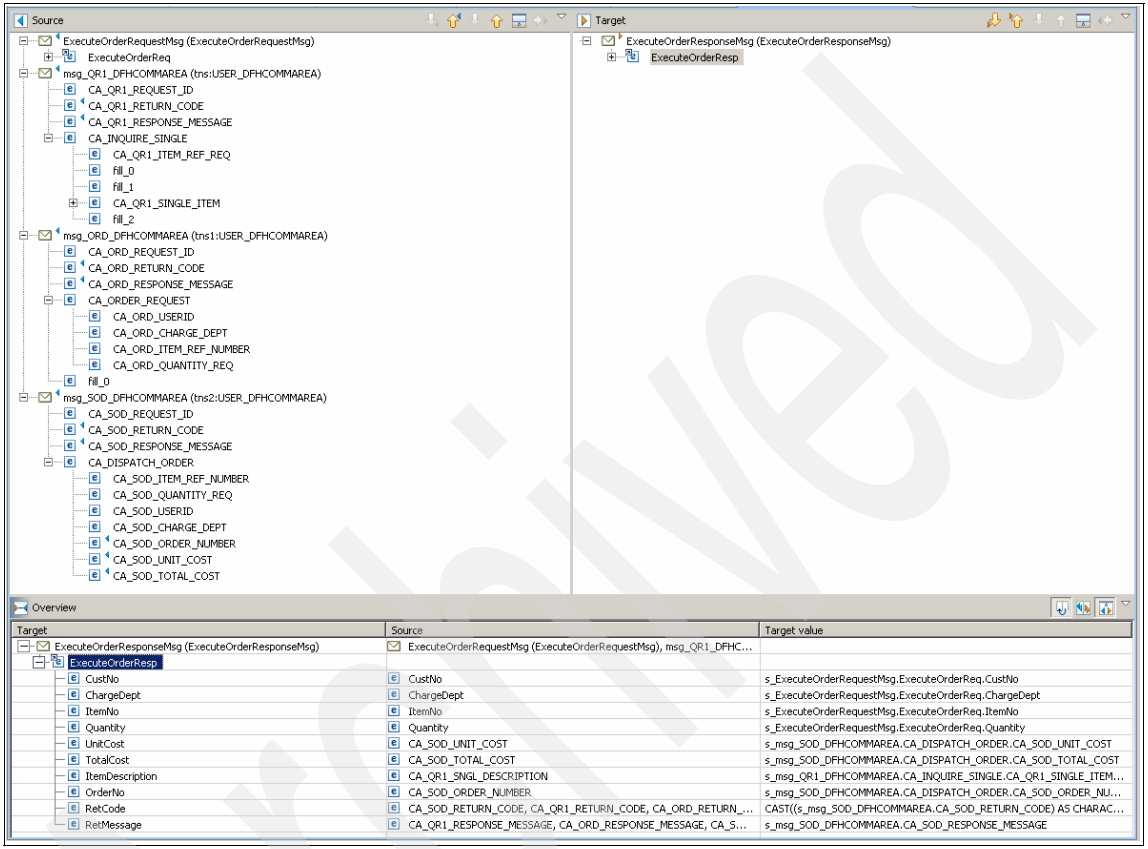

Figure 5-9 variableAssignOutput mappings

5.4.2 ESQL

The expressions for the Choice nodes are written as ESQL code. Example 5-1 shows the expressions created for this flow.

Example 5-1 ESQL code

```
CREATE FILTER MODULE isOnStock_Expression_1
   CREATE PROCEDURE Main(
      IN msg_ORD_DFHCOMMAREA REFERENCE
{'http://cobol.soa0xord.com/schemas'}:USER_DFHCOMMAREA
```

```
) RETURNS BOOLEAN
   BEGIN
      IF msg_ORD_DFHCOMMAREA.CA_ORD_RETURN_CODE = 97 THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END IF;
   END;

END MODULE;


CREATE FILTER MODULE isOnStock_Expression
   CREATE PROCEDURE Main(
      IN msg_ORD_DFHCOMMAREA REFERENCE
{'http://cobol.soa0xord.com/schemas'}:USER_DFHCOMMAREA

) RETURNS BOOLEAN
   BEGIN
      IF msg_ORD_DFHCOMMAREA.CA_ORD_RETURN_CODE = 00 THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END IF;
   END;

END MODULE;

CREATE FILTER MODULE isRegisteredItem_Expression
   CREATE PROCEDURE Main(
      IN msg_QR1_DFHCOMMAREA REFERENCE
{'http://cobol.soa0xqr1.com/schemas'}:USER_DFHCOMMAREA

) RETURNS BOOLEAN
   BEGIN
      IF msg_QR1_DFHCOMMAREA.CA_QR1_RETURN_CODE = 00 THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END IF;
   END;

END MODULE;
```

5.4.3 Generation of a service flow

Once a service flow has been modeled it must be deployed to CICS. A driver program and JCL are necessary to run the flow on z/OS. The generation process is determined by the properties in a WSDL file. Figure 5-10 shows the properties as we used them.

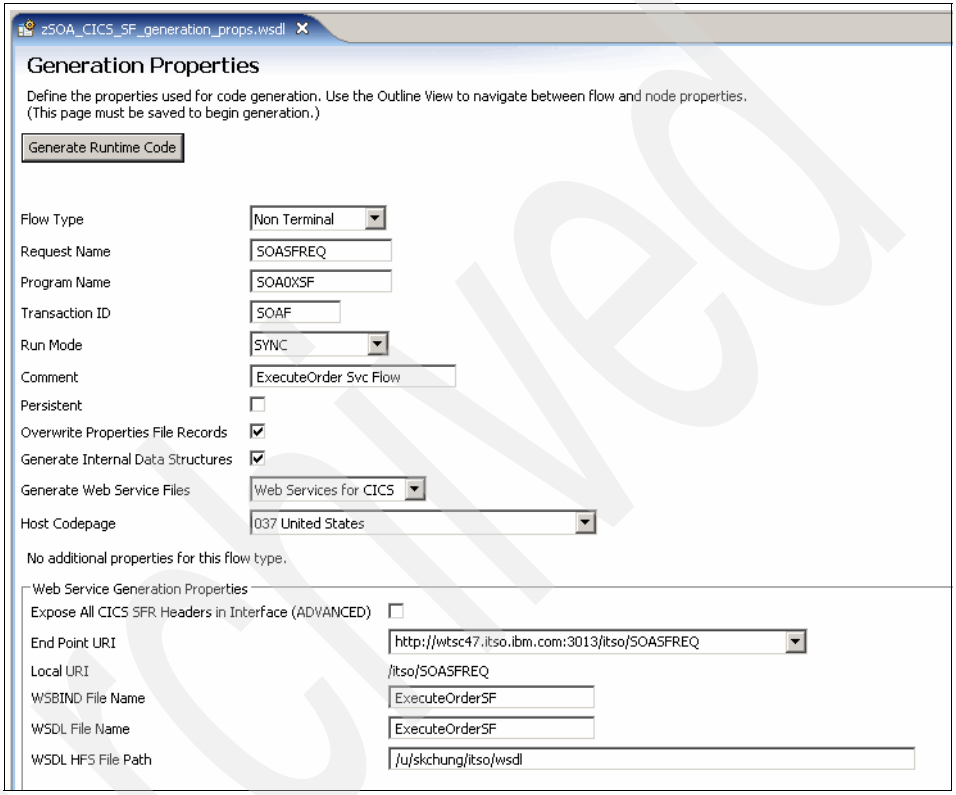

Figure 5-10

5.5 Additional material

Refer to Appendix A, “Additional material” on page 229 for instructions on obtaining the sample material.

6

Creating the Business Process

Once all back-end services are ready, work can start to build the Order process. In the previous chapters we have explained how to create Web service-enabled functions for Customer inquiry, Order placement and Credit check. The result is that we have a WSDL document for each one and we can use these WSDL documents to tie in services into our Order process.

In this chapter we discuss how to build the Order process using WebSphere Integration Developer. At a high level the steps are:

1. Create a project and import the WSDL documents of your services
2. Test the services
3. Create the Assembly Diagram
4. Design the process
5. Add a Human Task
6. Test the process

Important: If you construct the scenario in your own environment, be aware that you may need to choose you own naming conventions, as well as use your own variables for TCP/IP addresses, port number, host names, and so on. Please pay special attention to keeping these consistent. The names and variables we use in this publication are only meant as examples.

6.1 Creating a project

To create a project in WID, perform the following steps:

1. Start the WID workbench, and specify the workspace, as shown in Figure 6-1. Click **OK**.

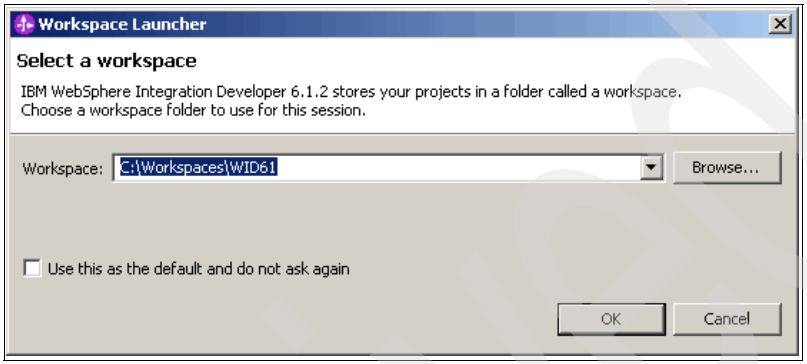

Figure 6-1 Selecting workspace

2. Create a new Module by selecting **File** → **New** → **Other**, as shown in Figure 6-2.

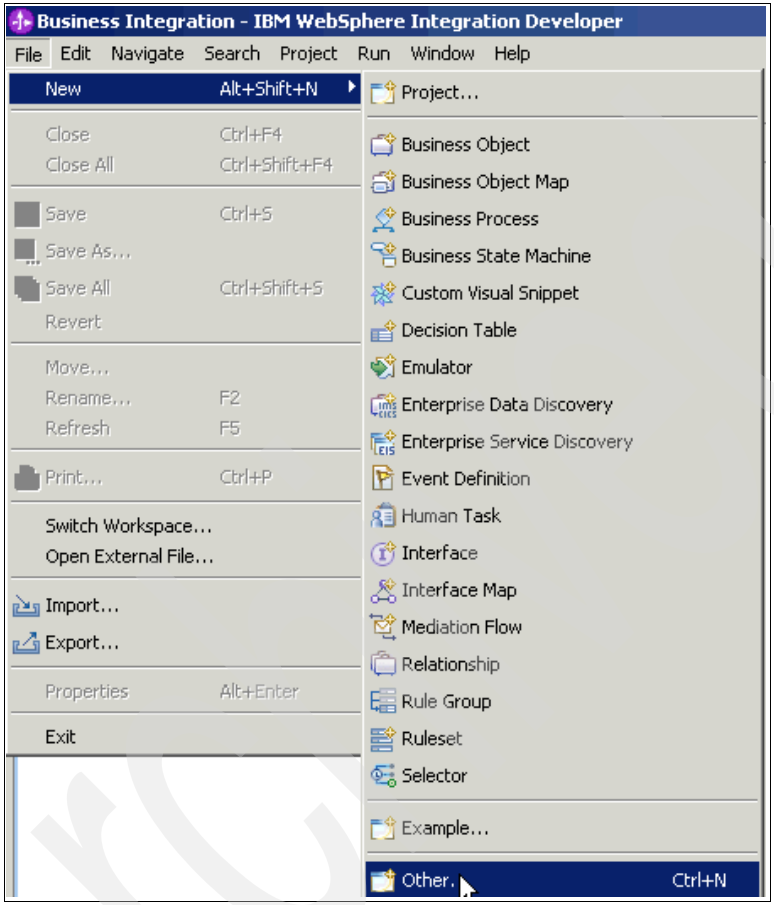

Figure 6-2 Creating a new Module

3. In the “Select a wizard” dialog, select **Module** (Figure 6-3). Click **Next**.

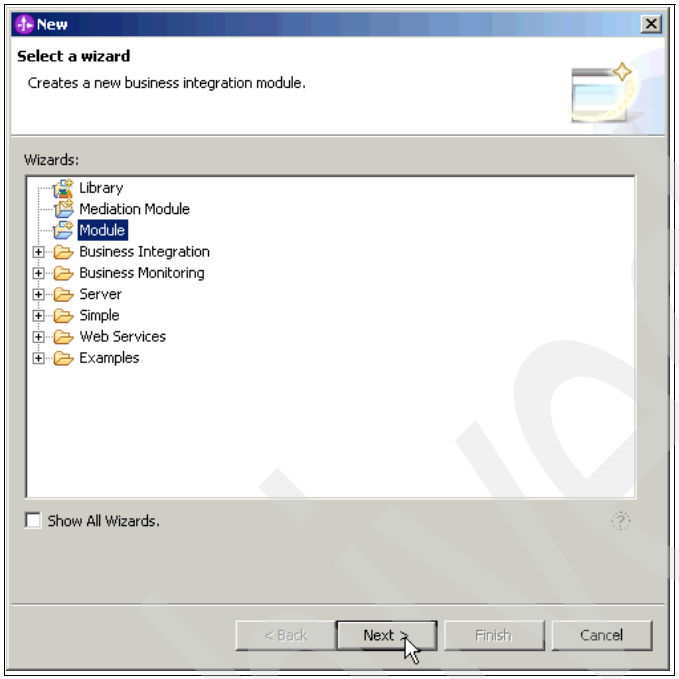

Figure 6-3 Select a Module

4. In the **New Module** dialog, type the name of the module, which in our case is `OrderProcess`, as shown in Figure 6-4. Leave all defaults.

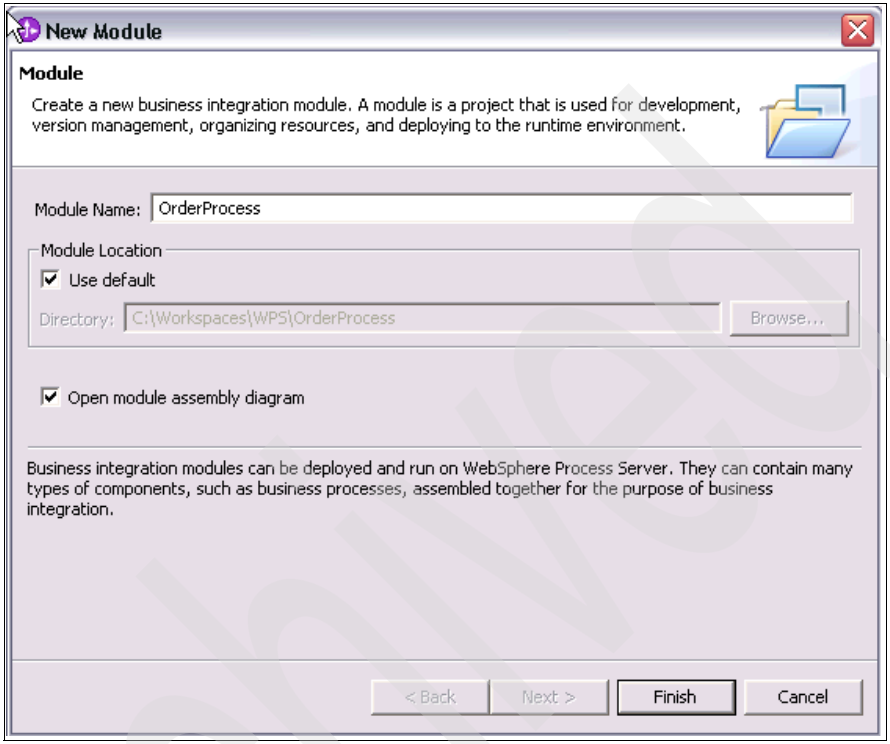

Figure 6-4 Naming the new Module

5. Click **Finish**.

   Your Business Integration view will look as shown in Figure 6-5. At the same time the Assembly Diagram window opens.

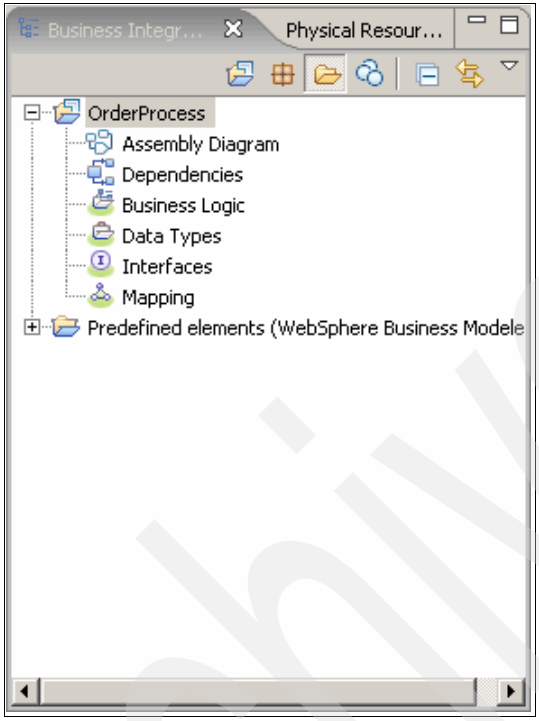

Figure 6-5 Business Integration view after Module creation

6. Import the WSDL documents of the services previously created. Select **Interfaces,** right-click and select **Import** (Figure 6-6).

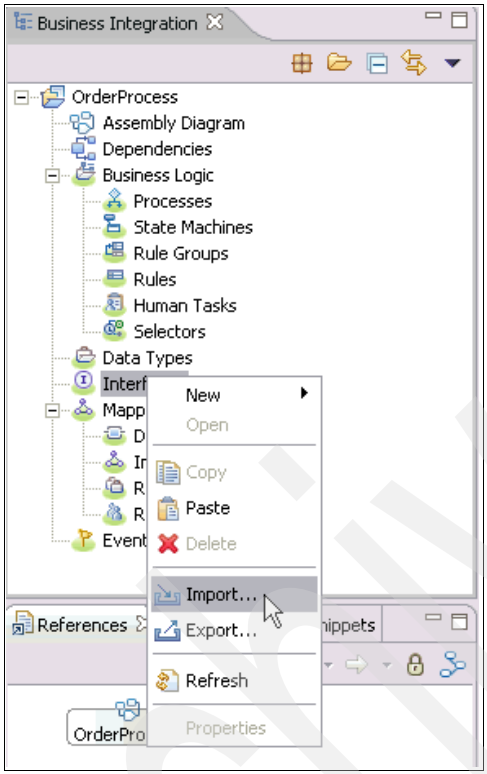

Figure 6-6 Importing WSDLs

7. On the Import dialog, select **WSDL/Interface** (Figure 6-7). Click **Next.**

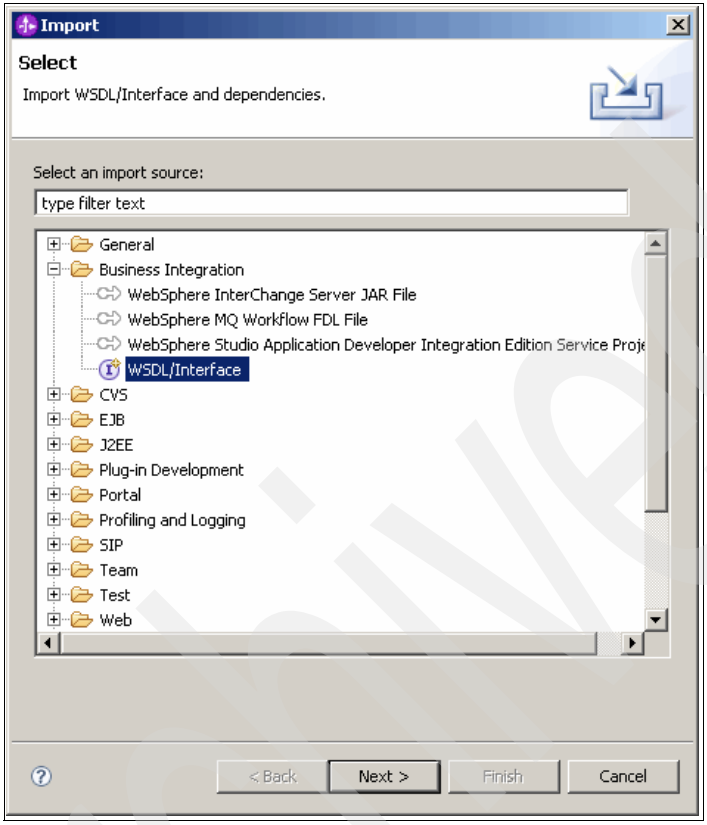

Figure 6-7 Import wizard

8. In the **Import from** directory field, browse to the directory where the four WSDLs are located using the **Browse** button (Figure 6-8).
9. In the WSDL/Interface Import, mark the **WSDL** field on the left. This will mark all the WSDLs. In addition to the three WSDL documents of the services created earlier there is also a WSDL document describing the interface to the Order process. We use this WSDL later to create the user interface that will call this Order process. Click **Finish**.

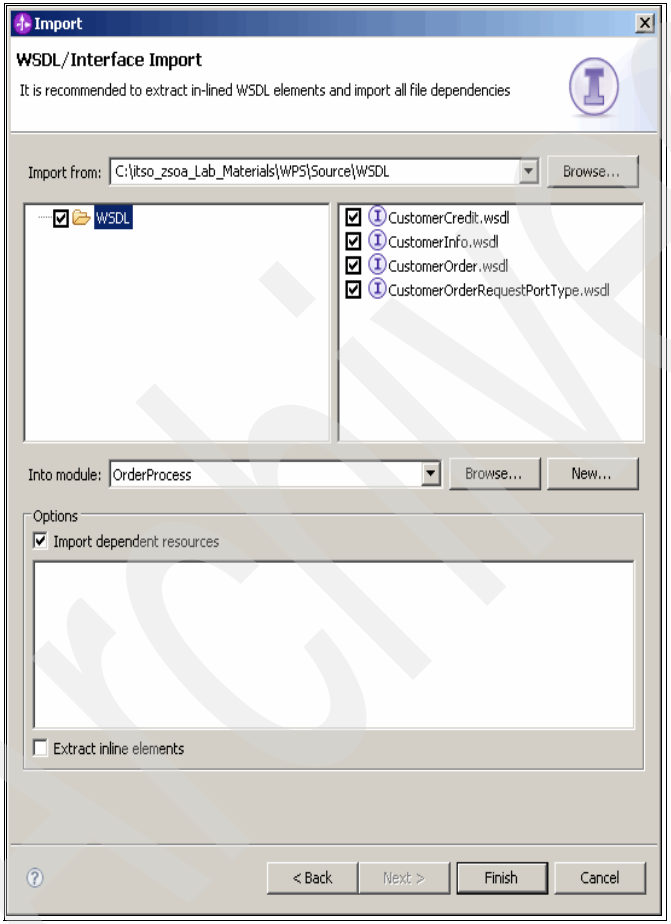

Figure 6-8 Selecting all the WSDLs

After importing the WSDLs, your Business Integration View will look as shown in Figure 6-9.

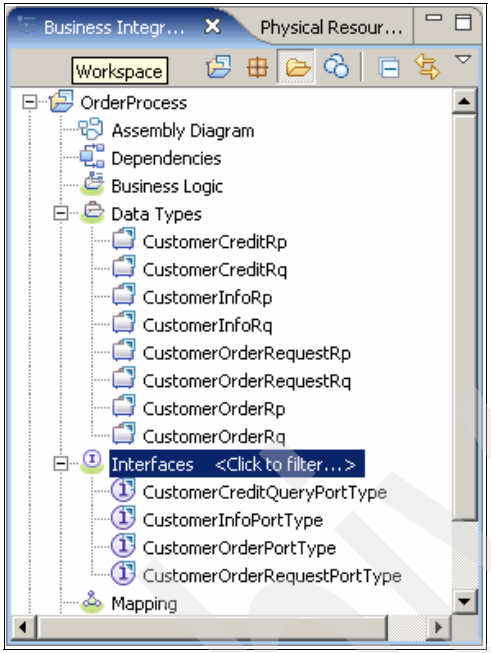

Figure 6-9 Business Integration view after importing WSDLs

6.2 Testing the services

The imported WSDLs are now listed in the Interfaces folder in the module OrderProcess. In order to test the WSDLs, the endpoints must be modified to match your environment. First, customize the WSDLs to point to the ports that the specific service provider is listening on. We demonstrate this for the Customer inquiry service:

1. Expand the **Web Service Ports** folder of your module and double-click **CustomerInfoPort**. This is the WSDL for the Customer inquiry service created in Chapter 3, “Customer inquiry service” on page 33. A new window will open as shown in Figure 6-10 on page 95. This window displays the information in the CustomerInfo.wsdl file in an organized manner.
2. Make sure the **CustomerInfoPort** field is highlighted.

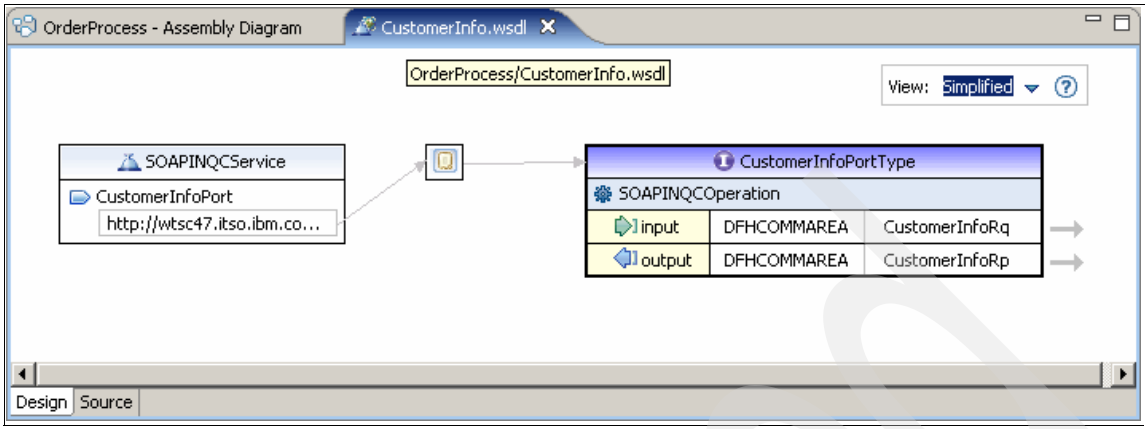

Figure 6-10 CustomerInfo.wsdl

3. Go to the Properties View located in the lower part of the workspace, and make sure the host name and port number match the ones for your CICS region. See Figure 6-11.

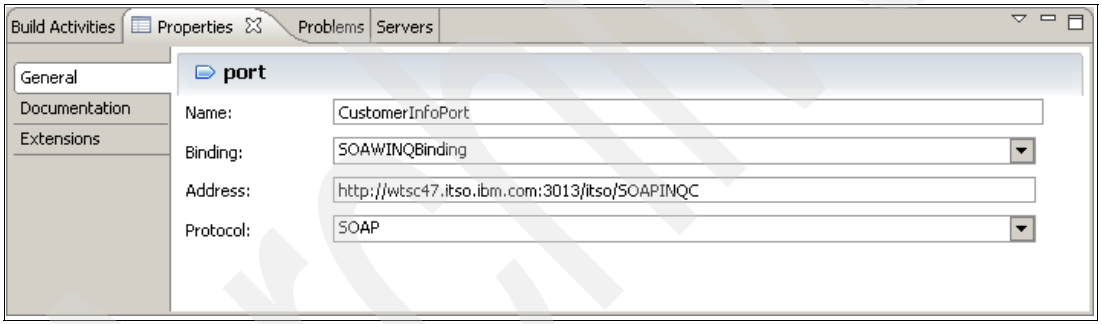

Figure 6-11 Verify/change the port number of the CustomerInfo Web service

Follow the same process for the other two WSDLs. Once you have set the endpoints correctly you can test the Web services.

Testing the Customer Inquiry Service

Perform the following steps to test the Customer inquiry service:

1. Select **Web Services Ports** → **CustomerInfoPort**, right-click and select **Web Services** → **Test with Web Services Explorer**, as shown in Figure 6-12 on page 96.

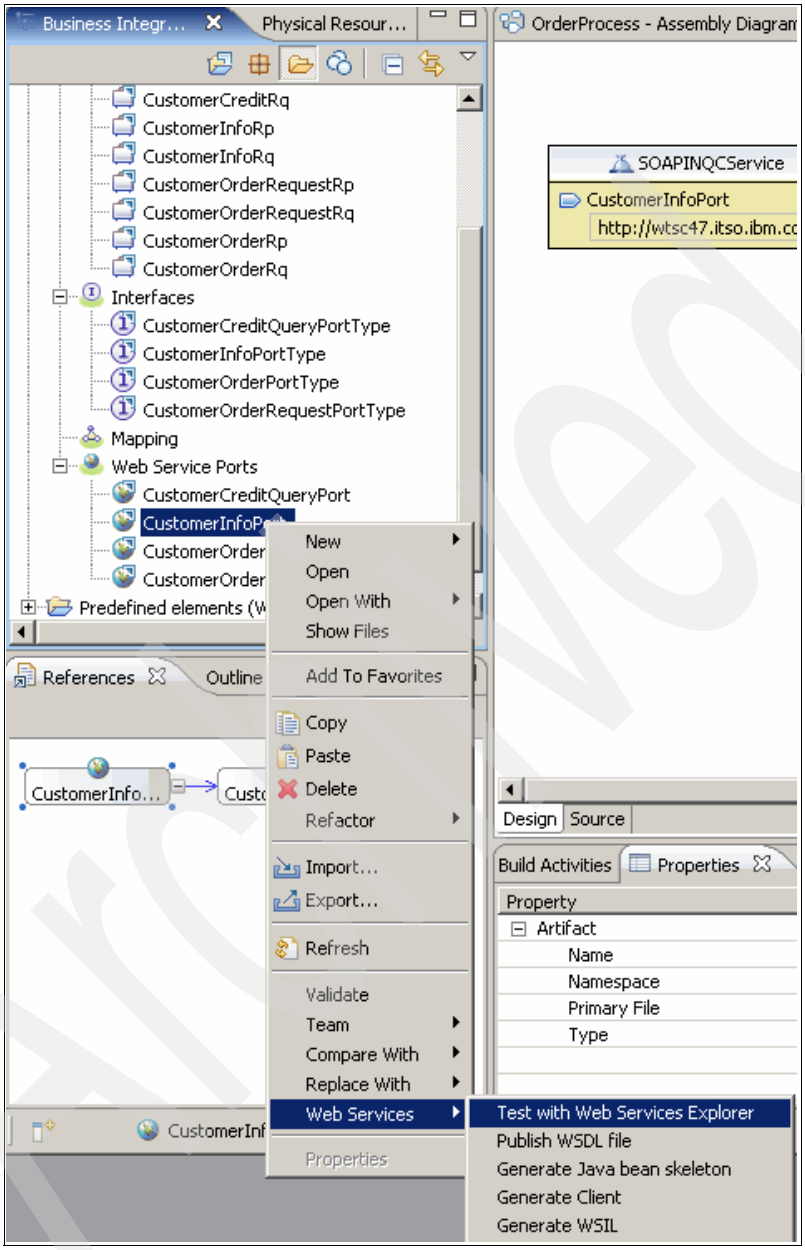

Figure 6-12 Testing CustomerInfo.wsdl with Web Services Explorer

2. In the Web Services Explorer window, fully expand the **SOAPINQCService** service under WSDL Main, and select **SOAPINQCOperation.** This will give you an input field where the required input for the CICS transaction can be typed. See Figure 6-13.

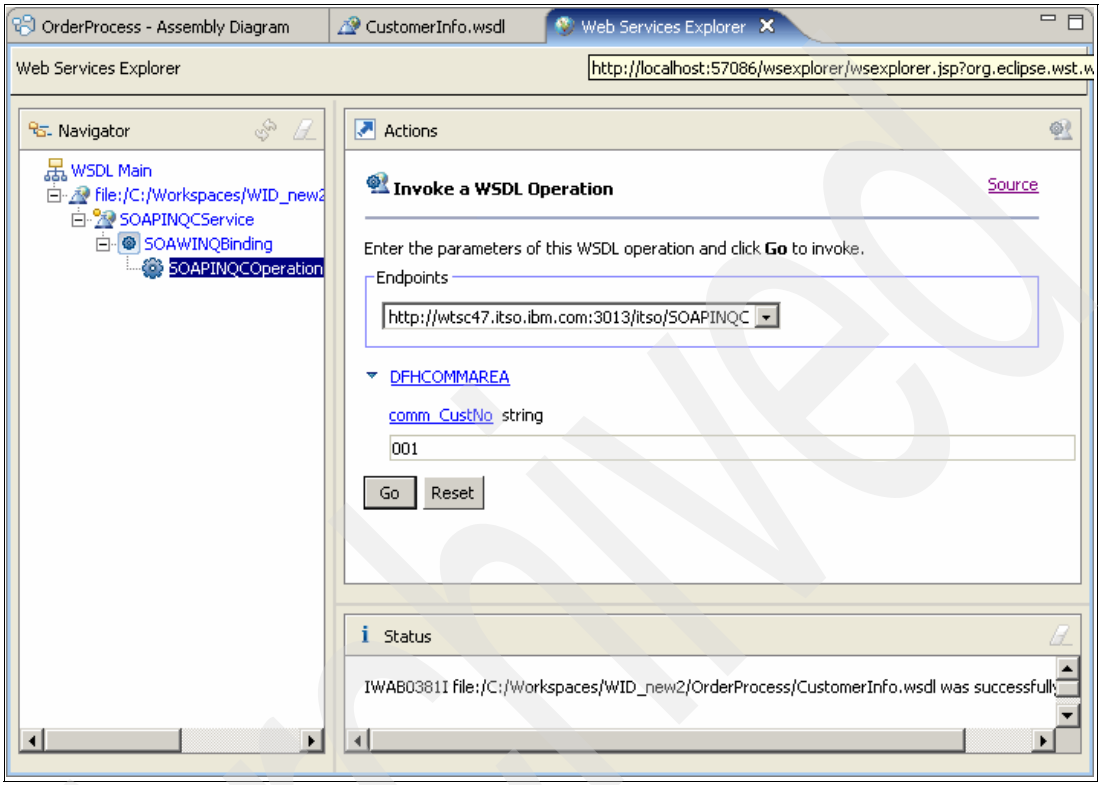

Figure 6-13 Expanding the service

3. Type a valid customer number, such as 001, in the “comm_CustNo” field and click **Go**. You should see the result of the CICS transaction in the Status area, as shown in Figure 6-14.

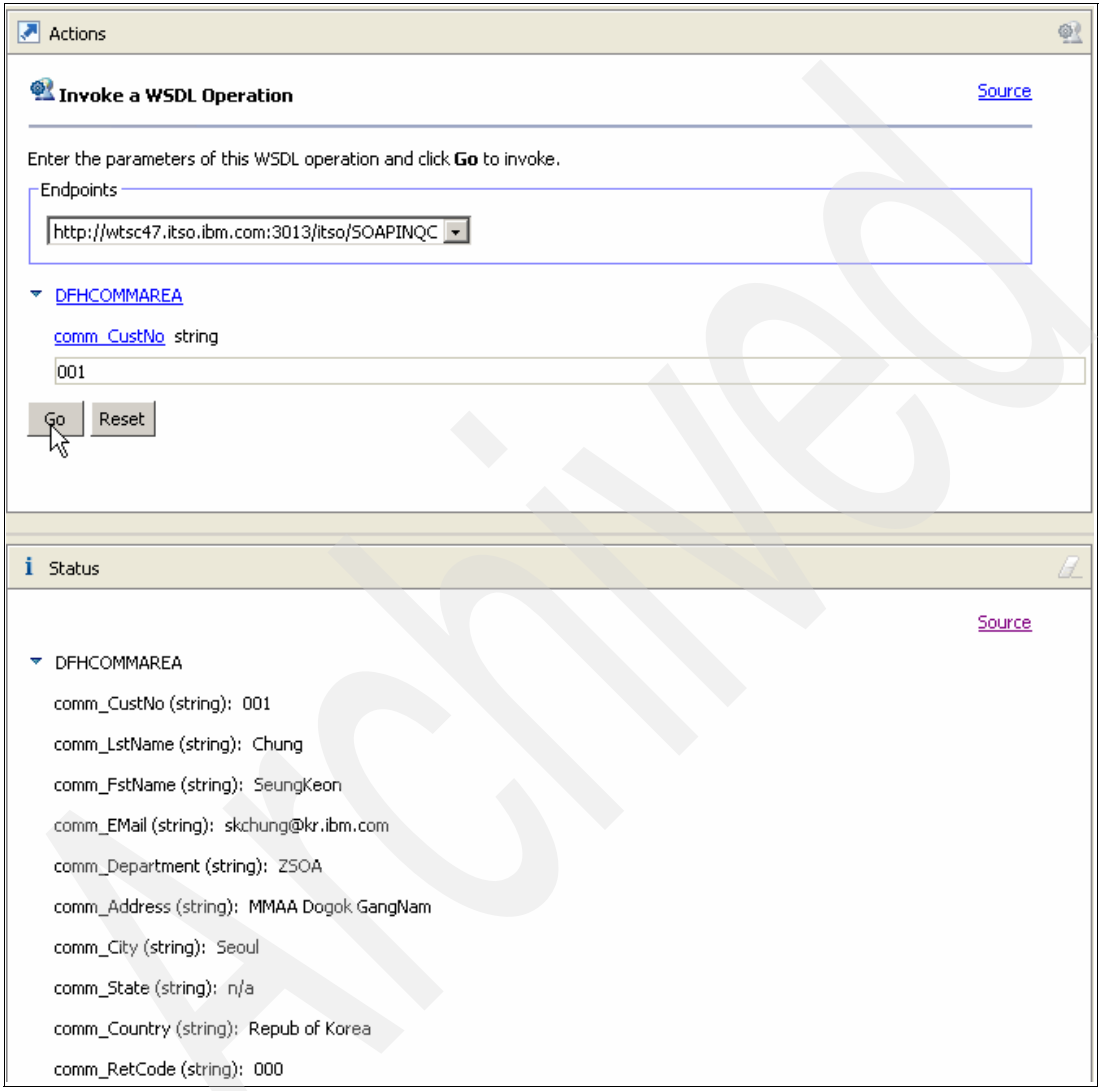

Figure 6-14 Testing the CustomerInfo service with output

Perform the same steps to test the Credit check and the Order placement services. Make sure you expand the WSDL document in full, so you reach the point where you can insert the input values.

6.3 Creating the Assembly Diagram

In the next steps we demonstrate how to create the process that will use the WSDLs just tested.

1. Under the **Components** tab select the **Process** icon from the palette on the left side of the editor and drag (move the mouse cursor *without* holding down the mouse button) and drop it (left-click) on the canvas area. See Figure 6-15.

\

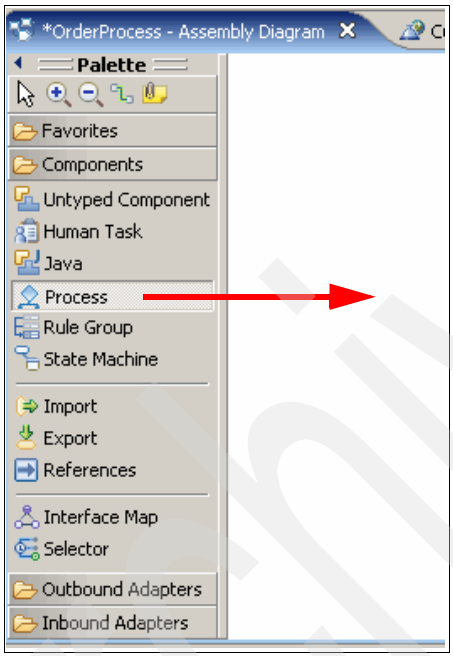

Figure 6-15 Opening the Assembly Diagram

2. Click *once* on the name **Component1** and rename it `OrderProcess`, as shown in Figure 6-16 on page 100. Press **Enter**.

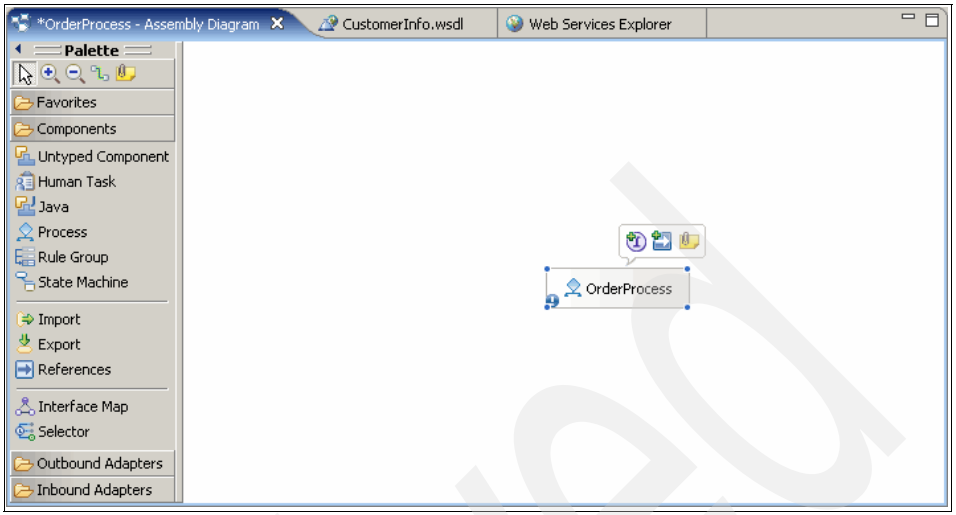

Figure 6-16 Renaming the component

3. Select **CustomerOrderRequestPortType** from the **Interfaces** folder, and drag and drop it into the canvas, to the left of the OrderProcess. On the Component Creation dialog select **Export with Web Service Binding** and click **OK** (Figure 6-17).

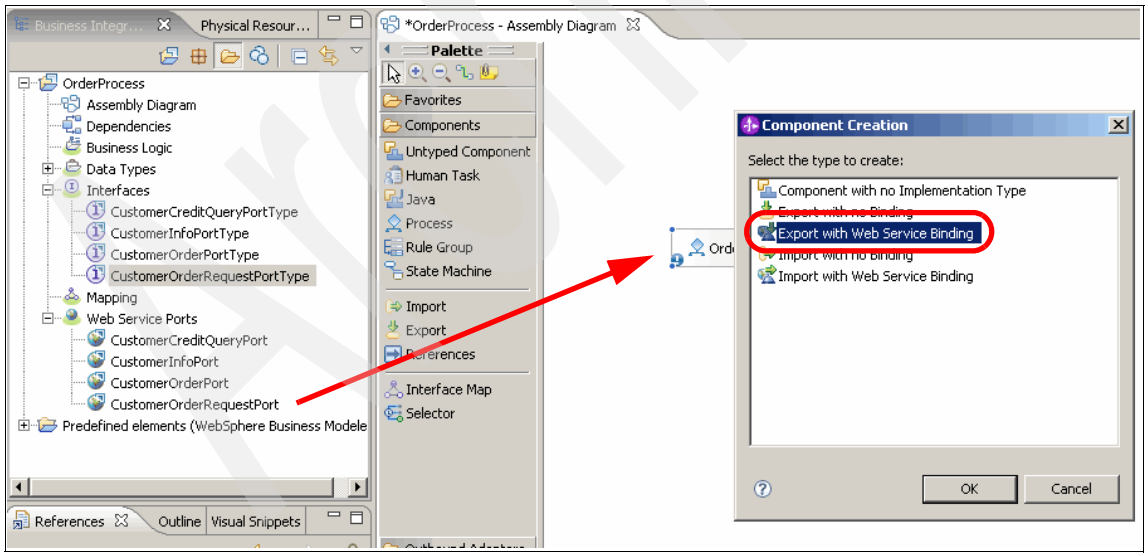

Figure 6-17 Export with Web Service binding

4. Select **soap/http** as the transport, and click **OK** (Figure 6-18).

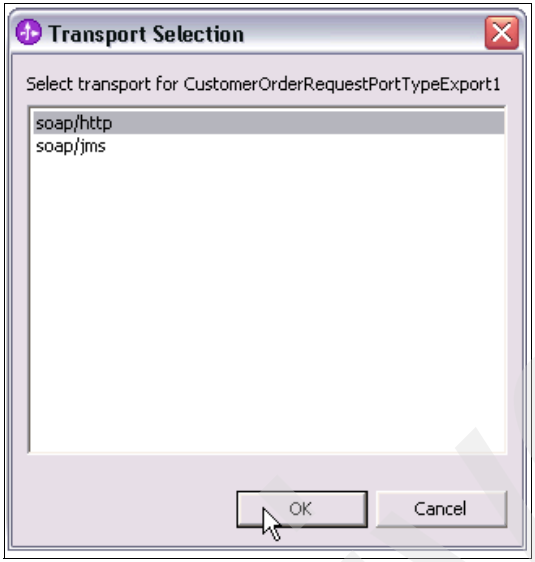

Figure 6-18 Selecting the transport type

> **Note:** You can easily move objects around in the canvas by selecting an object with the left mouse button, holding the left mouse button and dragging the object to another location in the canvas, and releasing the left mouse button.
>
> Move the just added Interface to the left of the OrderProcess Component, so that they line up nicely.

5. Move the objects until you can clearly see both of them.
6. Hover your mouse over **CustomerOrderRequestPortTypeExport1**. An orange handle appears at the right of the interface. This is the wire's source node. See Figure 6-19.

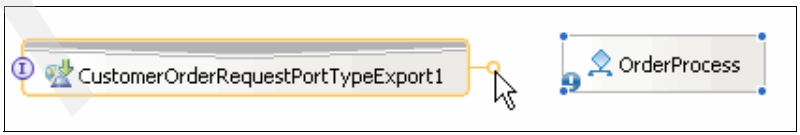

Figure 6-19 Orange handle

7. Drag the wire to the target node, **OrderProcess**, and release. The wire will turn blue temporarily (Figure 6-20).

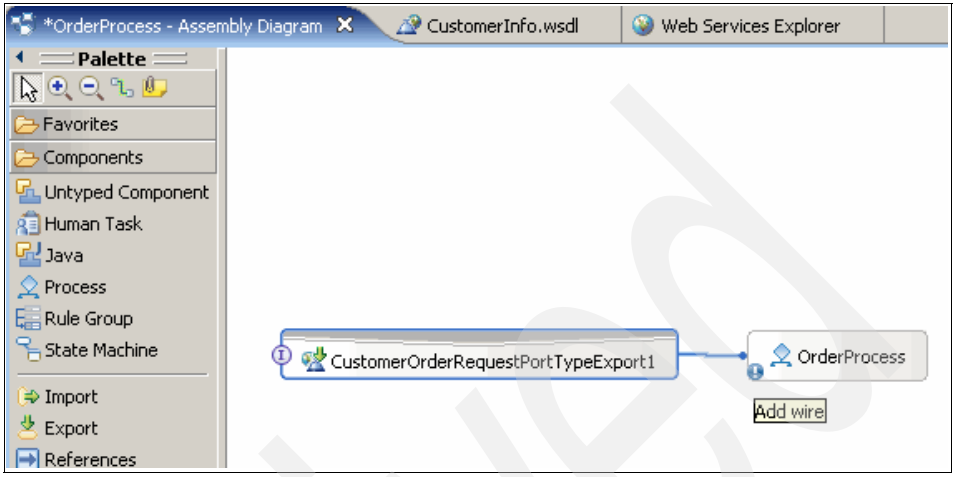

Figure 6-20 Wiring the nodes

8. Click **OK** in the Add Wire dialog box (Figure 6-21).

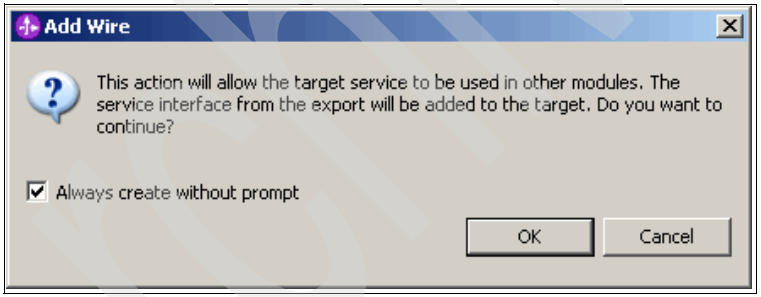

Figure 6-21 Confirmation window

After the wire is connected, it will turn light grey.

> **What you have so far:** You now have a place holder of a Component called **OrderProcess** (still empty) and an Interface for the request into that Process.

Next we add the Web services we need to call, and make the relevant settings. The first Web service we invoke is the **CustomerInfoPort** service, which will get the customer information from CICS.

9. Click **CustomerInfoPort** in the **Web Service Ports** folder and drag it to the canvas, approximately to the right of the **OrderProcess** component. Select **Import with Web Service Binding** in the pop-up window (Figure 6-22).

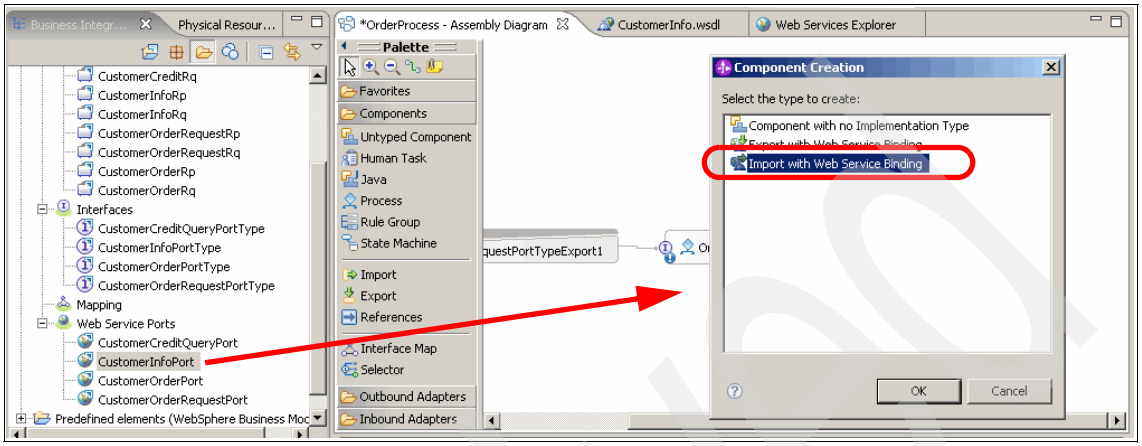

Figure 6-22 Importing the Customer Info service

10. In the Import Details pop-up window, click the radio button for **Do not specify a web service port at this time** (Figure 6-23).

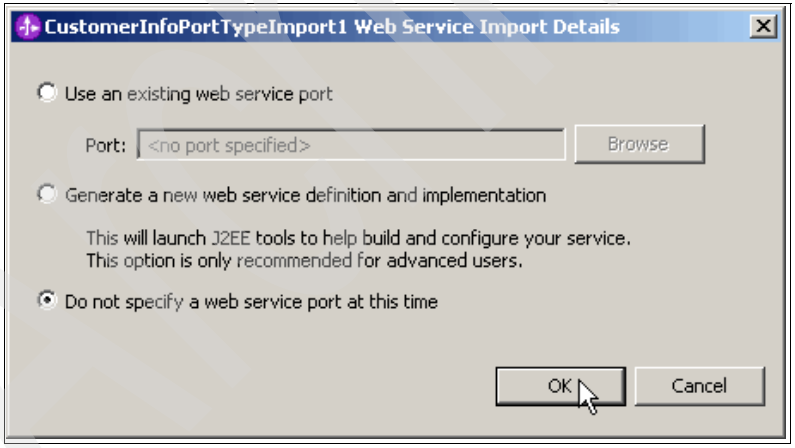

Figure 6-23 Import details

11. Add this service as a reference to the Process. Hover your mouse over the right side of **OrderProcess** block. An orange handle appears again. This is the wire's source node.

12. Drag the wire to the target node, **CustomerInfoPortTypeImport1**, and release. The wire will temporarily turn blue (Figure 6-24 on page 104).

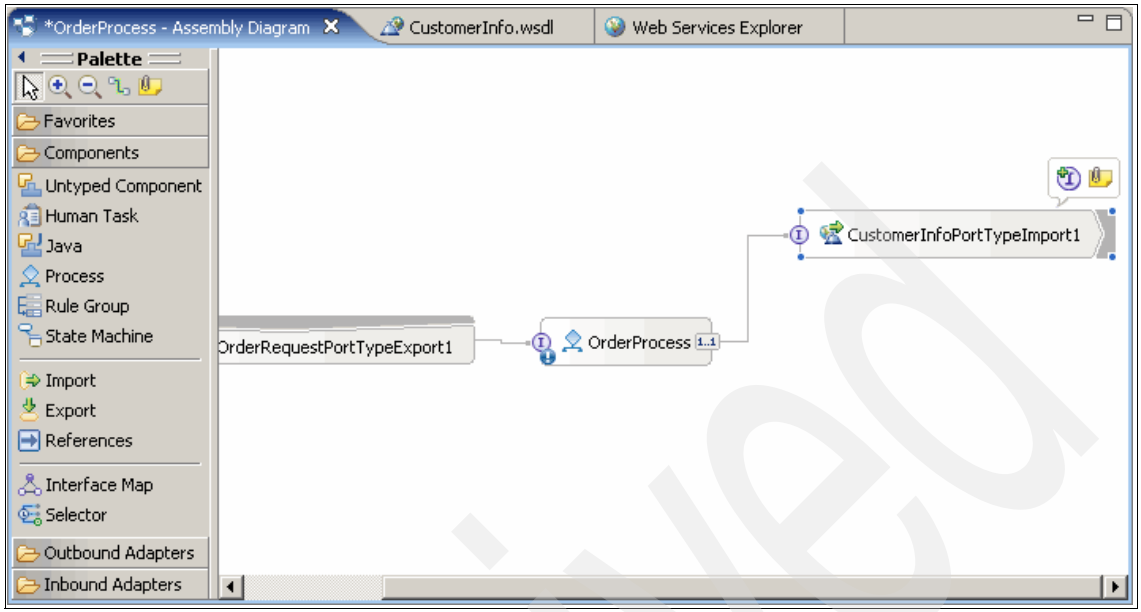

Figure 6-24 Wiring the Order Process and the Customer Info service

What we have so far: We now have an Interface for the request, a placeholder for the OrderProcess Component and a new Interface for the CustomerInfo service in CICS.

The next Web Service we will invoke is the credit check service, **CustomerCreditQueryService**.

13. Click **CustomerCreditQueryPort** in the **Web Service Ports** folder, drag it to the canvas approximately below the OrderProcess, and select **Import with Web Service Binding** (Figure 6-25 on page 105).

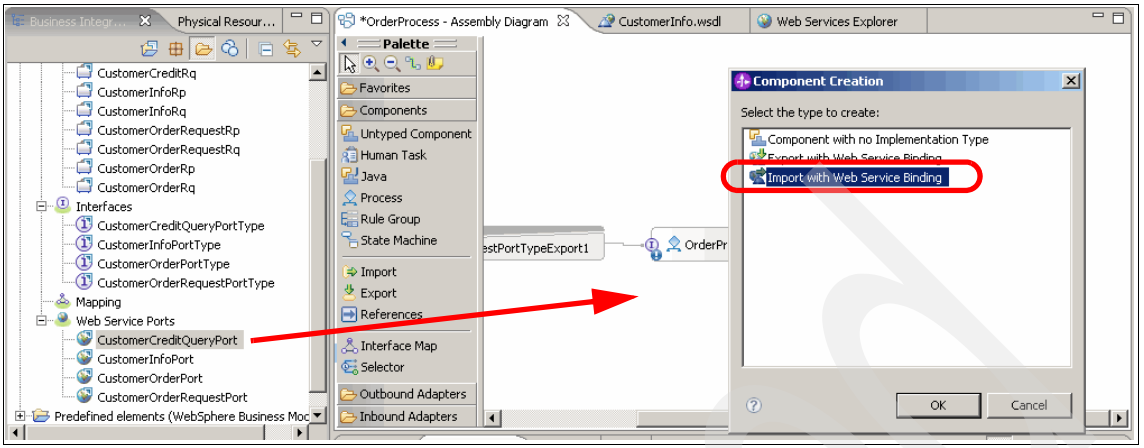

Figure 6-25 Importing the Customer Credit Check service

14. Wire the **OrderProcess** and the Credit check service together. Hover your mouse over the right side of the **OrderProcess** block, just below the existing reference connector. An orange handle appears. This is the wire's source node.

15. Drag the wire to the target node, **CustomerCreditQueryPortTypeImport1**, and release (Figure 6-26).

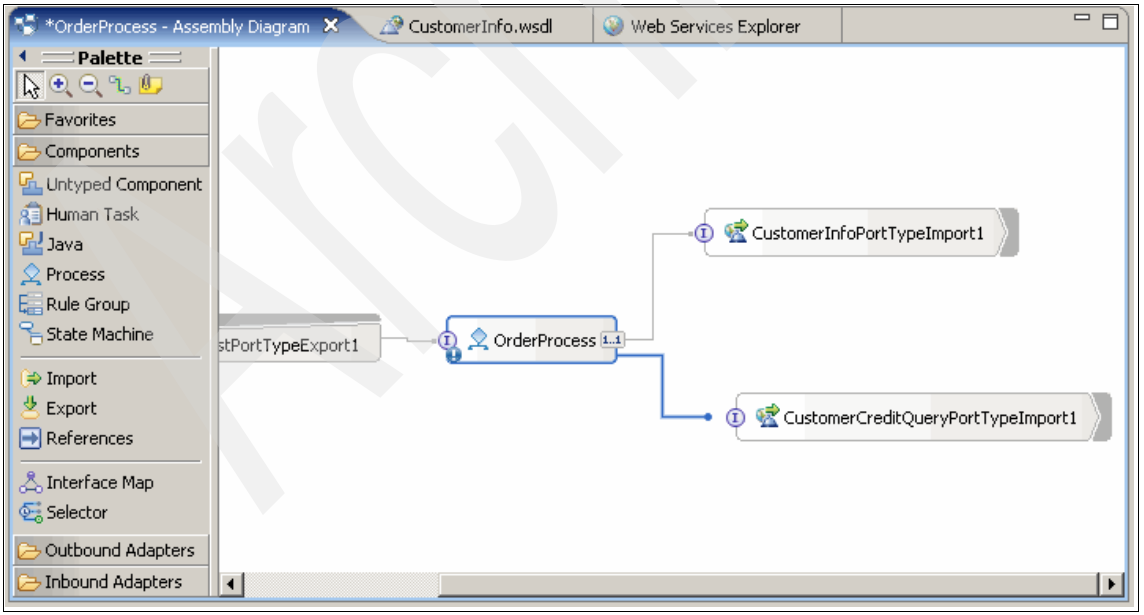

Figure 6-26 Wiring the Order Process and the Customer Credit Check service

What we have so far: An Interface for the request, a placeholder for the OrderProcess Component, an Interface to the CustomerInfo service in CICS, and a new Interface to the CustomerCreditQuery service in Broker.

16. The last Web service to invoke is the Customer order service. Click **CustomerOrderPort** in the **Web Service Ports** folder, drag it to the canvas approximately below the CustomerCreditQuery, and select **Import with Web Service binding** (Figure 6-27).

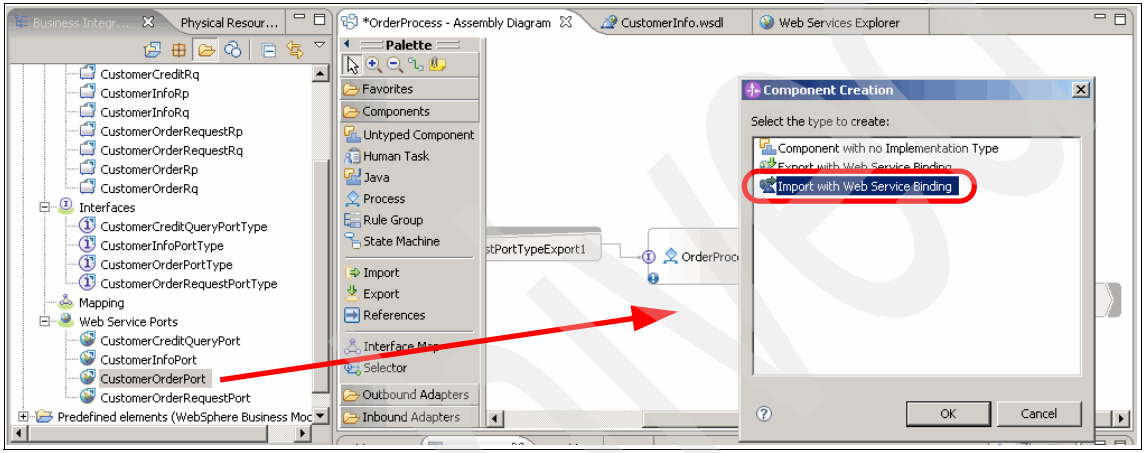

Figure 6-27 Importing the Customer Order service

17. Wire the **OrderProcess** and the reference together. Connect **OrderProcess** to **CustomerOrderTypeImport1**. See Figure 6-28 on page 107.

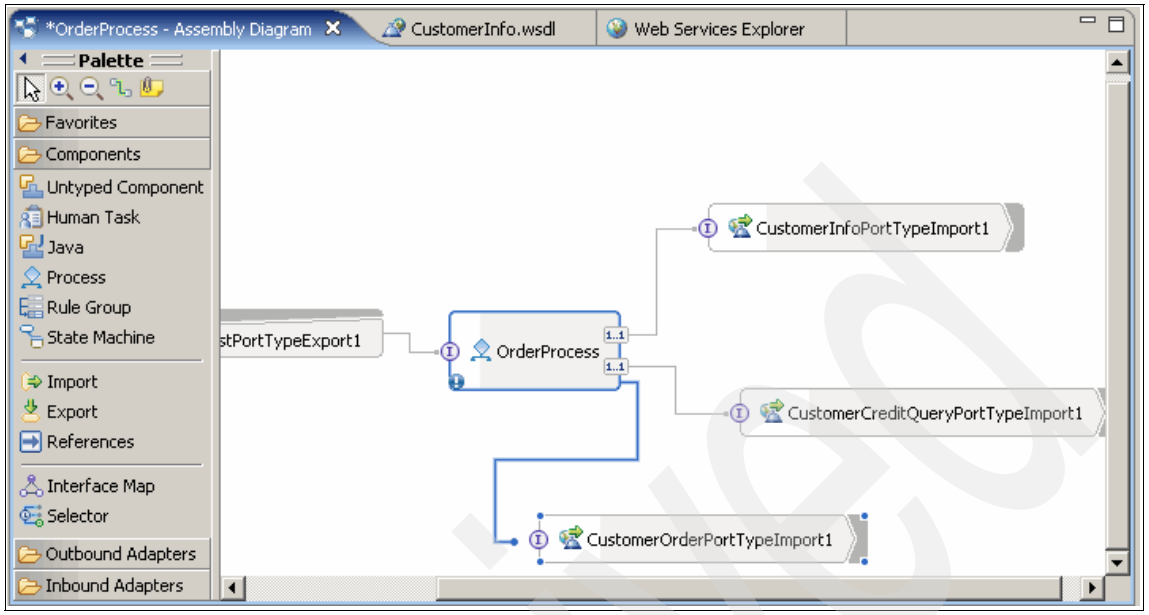

Figure 6-28 Wiring the Order Process and the Customer Order service

What we have so far: An Interface for the request, a placeholder for the OrderProcess Component, an Interface to the CustomerInfo service in CICS, an Interface to the CustomerCreditQuery service in Broker and an Interface to the CustomerOrder service in CICS.

18. You can arrange the canvas to get a better view. Right-click within the canvas and select **Layout Contents**, your Assembly Diagram should look like the one shown in Figure 6-29 on page 108.

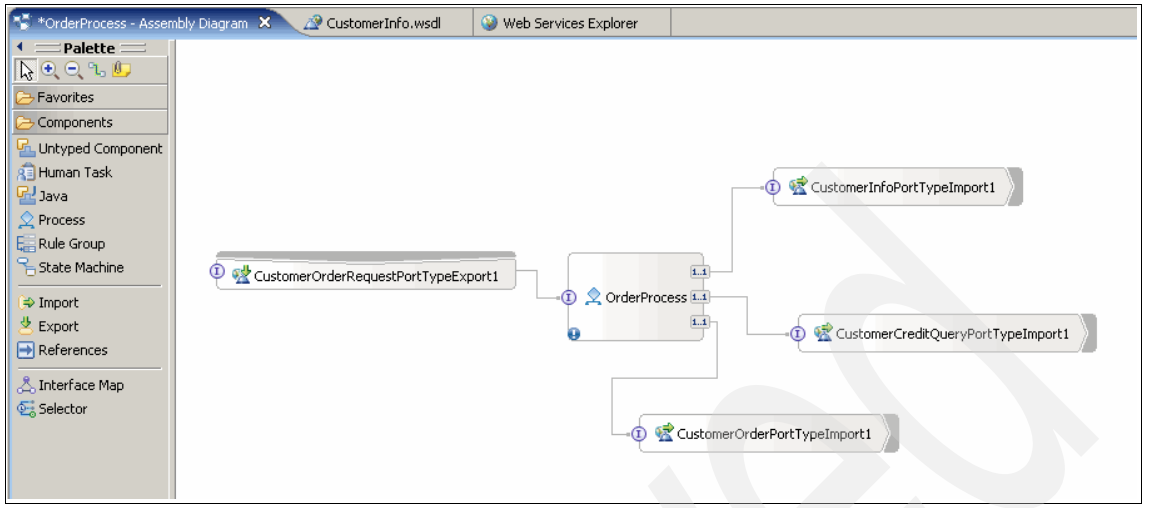

Figure 6-29 Assembly Diagram

6.4 Designing the process

Now we are ready to draw the flow of the OrderProcess process.

1. To start creating the flow, double-click **OrderProcess** in the assembly diagram and click **Yes** in the pop-up window shown in Figure 6-30.

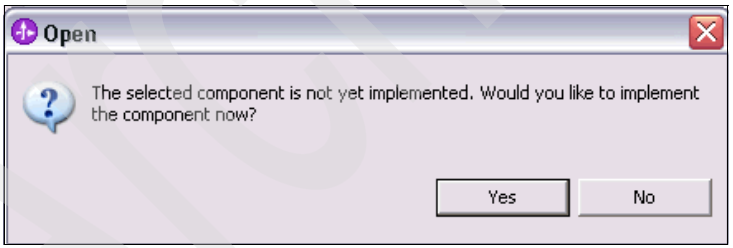

Figure 6-30 Confirmation window

2. Click **OK** in the Generate Implementation dialog shown in Figure 6-31 on page 109.

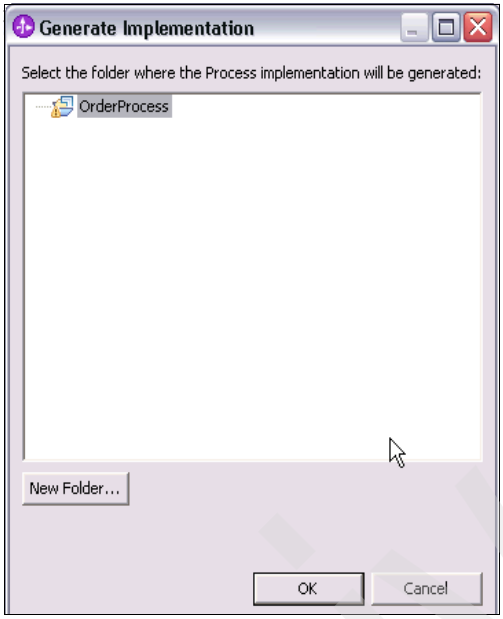

Figure 6-31 Selecting the folder

The Business Process editor opens, as shown in Figure 6-32.

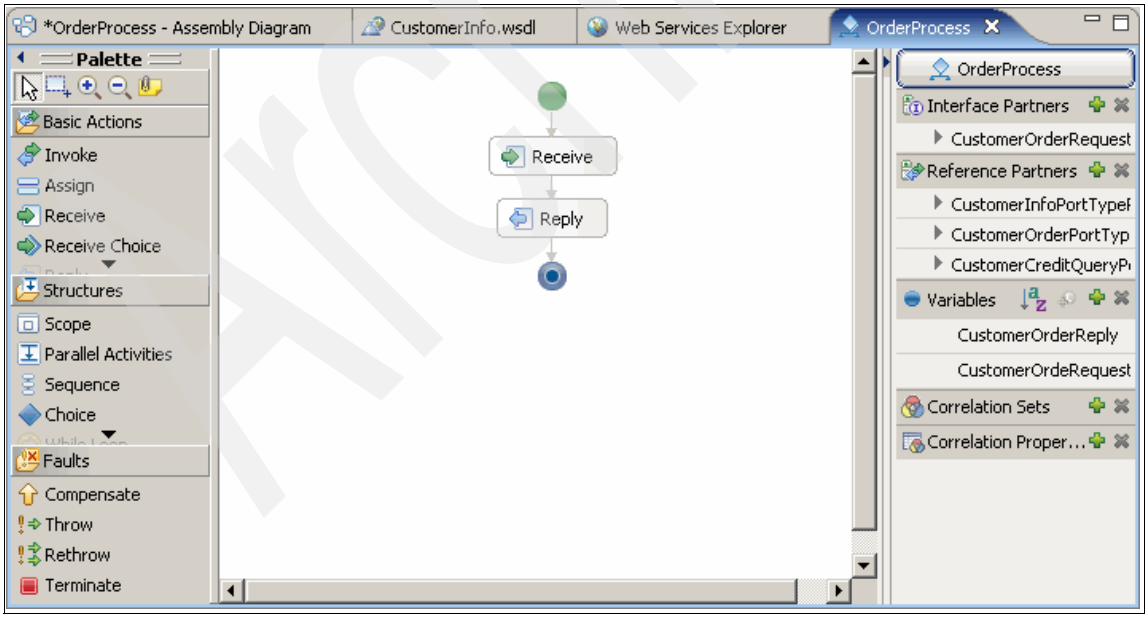

Figure 6-32 Business Process editor

This is called a BPEL Process. *BPEL* stands for *Business Process Execution Language*. In WID BPEL is generated using *activities*. Activities are available via the palette in the Business Process editor. When a new Business Process is created, the editor is opened and the Receive and Reply nodes are defined. Data is passed to the Receive node, and we decide what needs to be done with this data before it is send back via the Reply node. The final Business Process will look as shown in Figure 6-33.

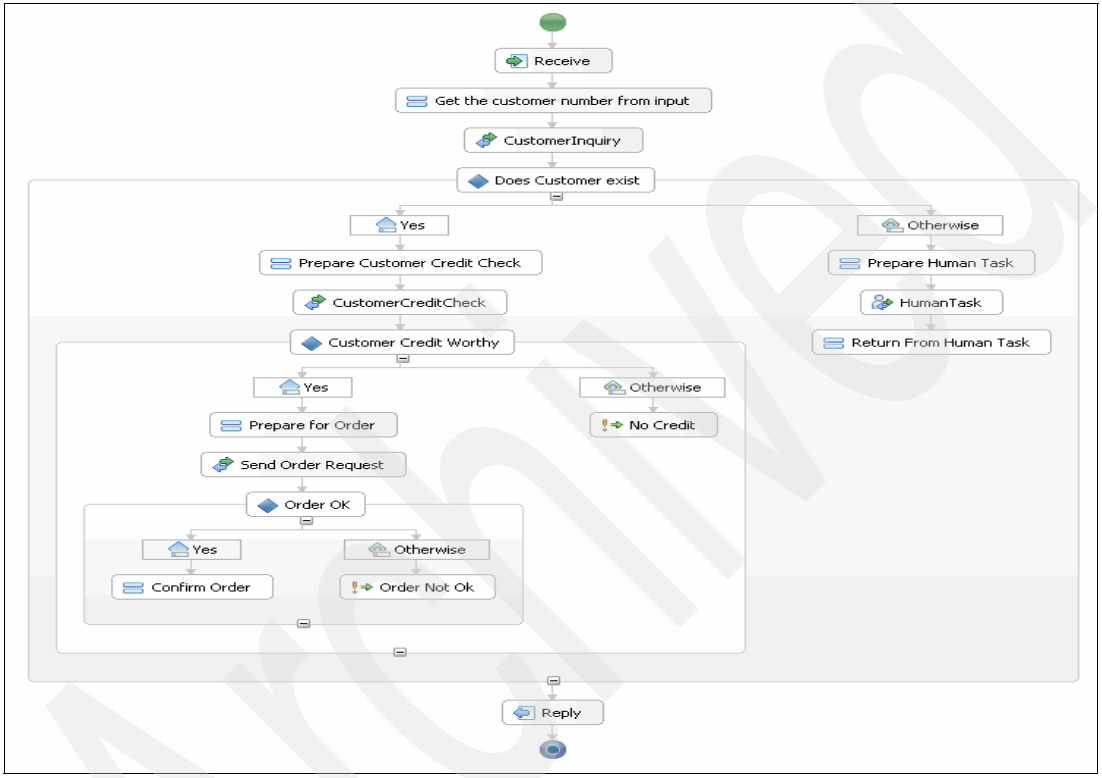

Figure 6-33 Final Order process

We now start creating this process. We need several variables to pass the data between the nodes. Assets used in this process are located in the right side of the Business Process editor. *Interface partners* represent the incoming data, *Reference partners* represent the services we interact with, and *Variables* contain the data being passed in the process.

3. Create a new variable. In the right side of the editor select **Variables** and click the green plus sign, as shown in Figure 6-34. The “Add a Variable” pane will open.

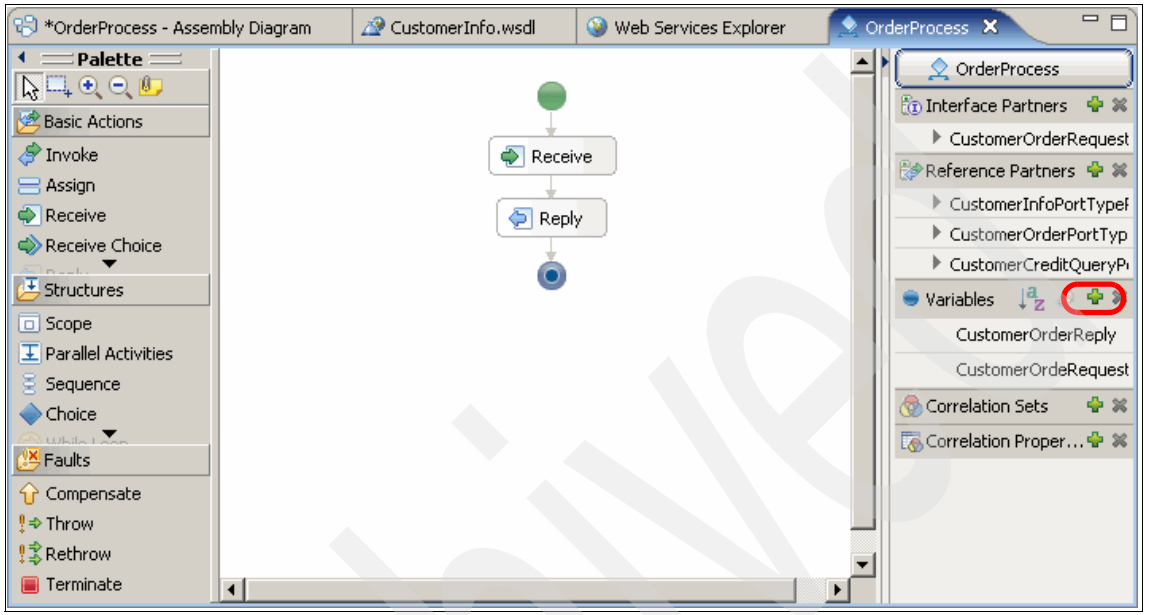

Figure 6-34 Creating a variable

4. In the Add a Variable pane, associate the new variable with a Data Type. Type in the variable name `CustomerCreditCheckRequest`**.** Specify the incoming Data Type. The Data Type was imported when the WSDL was imported, and is called **CustomerCreditRq** (Figure 6-35).

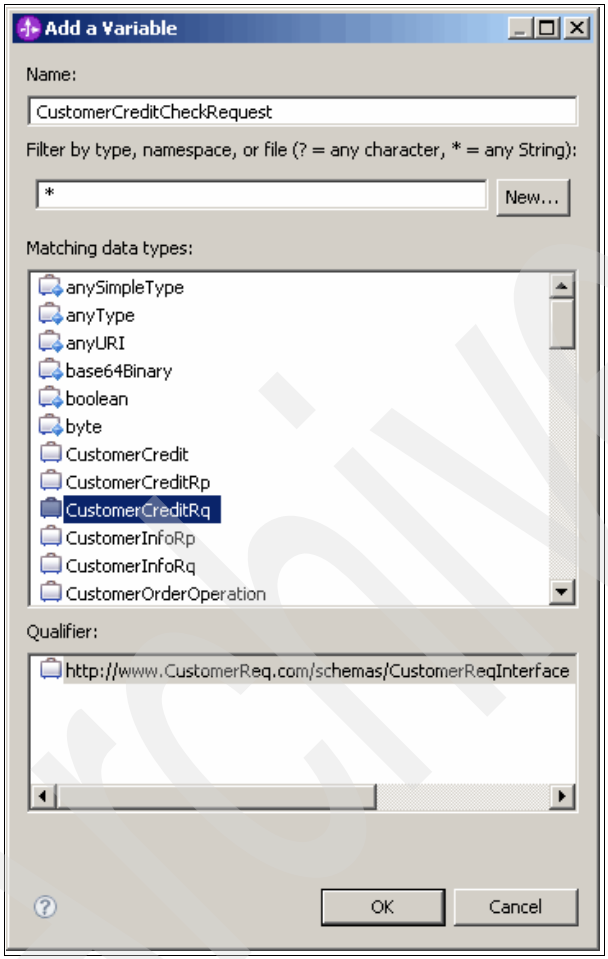

Figure 6-35 Creating a variable for the Customer Credit Check request

5. Define another variable and call it `CustomerCreditCheckReply`, use **CustomerCreditRp** as the data type and click **OK** (Figure 6-36).

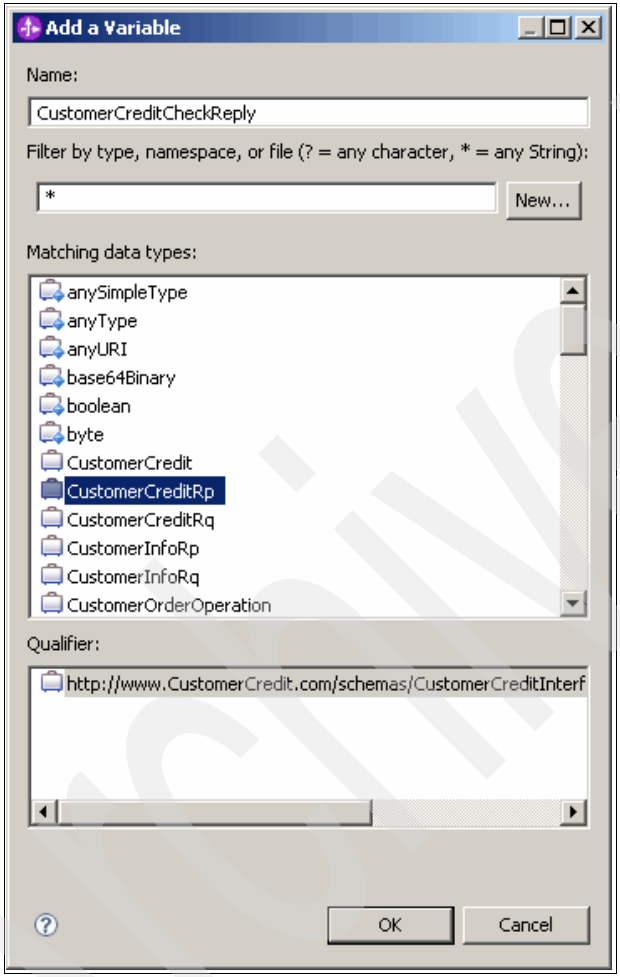

Figure 6-36 Creating a variable for the Customer Credit Check reply

6. Define the rest of the variables that will be used in this process. Create them as described in the previous steps and using Table 6-1 for reference.

Table 6-1 Other variables to be created

Name	Data Type
CustomerInfoRequest	CustomerInfoRq
CustomerInfoReply	CustomerInfoRp
CustomerOrderRequestRq	CustomerOrderRq
CustomerOrderRequestRp	CustomerOrderRp

7. To build the BPEL you need to add *nodes*. From the palette, select an **Assign** node from the **Basic Actions** folder and drop it *after* the **Receive node** (Figure 6-37 and Figure 6-38 on page 115).

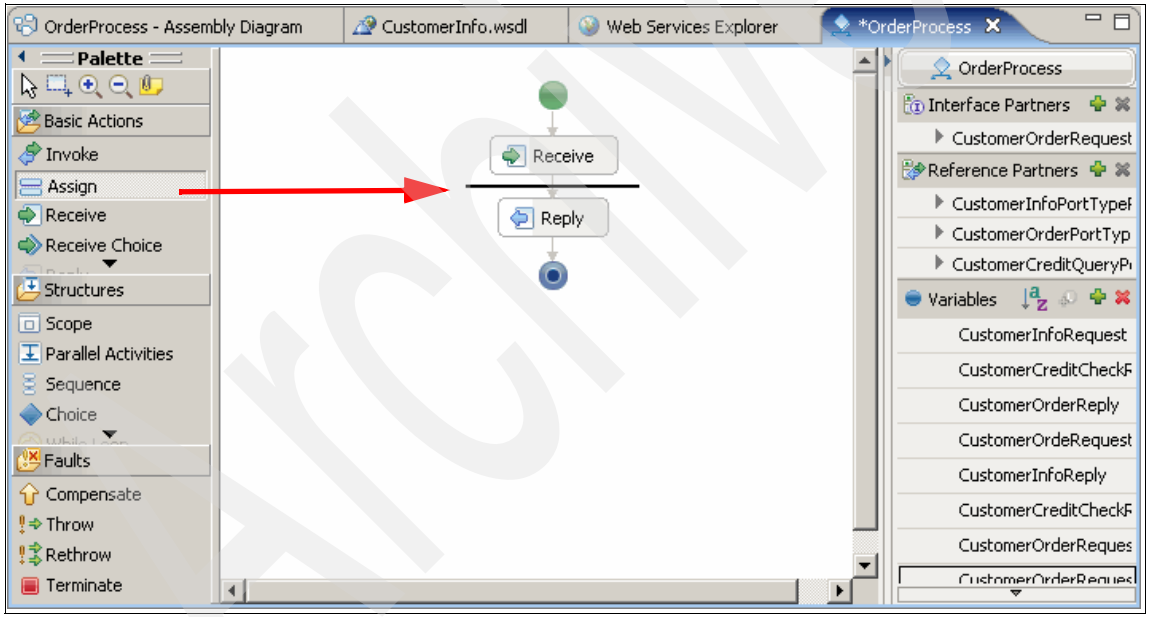

Figure 6-37 Selecting Assign from the palette

8. Change the node name to `Get the customer number from input` (Figure 6-38).

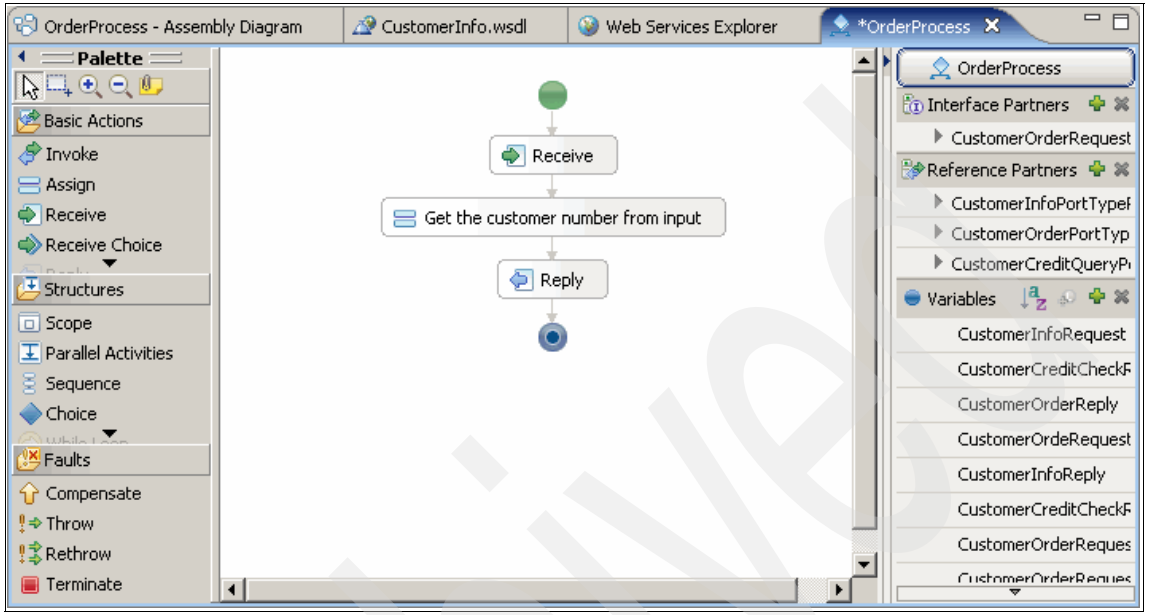

Figure 6-38 Changing the name of the Node

9. In the Assign From column of the **Details** view (located in the **Properties** tab at the bottom of the editor) select **Select From** → **CustomerOrdeRequest : CustomerOrderRequest** → **CustNo**, as shown in Figure 6-39.

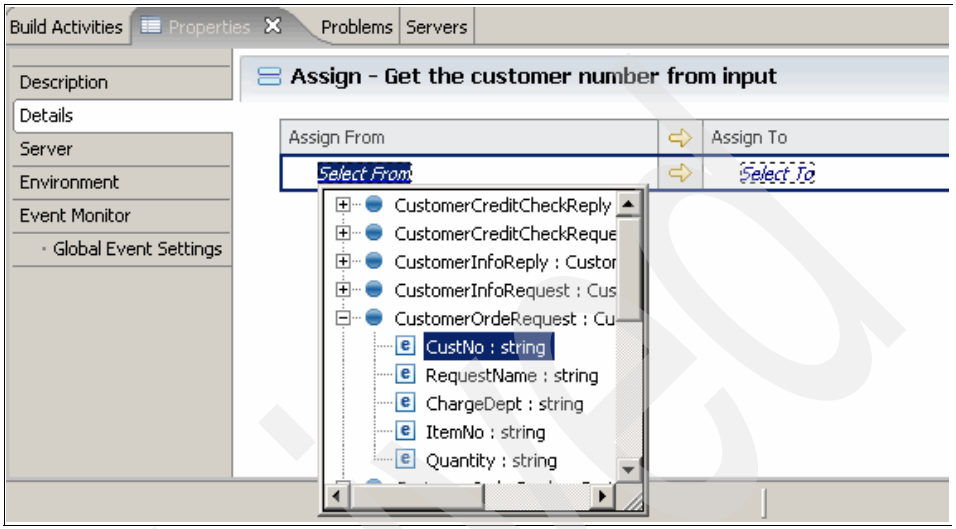

Figure 6-39 Assigning From variable

10. In the Assign To column, select **Select To** → **CustomerInfoRequest : CustomerInfoRq** → **comm_CustNo**, as shown in Figure 6-40.

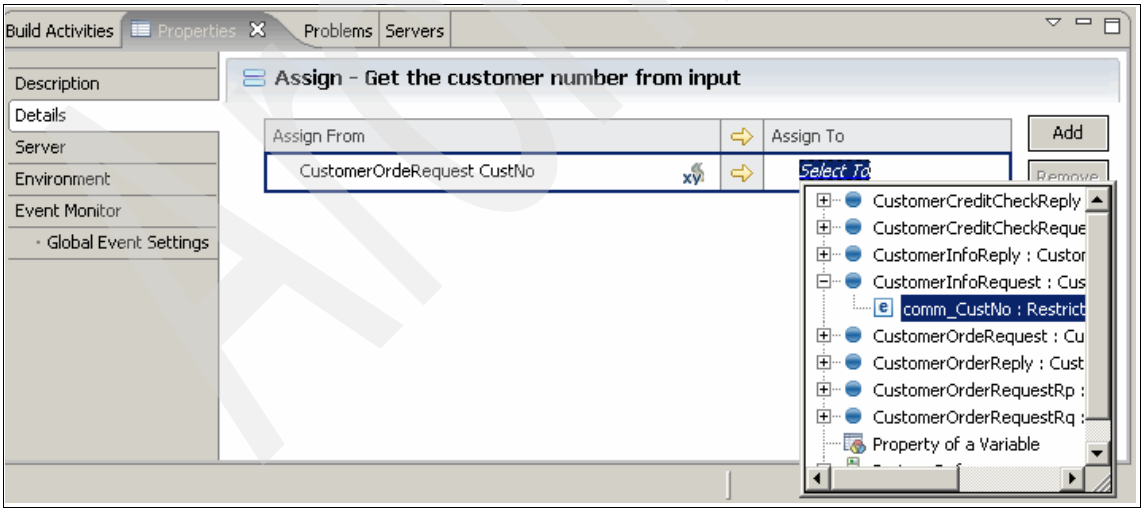

Figure 6-40 Assigning To variable

You now should have the assign statement shown in Figure 6-41.

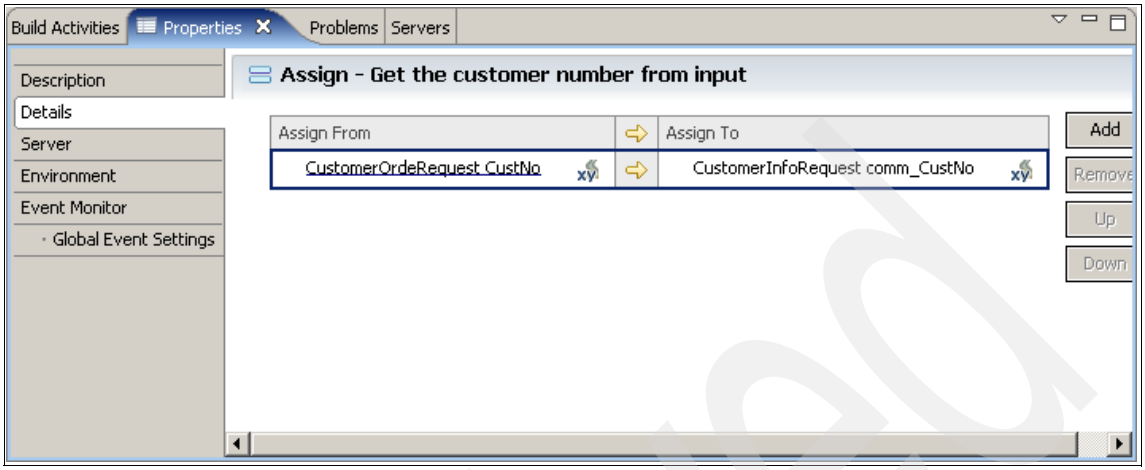

Figure 6-41 Assign statement

11. Select an **Invoke** node from the palette, and place it right after the Get customer Number from Input node, as shown in Figure 6-42.
12. Rename this node `CustomerInquiry`.

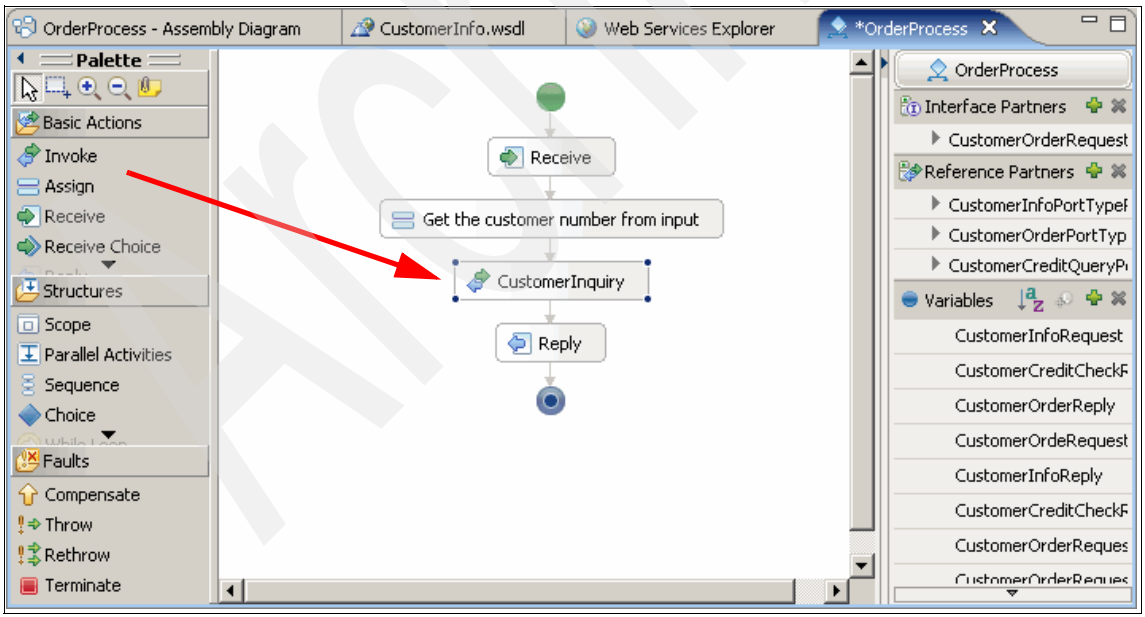

Figure 6-42 Changing the name of the Invoke Node

13. Go to the Details view of the **CustomerInquiry** Node, click **Browse**, and select **CustomerInfoPortTypePartner**, as shown in Figure 6-43. Click **OK**.

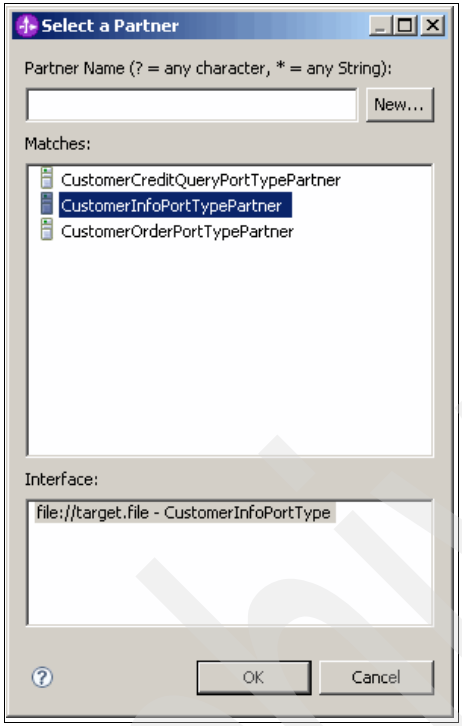

Figure 6-43 Selecting the Port of the Invoke Node

14. Still in the Details view, click **None** in the variable column for assigning both the input and output variables. For **Input** select **CustomerInfoRequest,** and for **Output** select **CustomerInfoReply**, as shown in Figure 6-44.

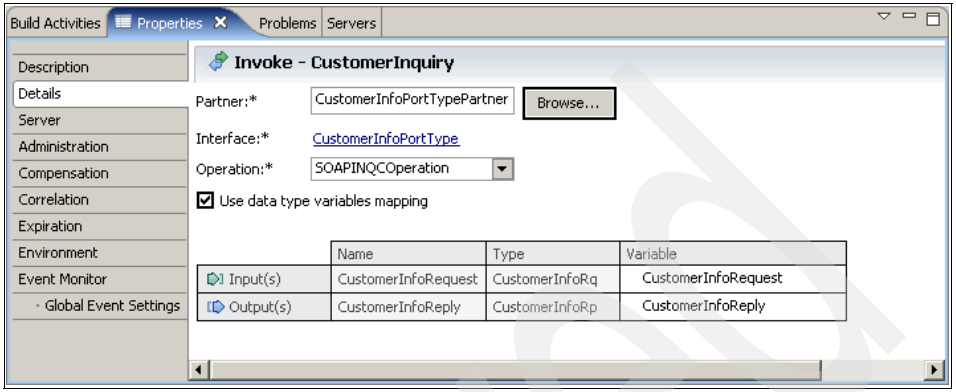

Figure 6-44 Assigning the variables

15. Add a **Choice** node by selecting **Choice** from the palette under the **Structures** folder. Place the node after the CustomerInquiry Node and rename it to `Does Customer exist`, as shown in Figure 6-45.

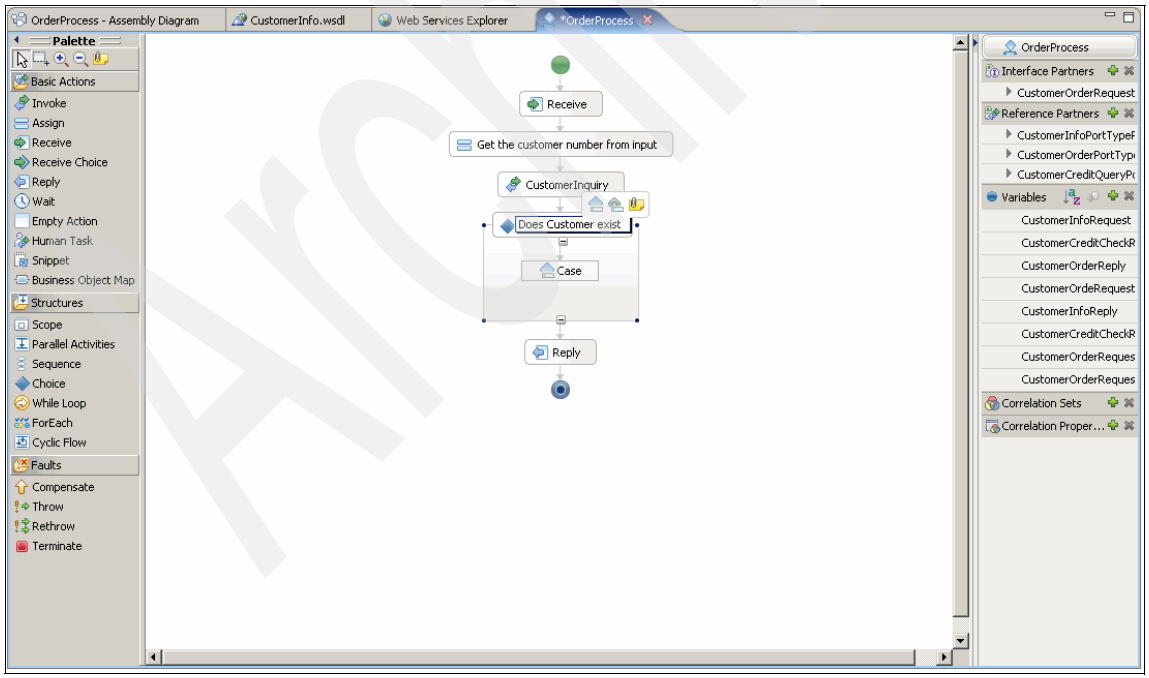

Figure 6-45 Placing and renaming the Choice Node

16. Select the **Case** node and change the name to `Yes` in the **Description** field in the Properties view, as shown in Figure 6-46.

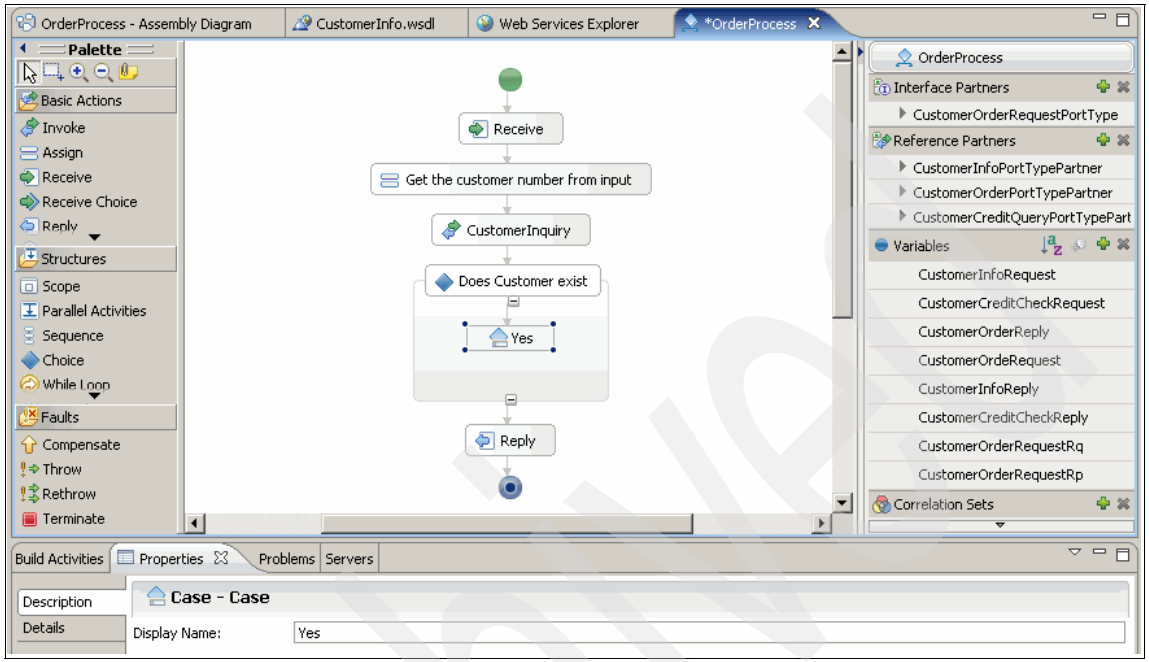

Figure 6-46 Change display name of Case Node

17. In the Details view change **Expression® language** to **Same as Process(Java)**.

18. Select **True** and expand **CustomerInfoReply** → **comm_RetCode : string**, as shown in Figure 6-47.

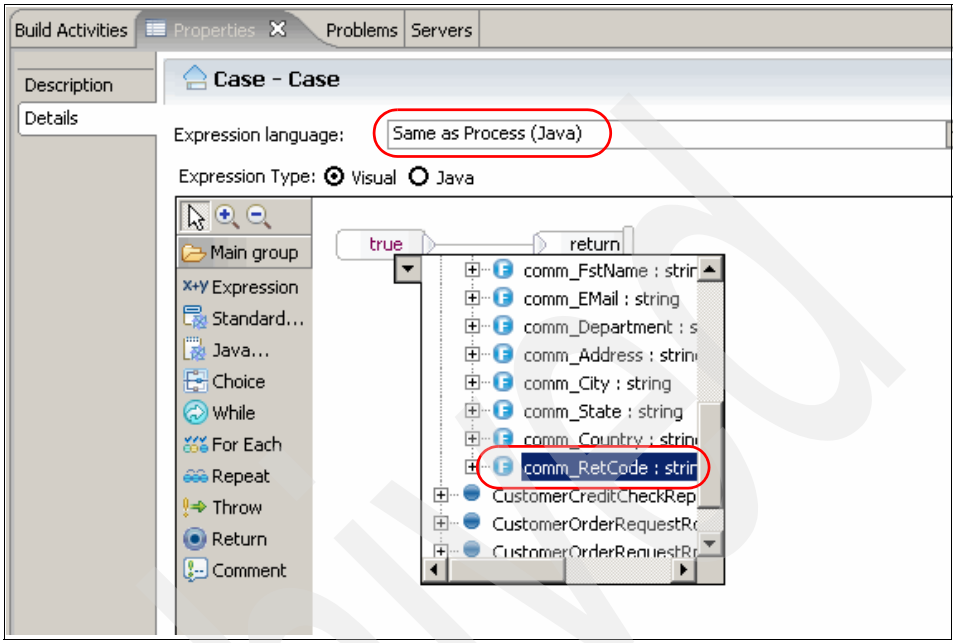

Figure 6-47 Selecting Return code

19. Select **==**, as shown in Figure 6-48.

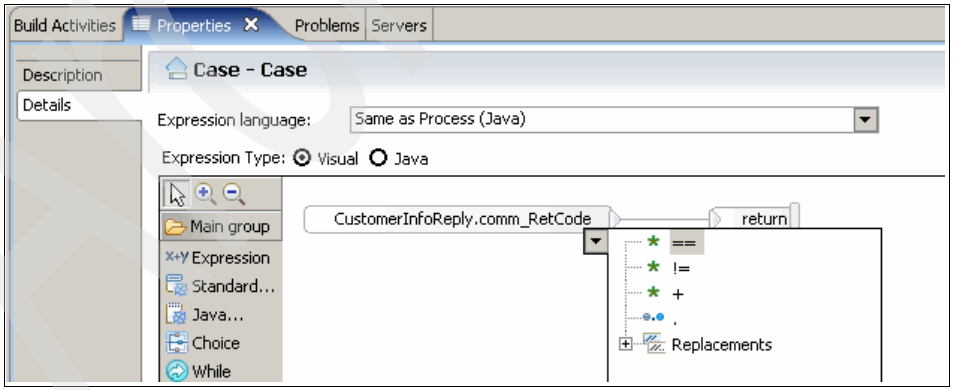

Figure 6-48 Selecting the operator

20. Select **String**, type the value 000**,** and click **Enter**, as shown in Figure 6-49.

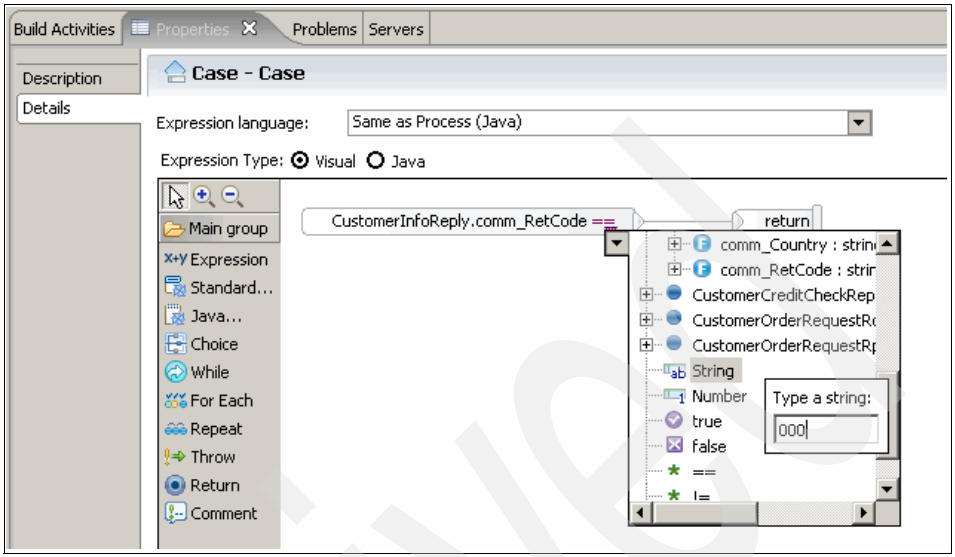

Figure 6-49 Selecting Type and specifying value

The final expression should look as shown in Figure 6-50.

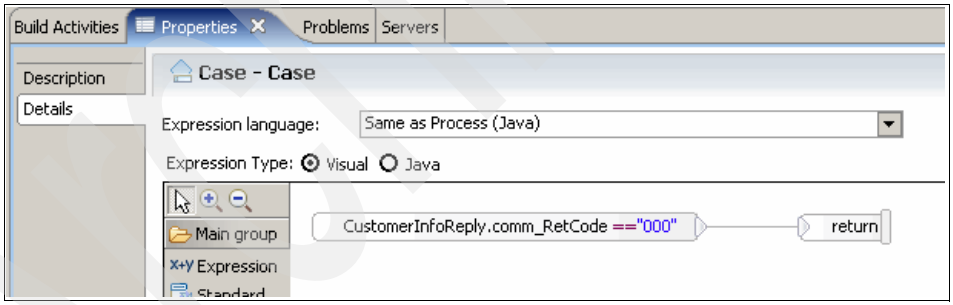

Figure 6-50 Final expression for Yes

21. If the customer is not found, we need to send a message or a fault. Add an *Otherwise* statement to the Choice node. Right-click **Does Customer Exist** and select **Add an Otherwise Element**, as shown in Figure 6-51.

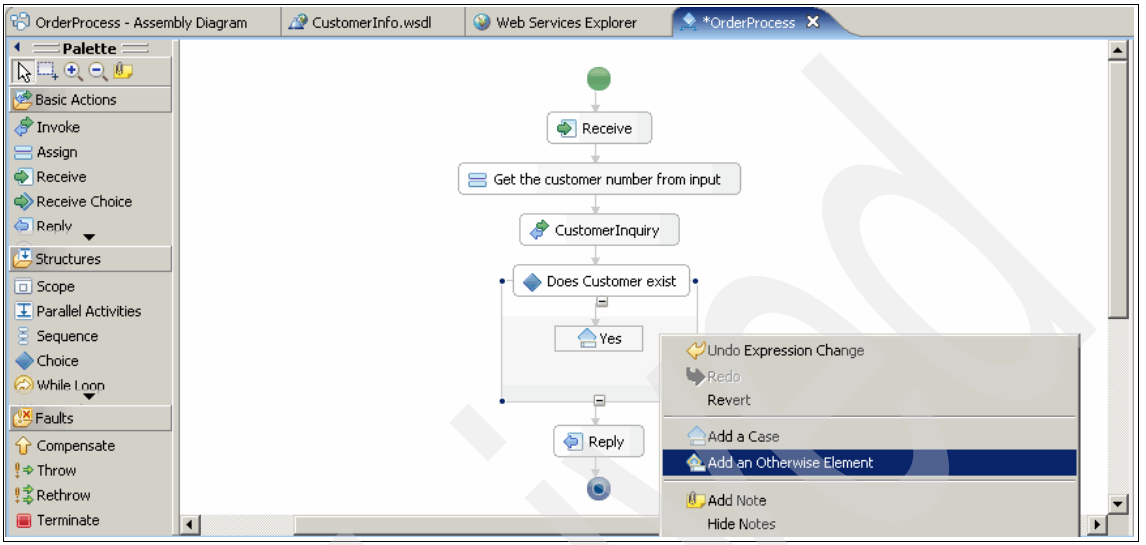

Figure 6-51 Add an Otherwise element

22. Select a **Throw** node from the palette under the Faults folder.
23. Place the node right after the **Otherwise** node, and rename it `Customer not Found`.
24. In the Properties view for the **Customer Not Found**, select **Details** and make sure the Fault Name is **CustomerNotFound**.

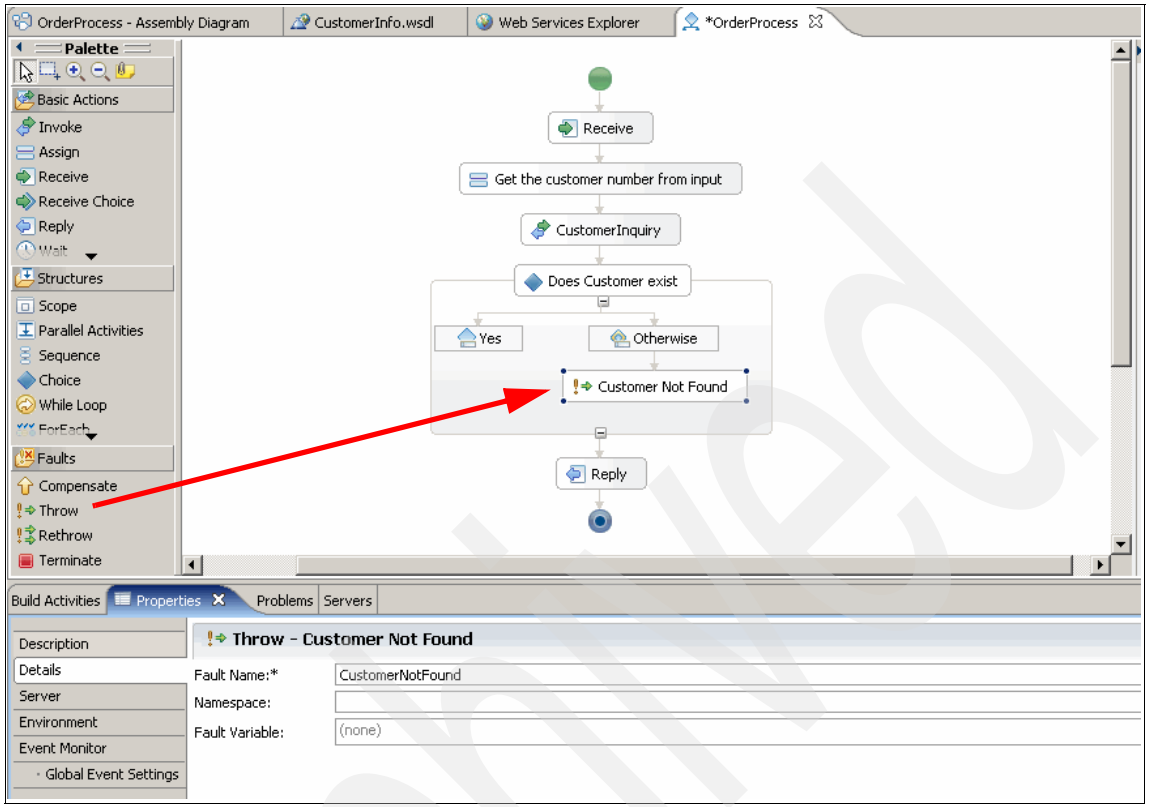

Figure 6-52 Select a Throw node from the palette

25. If the customer is found, we do a credit check verification. Select an **Assign** node from the palette, and place it right after the Yes node. Rename it `Prepare Customer Credit Check`.

26. In the Properties view select **Details** and make the following variable assignment:

   – Assign From variable: **CustomerInfoReply : CustomerInfoRp → comm_CustNo**

   – Assign To variable: **CustomerCreditCheckRequest : CustomerCreditRq → CustNo**

   See Figure 6-53 on page 125.

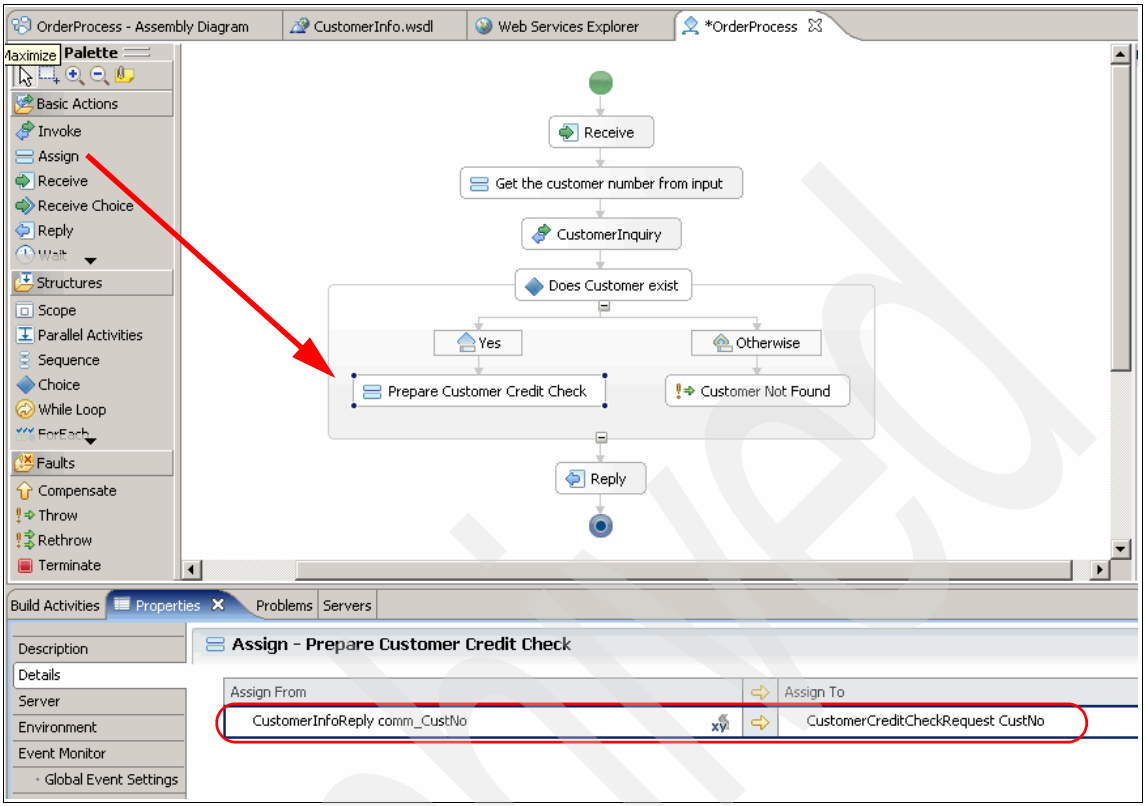

Figure 6-53 Adding an Assign node for the Credit Check

27. Select an **Invoke** node from the palette and drop it after the **Prepare Customer Credit Check** node. Rename it `CustomerCreditCheck`.
28. In the Properties view select **Details** and click **Browse** to find the port. Select **CustomerCreditQueryPortTypePartner**.
    - To set the input variable, click **None** in the variable column and select **CustomerCreditCheckRequest**.
    - To set the output variable, click **None** in the variable column and select **CustomerCreditCheckReply**.

    The details should now look as shown in Figure 6-54 on page 126.

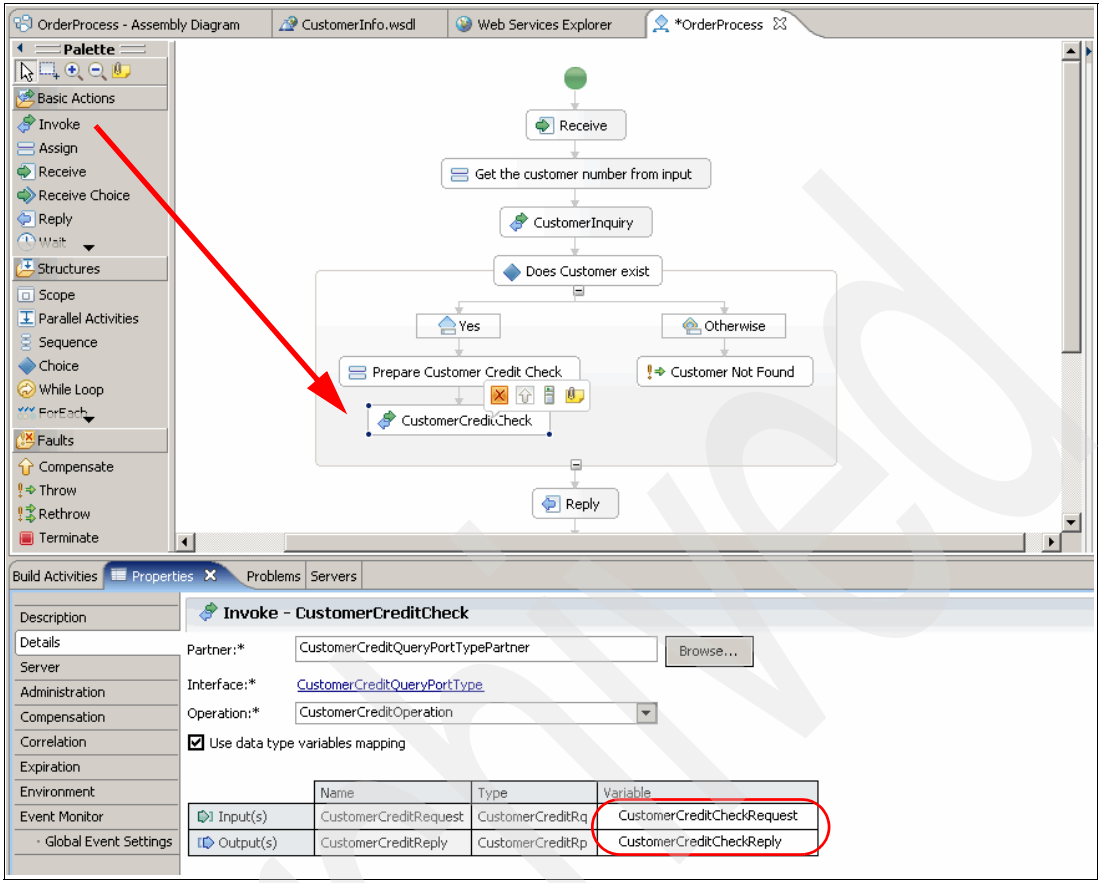

Figure 6-54 Add an Invoke node for the Customer Credit Check

29. After the Credit Check we add a **Choice** node from the palette, and rename it `Customer Credit Worthy`, as shown in Figure 6-55 on page 127.

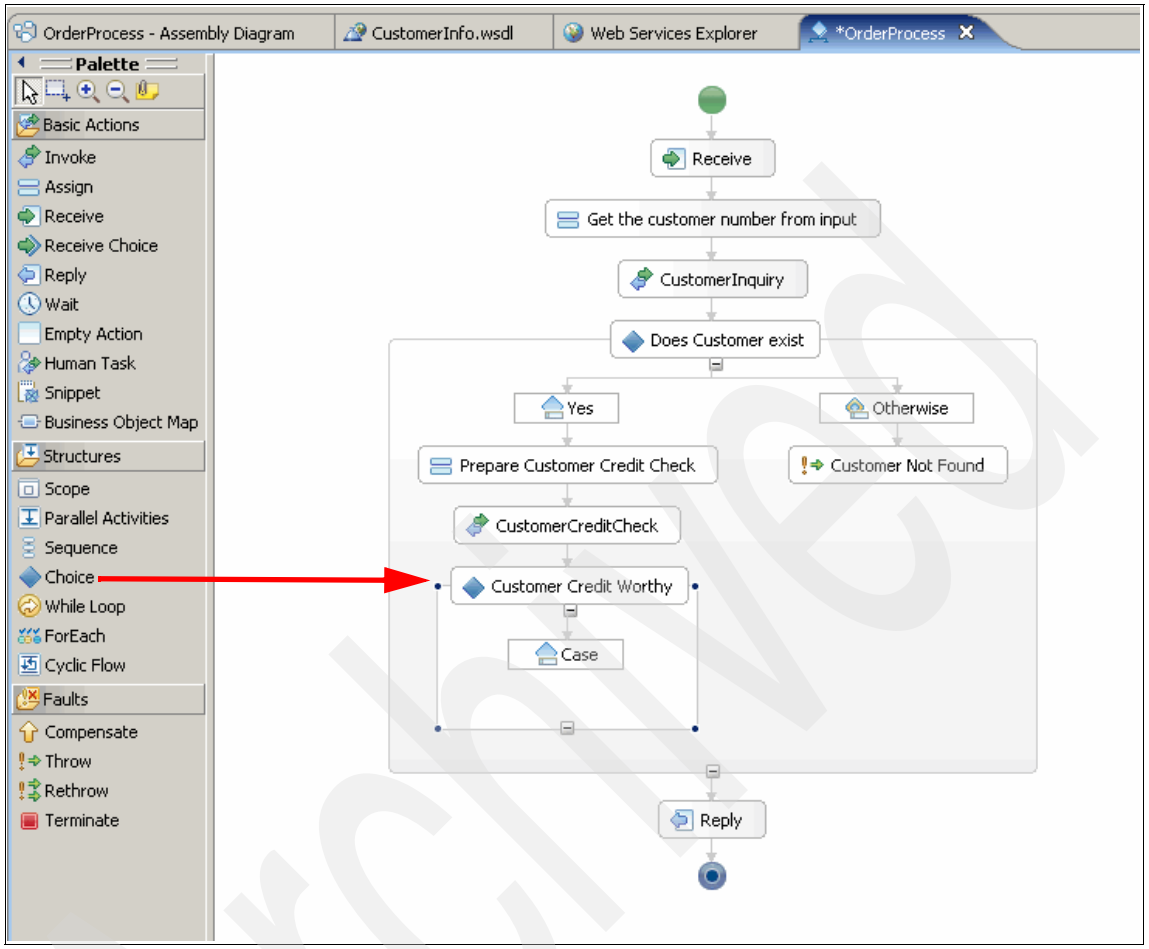

Figure 6-55 Adding a Choice node to the flow and renaming it

30. Change the Case to `Yes`, similar to what we did earlier. In the Properties View select **Details** and select **Expression Language: Same as Process(Java)** and **Expression Type: Java**, as shown in Figure 6-56 on page 128.

    Because you have selected to provide the expression by means of Java code, you will be guided to an editor. Before you get there, click **Yes** in the pop-up Question box.

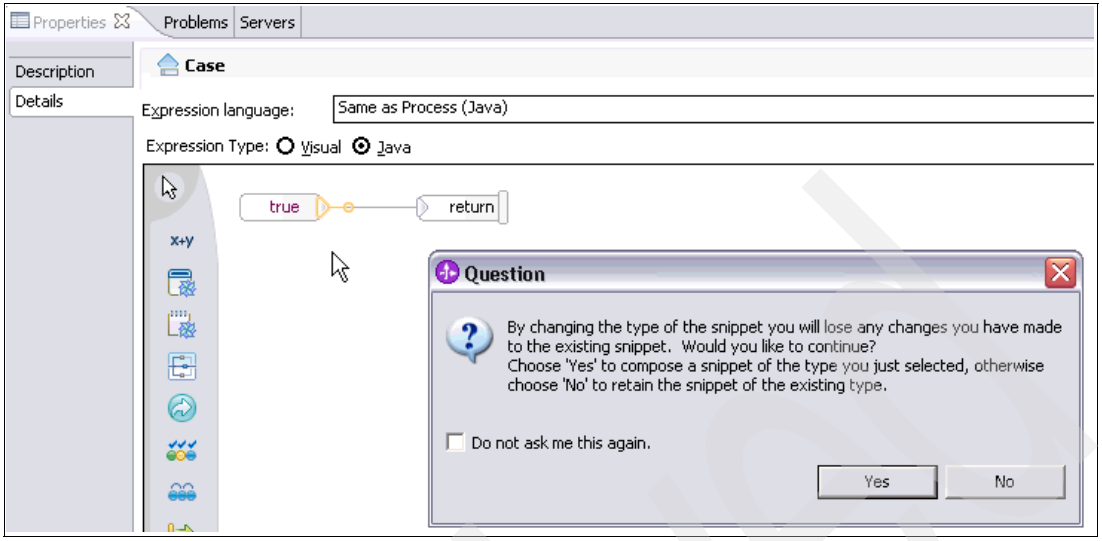

Figure 6-56 Case properties

31. In the editor area replace *all* default code with the following Java code:

```
// get the today's date
java.util.Date todaysDate =
java.util.Calendar.getInstance().getTime();
//get the date from the reply of when the credit expires
java.util.Date creditExpireDate =
CustomerCreditCheckReply.getDate("CreditExpireDate");
// compare the dates and make sure that today is not after the
credit has expired
// if the today is before the credit expiry date then return true
// otherwise return false
boolean __result__1 = false;
if (todaysDate.before(creditExpireDate)) { __result__1 = true;};
return __result__1;
```

32. Save the code and verify that there are no errors in Java code. Note that a red cross in the Yes block is OK for now.
33. Add an **Otherwise** element after the **CustomerCreditWorthy** choice node.
34. After the Otherwise element add a **Throw Fault** node from the palette, and rename it `No Credit`. In the Properties view select **Details** and set the Fault Type to **User Defined** and the Fault Name to `NoCredit`. See Figure 6-57 on page 129.

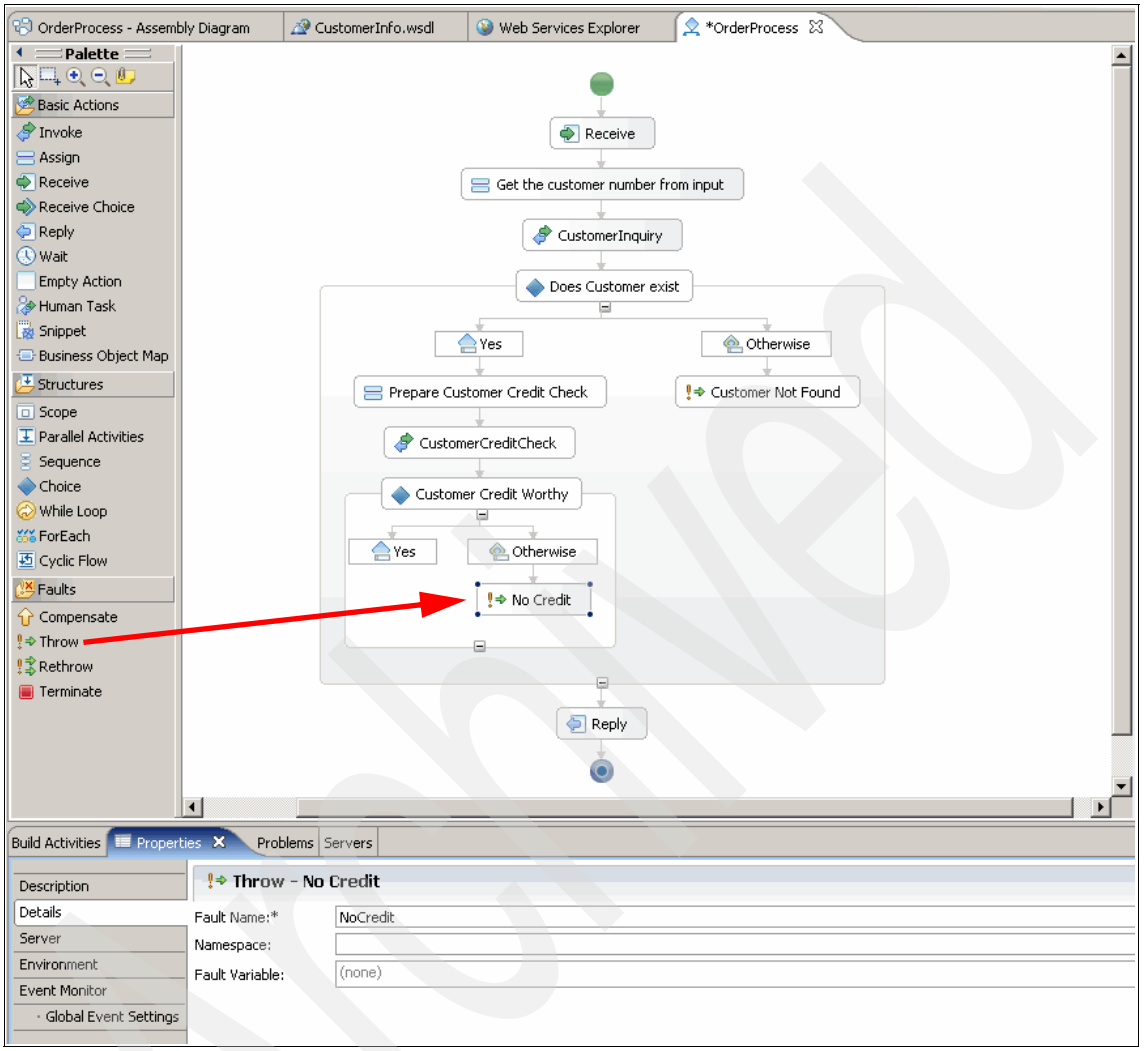

Figure 6-57 Adding a Throw Fault node

35. Add an **Assign** node from the palette after the Yes choice. Rename it `Prepare For Order`, as shown in Figure 6-58.

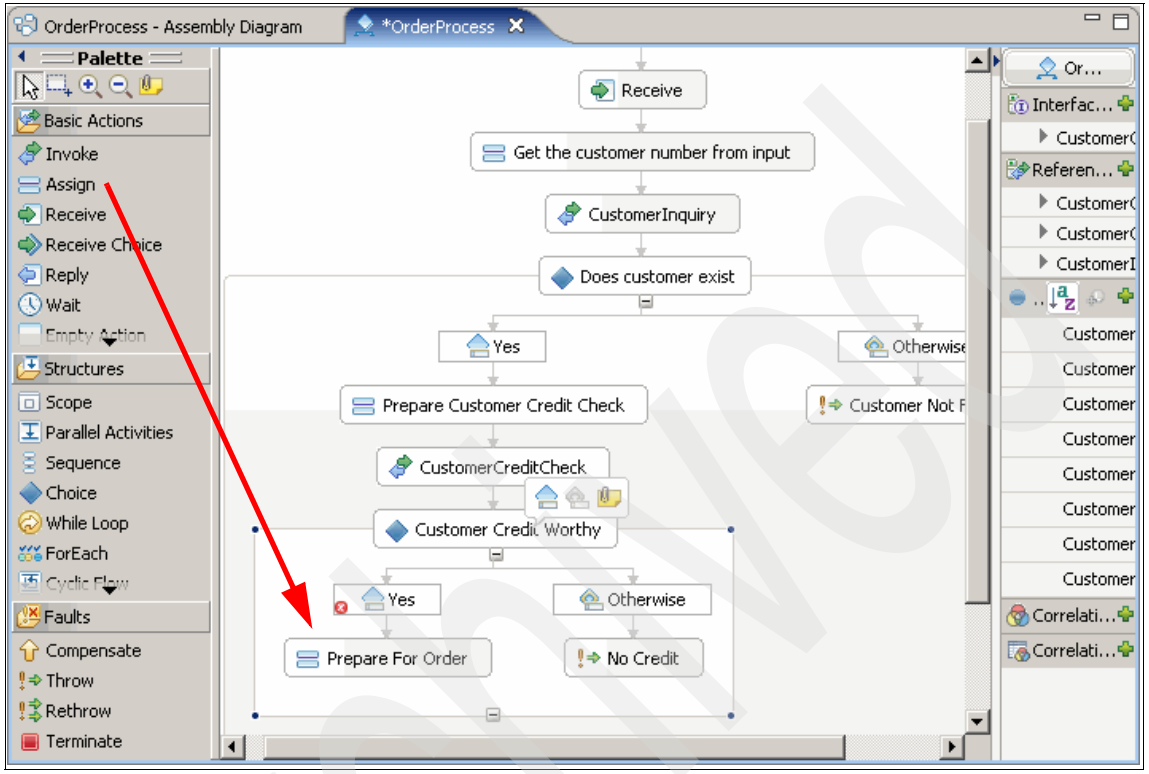

Figure 6-58 Adding an Assign Node and renaming it

36. Now you need to move all the variables needed for the order to be able to run the Place Order service. The variables are shown in Table 6-2 on page 131. Make sure the **Prepare for Order** node is selected before proceeding.

    Click the **Select From** and **Select To** fields to start assigning the first variable. Once you have mapped the first variable, click the **Add** button to create the next one. At the end you need to have five mappings. Take a close look at the screen shot!

Table 6-2 Variable mappings

From	To
CustomerOrdeRequest:CustomerOrderRequestRq >CustNo	CustomerOrderRequestRq:CustomerOrderRq > service_input_area > input_data_container >...>custno
CustomerOrdeRequest:CustomerOrderRequestRq > RequestName	CustomerOrderRequestRq:CustomerOrderRq > service_input_area > dfhmahr_requestv1 > dfhmahr_requestname
CustomerOrdeRequest:CustomerOrderRequestRq > ChargeDept	CustomerOrderRequestRq:CustomerOrderRq > service_input_area > input_data_container > ... > chargedept
CustomerOrdeRequest:CustomerOrderRequestRq > ItemNo	CustomerOrderRequestRq:CustomerOrderRq > service_input_area > input_data_container> ... > itemno
CustomerOrdeRequest:CustomerOrderRequestRq > Quantity	CustomerOrderRequestRq:CustomerOrderRq > service_input_area > input_data_container > ... > quantity

Figure 6-59 shows the variable mappings.

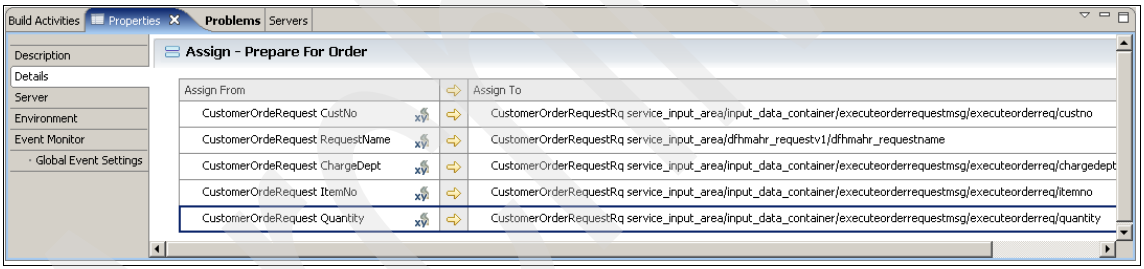

Figure 6-59 Assigned variables

37. Select an **Invoke** node from the palette, place it after the **Prepare For Order** node, and rename it `Send Order Request`. See Figure 6-60 on page 132.
38. In the Properties view, select **Details**, then browse for the Port Type to be used. Select **CustomerOrderPortTypePartner**.
39. Set the input variable to **CustomerOrderRequestRq**.
40. Set the output variable to **CustomerOrderRequestRp**.

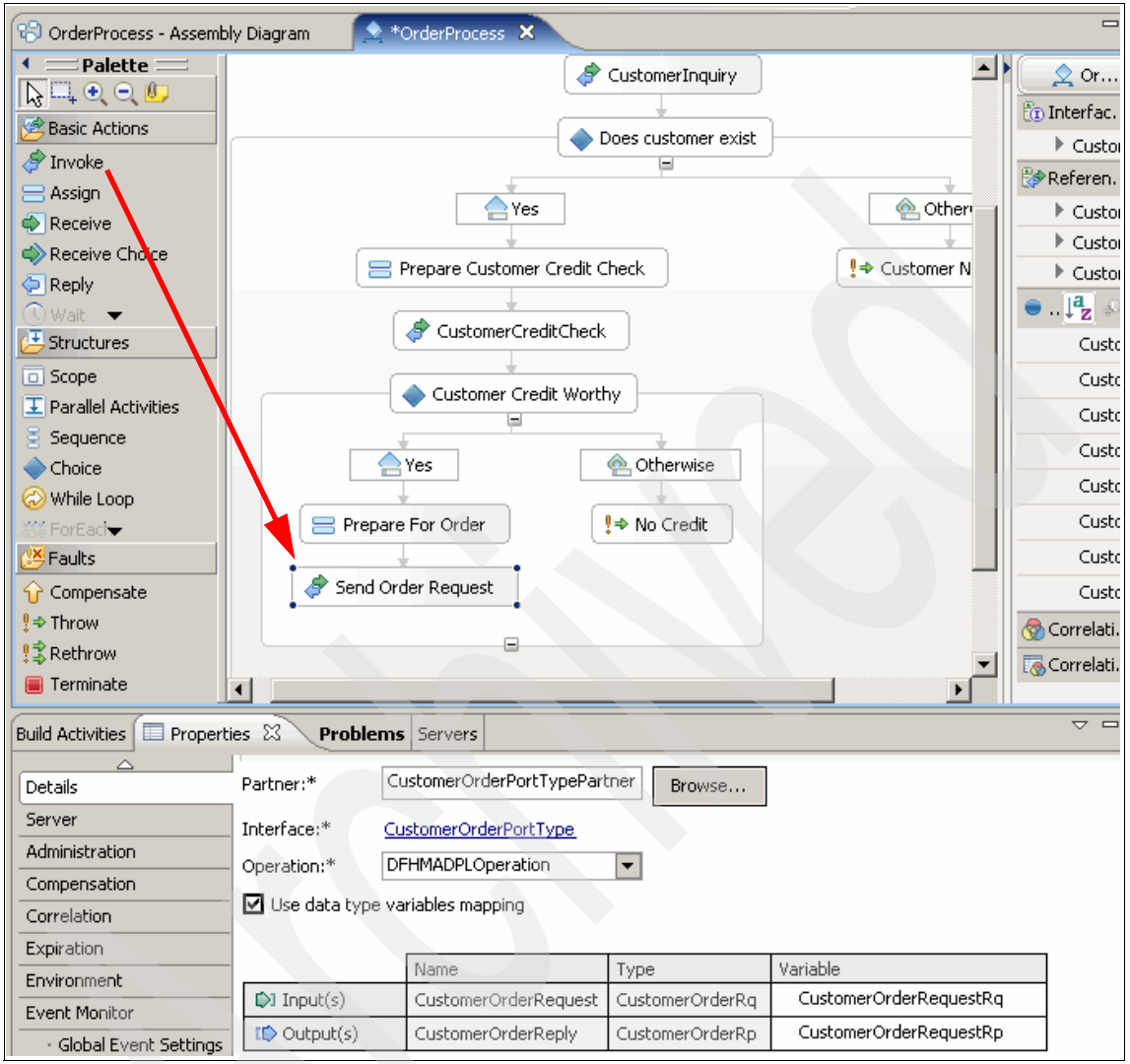

Figure 6-60 Adding an Invoke node for sending the request

41. Select a **Choice** node from the palette and add it after the **Send Order Request**. Rename it `Order OK` (Figure 6-61).

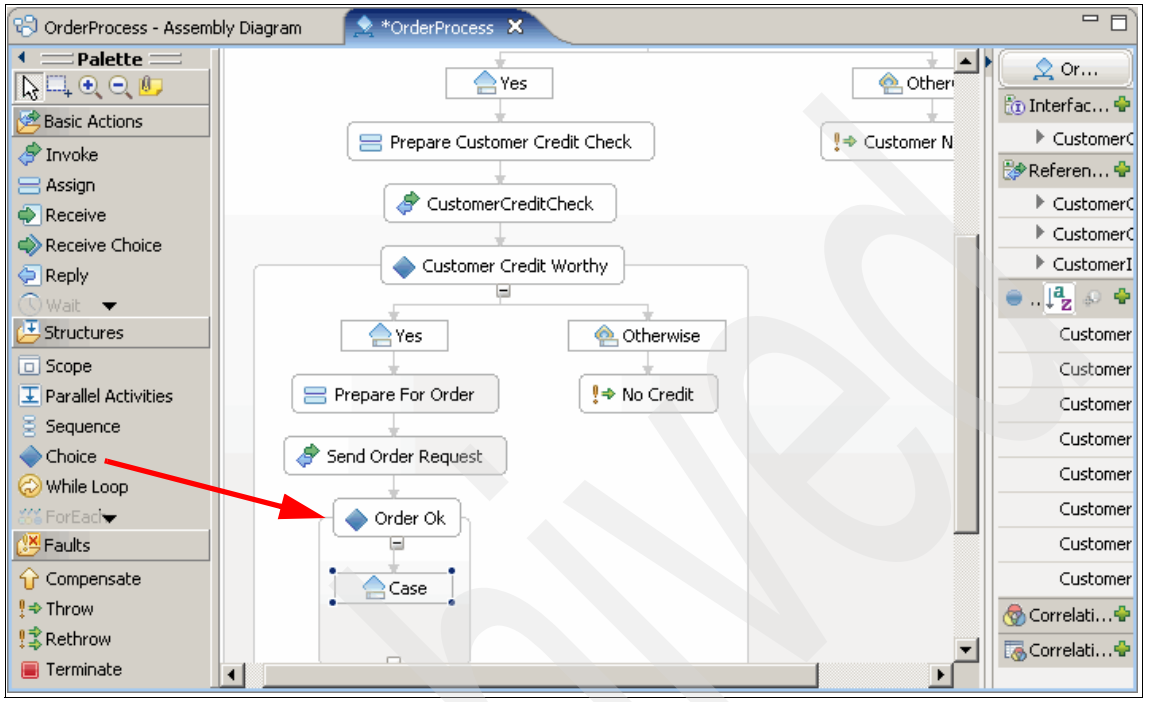

Figure 6-61 Adding a Choice node

42. Change the name of the Case to `Yes`. In the Properties view select **Details**. Select **Expression Language: Same as Process (Java)** and **Expression Type: Visual** (Figure 6-62).

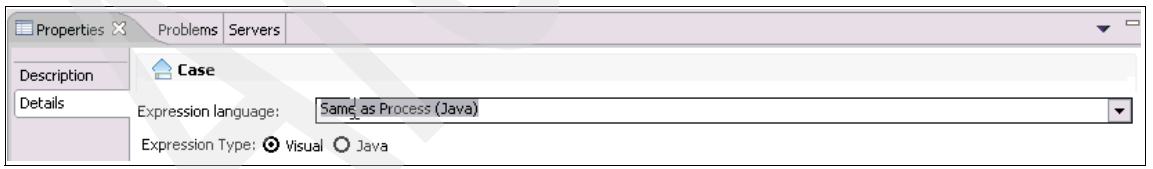

Figure 6-62 Case details

43. In the editor area select **True** and find **CustomerOrderRequestRp**. Expand until you find the field **retcode** and select it.

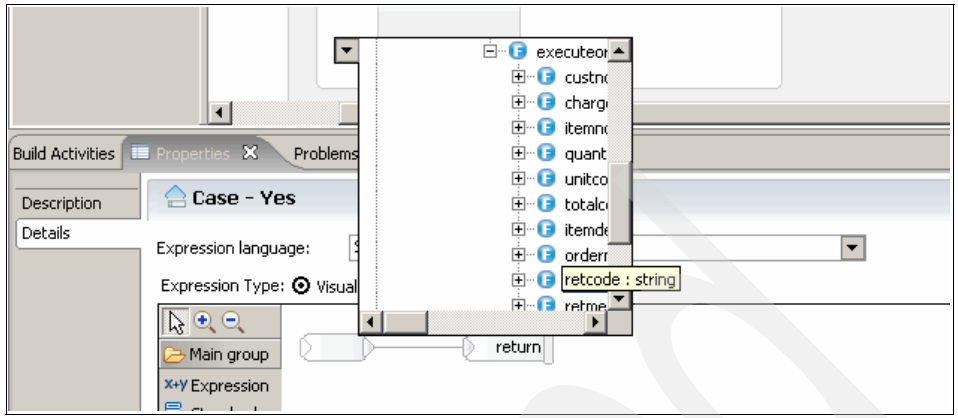

Figure 6-63 The expression

44. Select **==** as the operator, **String** as the Data Type and **00** as the value. The result is shown in Figure 6-64.

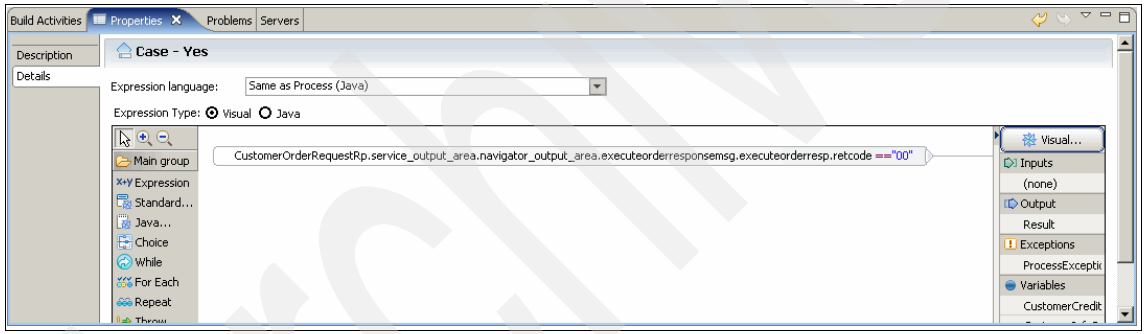

Figure 6-64 The final expression

45. Using the experience from the previous steps, from the palette select an **Otherwise** node and add it after the **Order Ok**; add a **Throw Fault** node after the **Otherwise** node. Rename the Fault `Order Not OK` (Figure 6-65 on page 135).

46. In the Properties view select **Details**, and select Fault Name **OrderNotOk**.

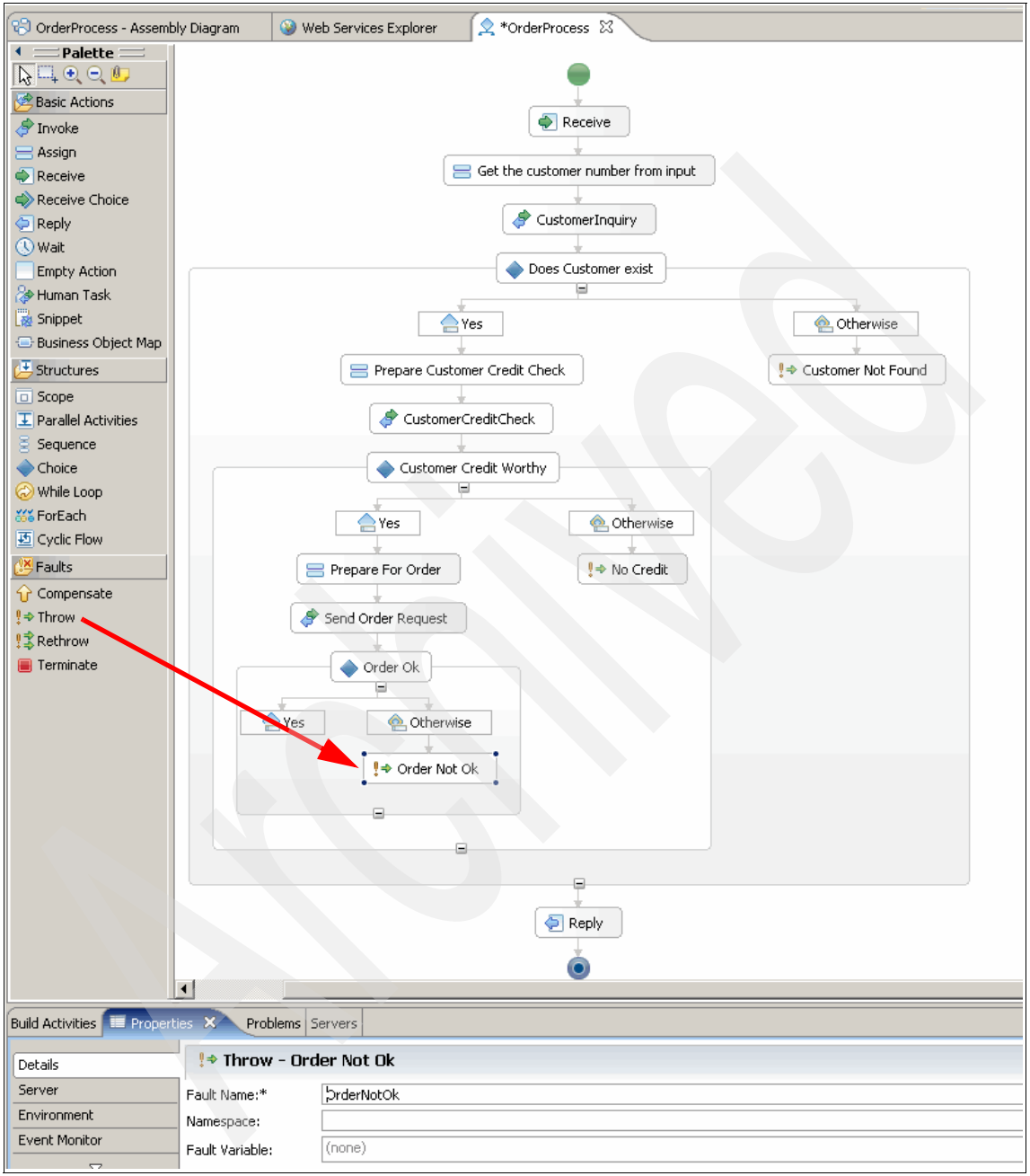

Figure 6-65 Order Not Ok

47. Select an **Assign** node from the palette and place it after the **Yes** Choice node. Rename it `Confirm Order`.

48. In the Properties view select **Details** and map the variables.

   – The From variable is **CustomerOrderRequestRp**, fully expanded down to **retmessage.**

   – The To variable is **CustomerOrderReply** → **Message**.

   See Figure 6-66.

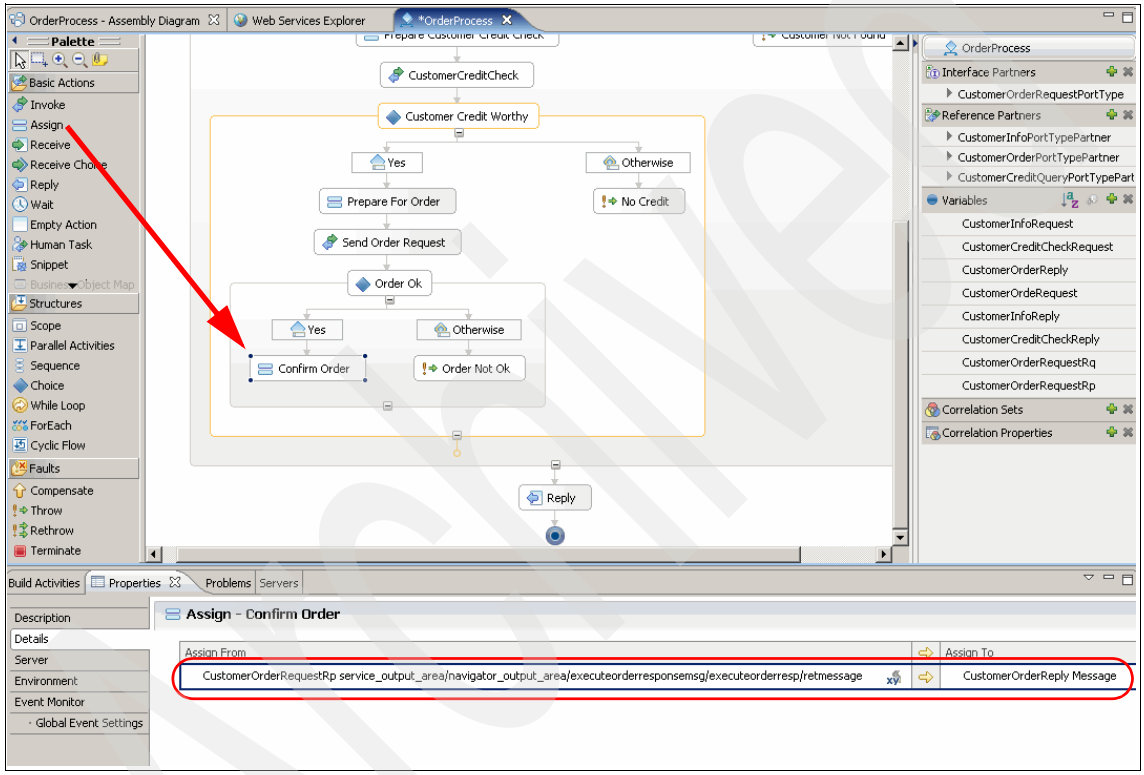

Figure 6-66 Adding an Assign node for confirming order

49. Check the Details tab for the **Reply** node, and make sure the variable is set to **CustomerOrderReply**, as shown in Figure 6-67 on page 137.

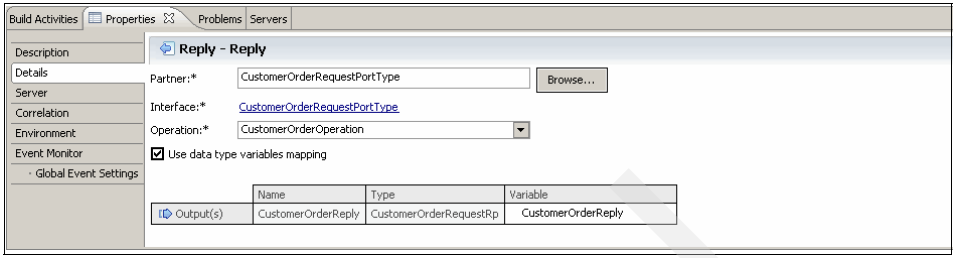

Figure 6-67 Checking details for Reply Node

50. Save all files by selecting **File** → **Save All**, as shown in Figure 6-68.

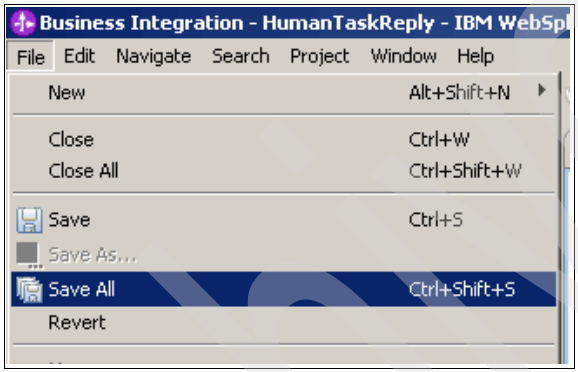

Figure 6-68 Saving all the work

51. Right-click **OrderProcess** in the Business Integration view and select **Test** → **Test Module**, as shown in Figure 6-69.

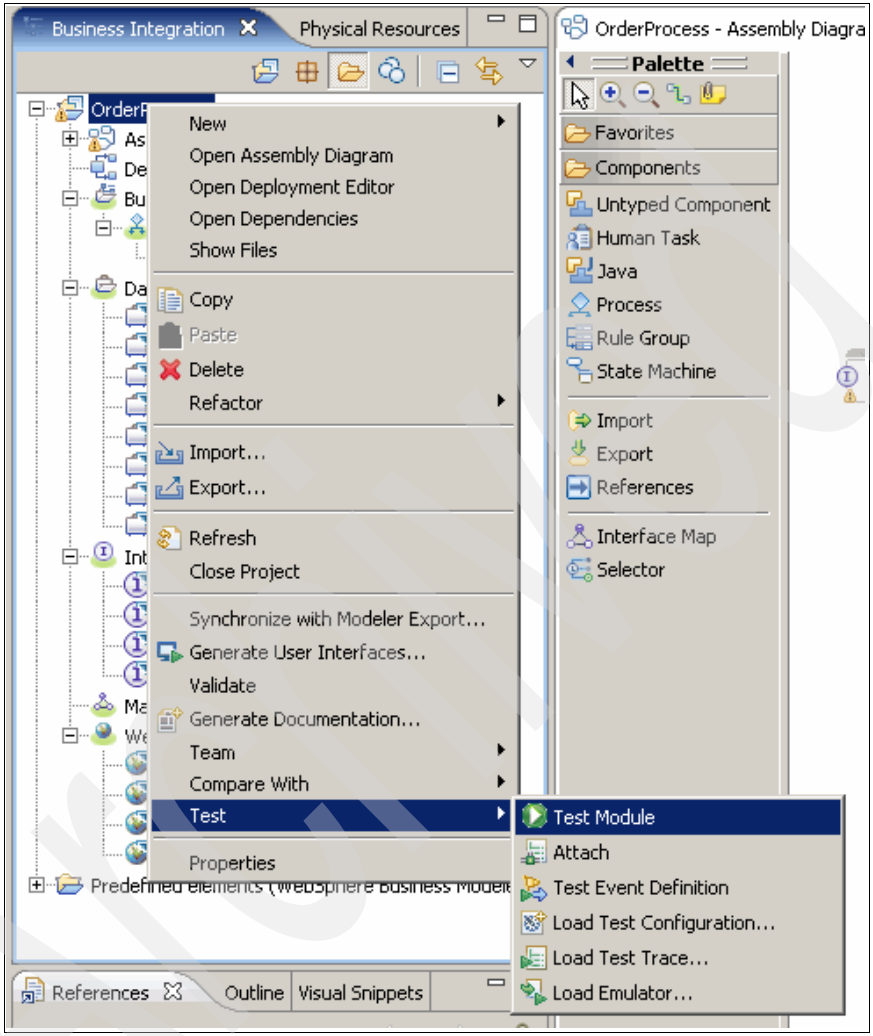

Figure 6-69 Testing the module

52. In the **OrderProcess_Test** window that opens, make sure that **Component** → **OrderProcess** is selected (Figure 6-70).

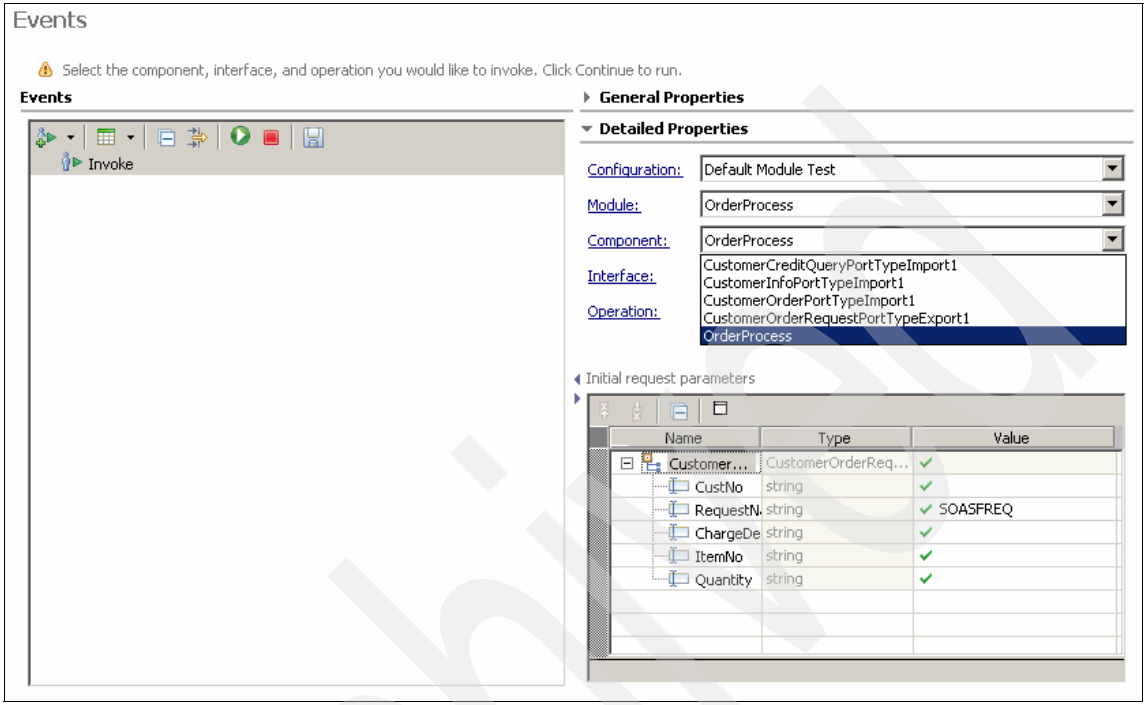

Figure 6-70 Making sure the right component is selected

53. In the “Initial request parameters” part of the window, double-click in the **Value** field after **CustNo** and type the value 001. Specify the other variables in the same way, as shown in Figure 6-71.

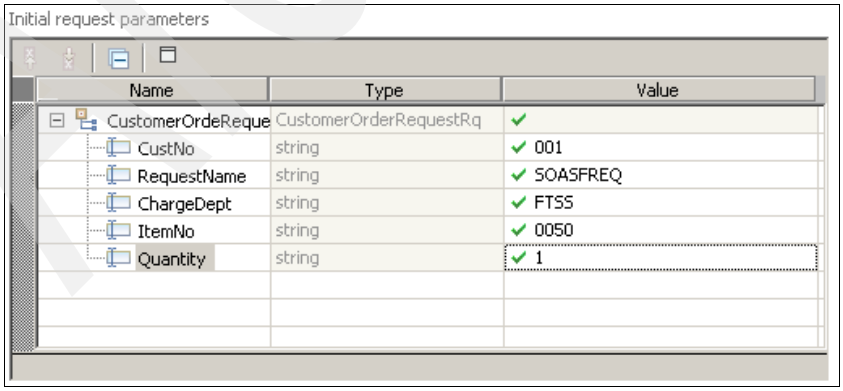

Figure 6-71 Test data

54. Click the **Continue** button (Figure 6-72).

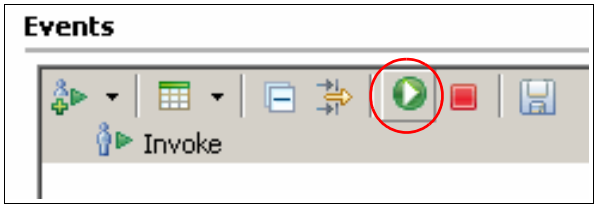

Figure 6-72 Continue button

55. Click **Finish** on the “Deployment Location” dialog (Figure 6-73).

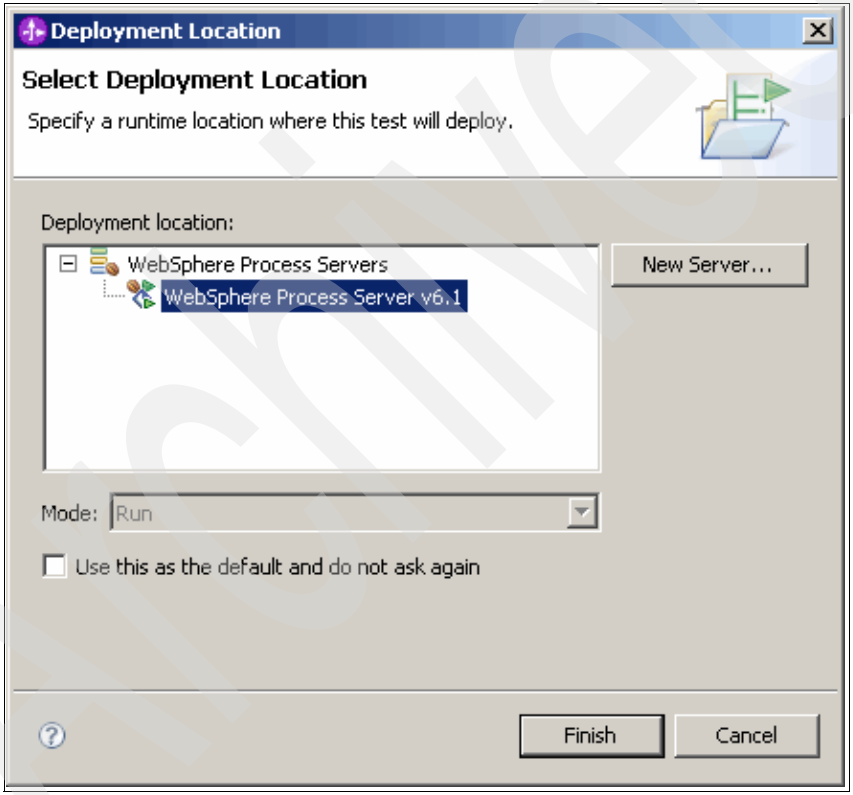

Figure 6-73 Deployment Location dialog

56. Click **OK** to accept the user ID and password (admin / admin) for the server since the security is turned on (Figure 6-74).

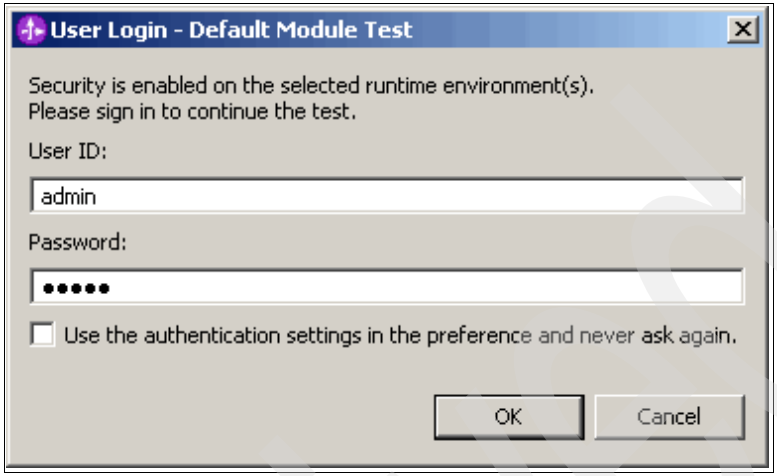

Figure 6-74 User login pop up

57. The progress indicator window shown in Figure 6-75 is displayed while validation takes place and while the application is deployed to the local WPS server. This process might take between 5 and 10 minutes, depending on your workstation, when you do this for the first time. Subsequent tests will go significantly faster (probably less than 1 minute).

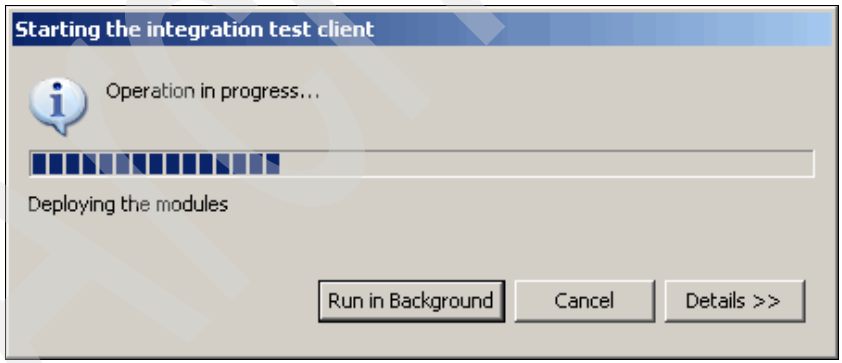

Figure 6-75 Deployment progress window

58. When the process is complete, you will see `ORDER SUCESSFULLY PLACED` in the Message → Value field in the **Return parameters** section. See Figure 6-76.

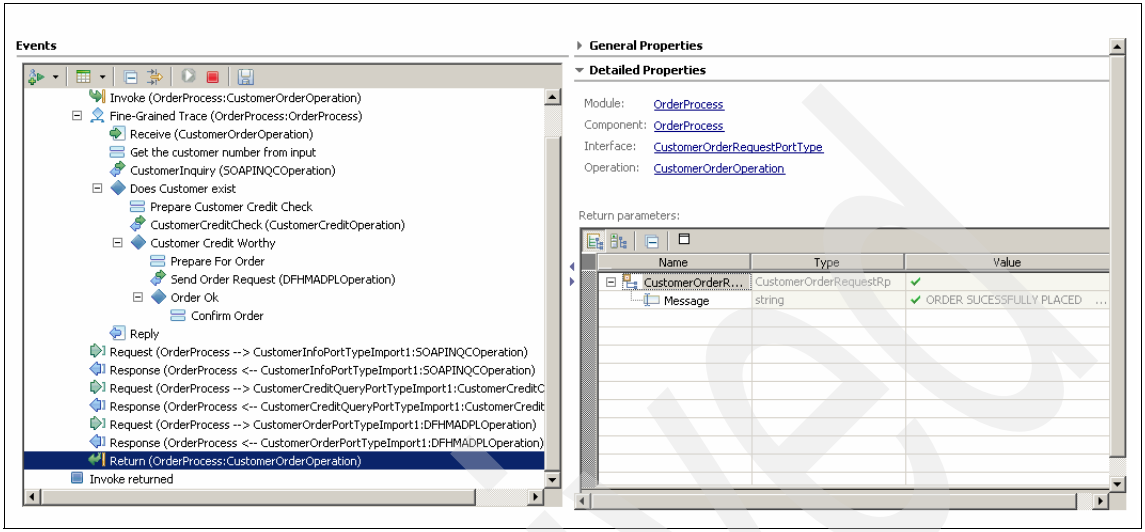

Figure 6-76 Process has run successfully

What has happened?: If you were able to run the process successfully, your process has called a Web service in CICS to look up Customer information based on the CustNo value you provided. Subsequently it invoked a message flow in WebSphere Message broker calling a DB2 Stored Procedure to check the credit of that customer. Finally, if the customer exists and his or her credit is OK, an order was placed for item number 0050 for a quantity of 1. You can test the process with other values, if desired.

Now that the automated tasks have been completed and tested, we can add the Human Task for the situation where a customer number does not exist.

6.5 Adding a Human Task

In this section we explain how you can use WebSphere Integration Developer (WID) to add a Human Task to a process. A *Human Task* is a task that the system assigns to a person for completion. A Human Task has a particular role or resource assigned to do the task, which is the primary owner. A Human Task can also have escalations that define what should take place if some aspect of the Human Task does not complete on time.

In this example, if the customer is not known then a task will be assigned to an agent to determine if the customer is valid or not. When this situation occurs the process comes basically to a halt, until the Human Task is completed by the agent. WPS will wait with continuation of the process until it receives a trigger.

A Human Task is just like any other task and requires that a request and reply be defined for it. First we define the Request object using these steps.

1. Expand the **OrderProcess** module to show its parts. Create a new Business Object by right-clicking **Data Types** → **New** → **Business Object**, as shown in Figure 6-77.

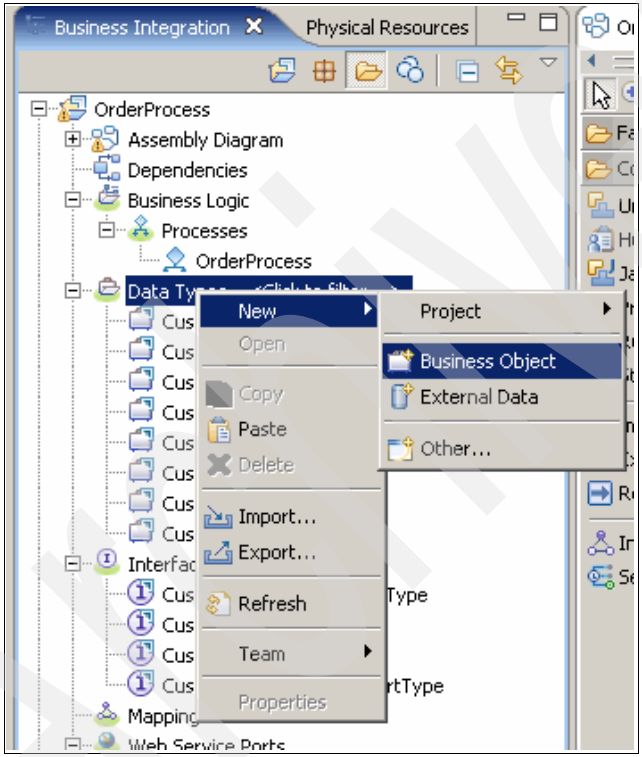

Figure 6-77 Creating a new Business Object for the Human Task

2. In the **New Business Object** window add `HumanTaskRequest` to the Name field and click **Finish** (Figure 6-78 on page 144). Do not change any other fields.

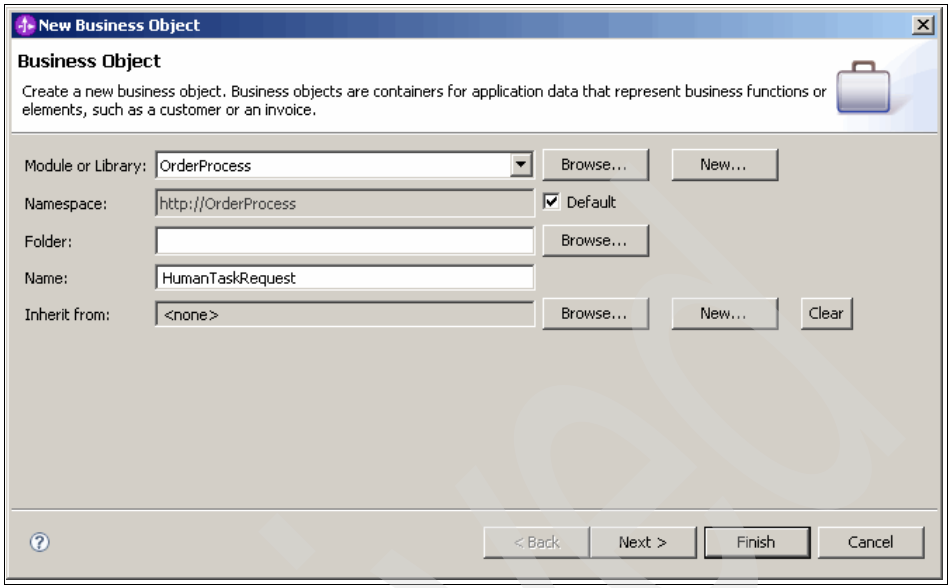

Figure 6-78 Adding the HumanTaskRequest Business Object

3. The new **Business Object** appears in the main WID editing window. Attributes of the request must be added to the Business Object. Select the **Add** icon from the editing window (Figure 6-79).

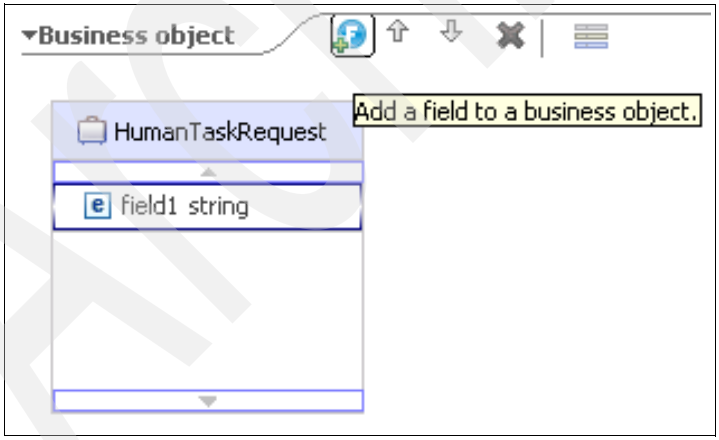

Figure 6-79 Adding an attribute to the HumanTaskRequest Business Object

4. A new row with the attribute will appear in the HumanTaskRequest. Rename this attribute (with initial name **field1**) to `CustNo` and leave it defined as a string (Figure 6-80 on page 145).

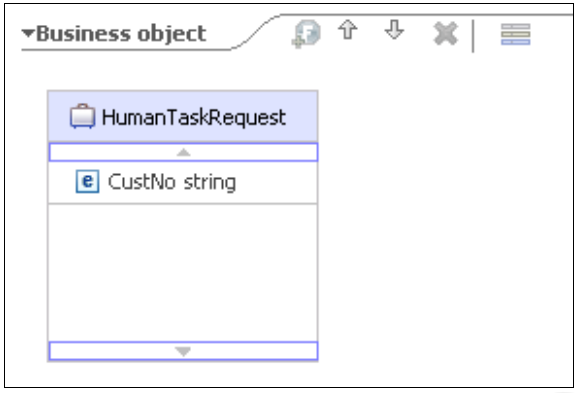

Figure 6-80 Adding the Customer number to the HumanTaskRequest

5. Create a reply for the Human Task. In this example we are only going to send back a message indicating whether the customer is valid or not valid. As demonstrated previously, expand the **OrderProcess** module to show its parts. Create a new Business Object by right-clicking **Data Types** → **New** → **Business Object**.
6. In the **New Business Object Window** add `HumanTaskReply` to the Name field and click **Finish** (Figure 6-81). Do not change any other fields.

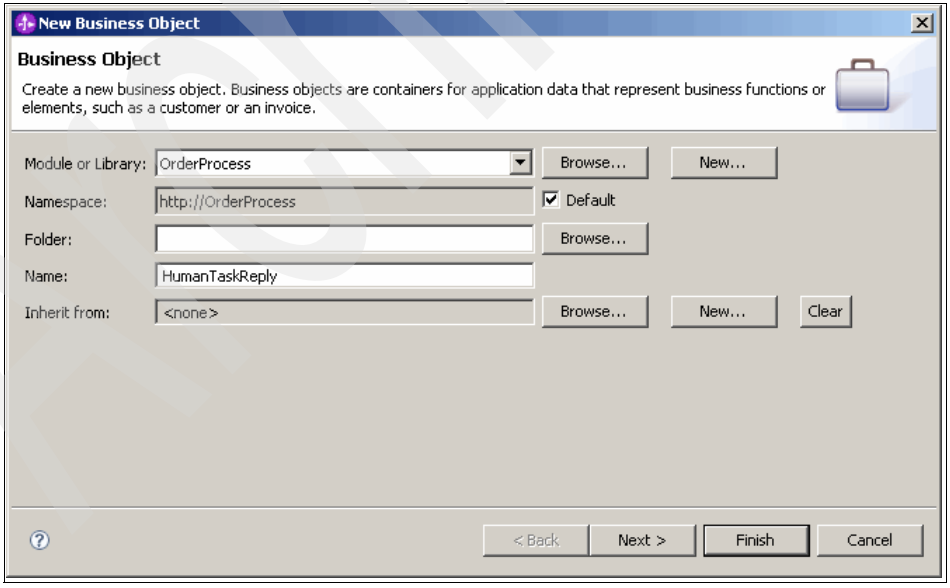

Figure 6-81 Adding the Human Task Reply Business Object

7. The new Business Object appears in the main WID editing window. Attributes of the request must be added to the Business Object. Click the **Add** icon from the editing window.
8. A new row with the attribute will appear in the HumanTaskRequest. Rename this attribute to `Message` and leave it defined as a string (Figure 6-82). This will be the reply message from the operation.

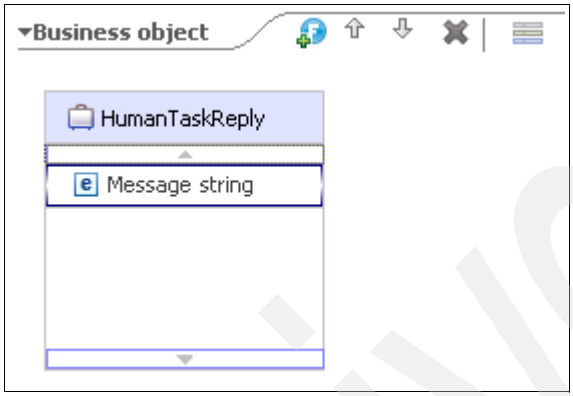

Figure 6-82 Adding the reply message for the HumanTaskReply

9. Save your changes so the work done is not lost. Do this by selecting **File menu** → **Save All** (Figure 6-83).

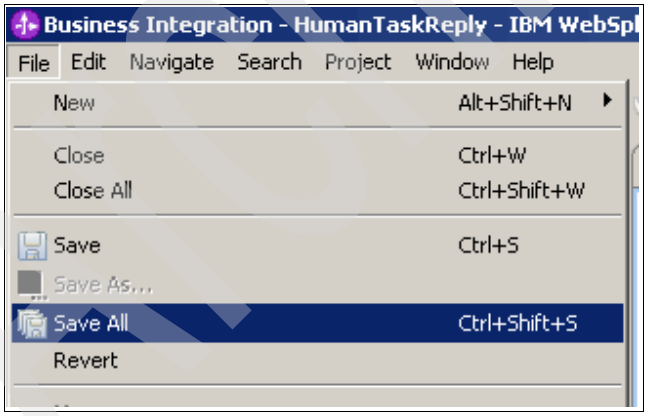

Figure 6-83 Save all the changes

10. A new *Interface* must be added to the Human Task. This will describe the request and reply to the Human Task, which we created in the previous steps. To create a new Interface, right-click **Interfaces** → **New** → **Interface** (Figure 6-84).

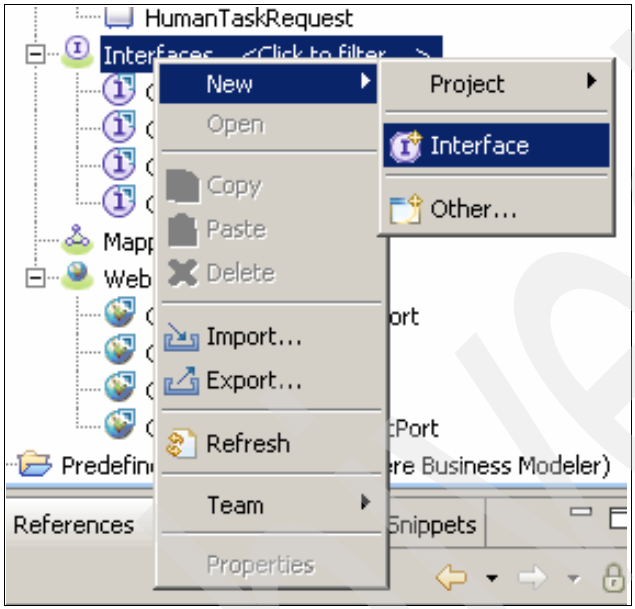

Figure 6-84 Creating a new interface for the Human Task

11. In the pop-up window name the new interface `HumanTask` and click **Finish** (Figure 6-85). Do not change any other fields.

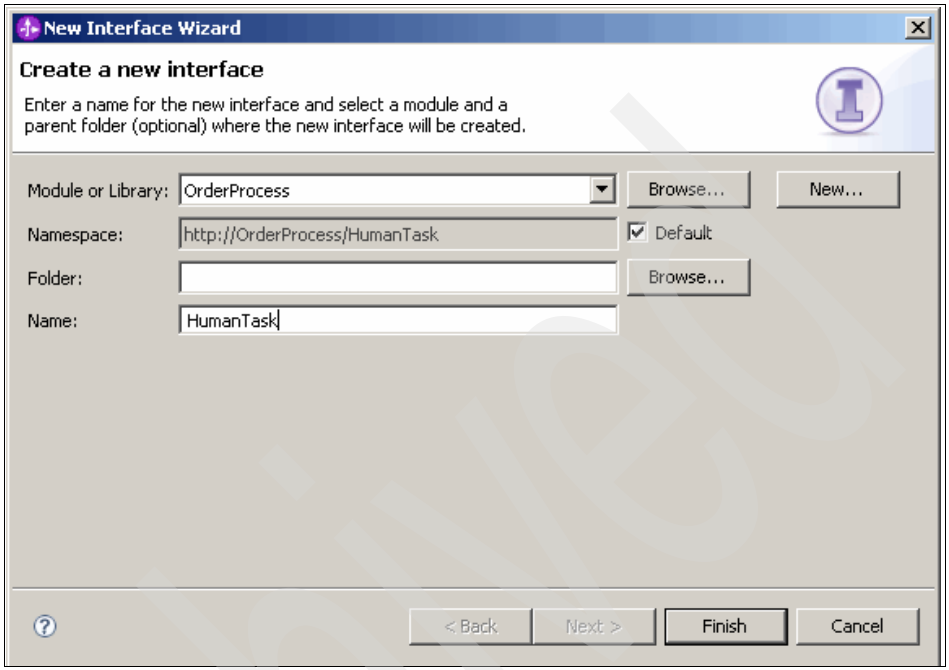

Figure 6-85 Creating the interface for the Human Task with specifications

12. Once the new interface is created the main editing window will appear with the new interface. The request and reply that were created earlier must be added to the interface. This is done by clicking the **Add Request Response Operation** icon as show in Figure 6-86.

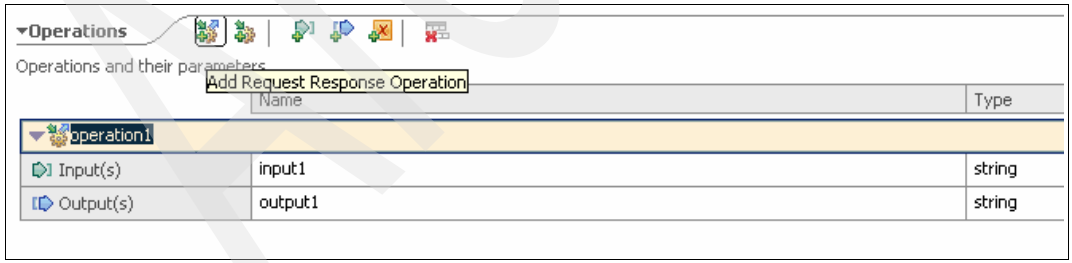

Figure 6-86 Adding the Request Response Operation to the Human Task Interface

13. Change the Operation Name, initially shown as **operation1**, to **HumanTask** (Figure 6-87).

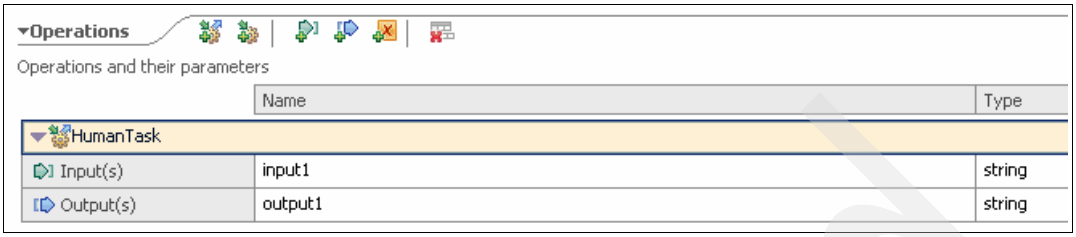

Figure 6-87 Changing the name of the operation for the Human Task interface

14. Change **Input(s)** to **HumanTaskRq**. You can do this by clicking **Input1**. Clicking the type produces a drop-down list, as shown in Figure 6-88. Select **HumanTaskRequest** from the list.

Figure 6-88 Changing the input to the Human Task Interface

15. Change **Output(s)** to **HumanTaskRp** by clicking **Output1**, accessing the type drop-down list, and selecting the **HumanTaskReply** (Figure 6-89 on page 150).

Figure 6-89 Adding the output to the Human Task Interface

16. Save your changes by selecting **File menu** → **Save All**.

17. The Human Task can now be integrated into the **OrderProcess**. Display the OrderProcess flow shown in Figure 6-90 by selecting **OrderProcess** → **Business Logic** → **Processes** and double-clicking **OrderProcess** in the navigation pane.

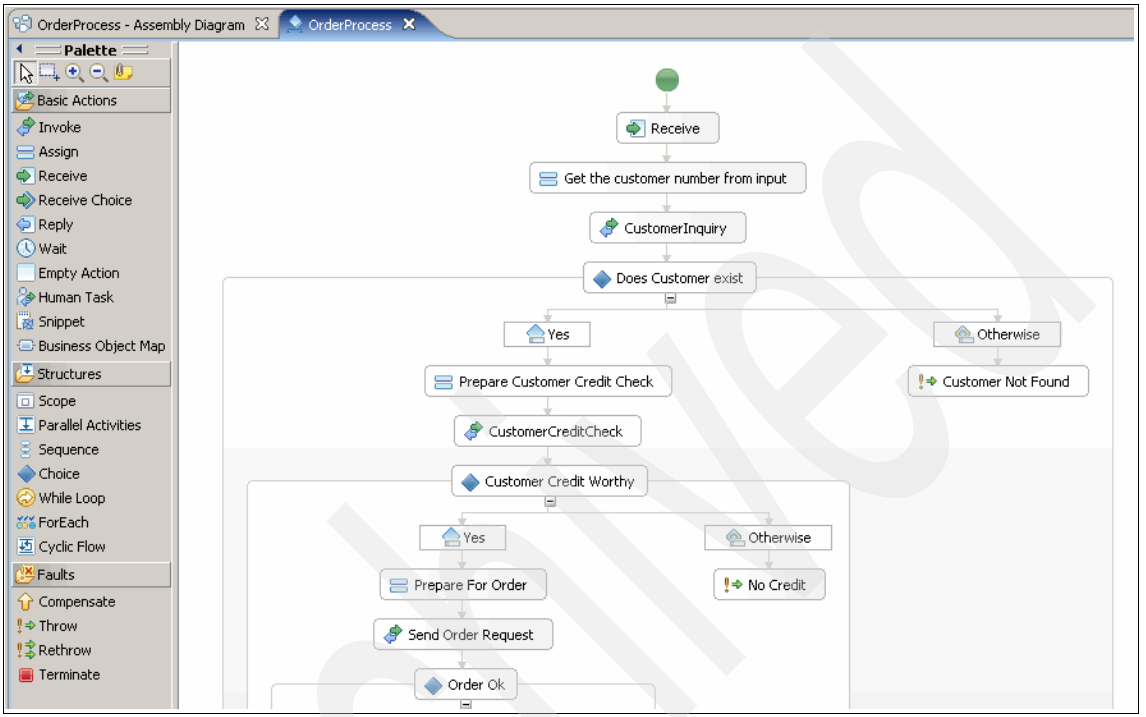

Figure 6-90 The OrderProcess process flow

18. Delete the **Customer Not Found** assignment by right-clicking it and selecting **Delete (**Figure 6-91).

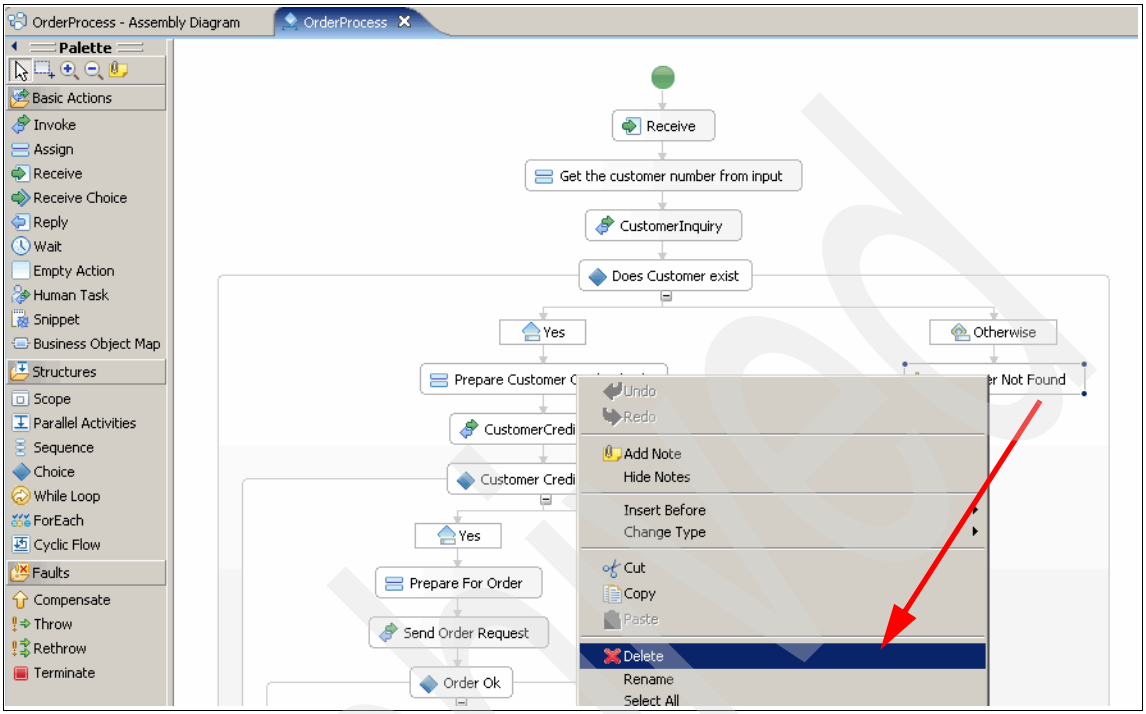

Figure 6-91 Deleting the assignment from the process flow

19. Drag an **Assign** node under the **Otherwise** node of the **Does Customer Exist**. Rename the node `Prepare Human Task` as shown in Figure 6-92.

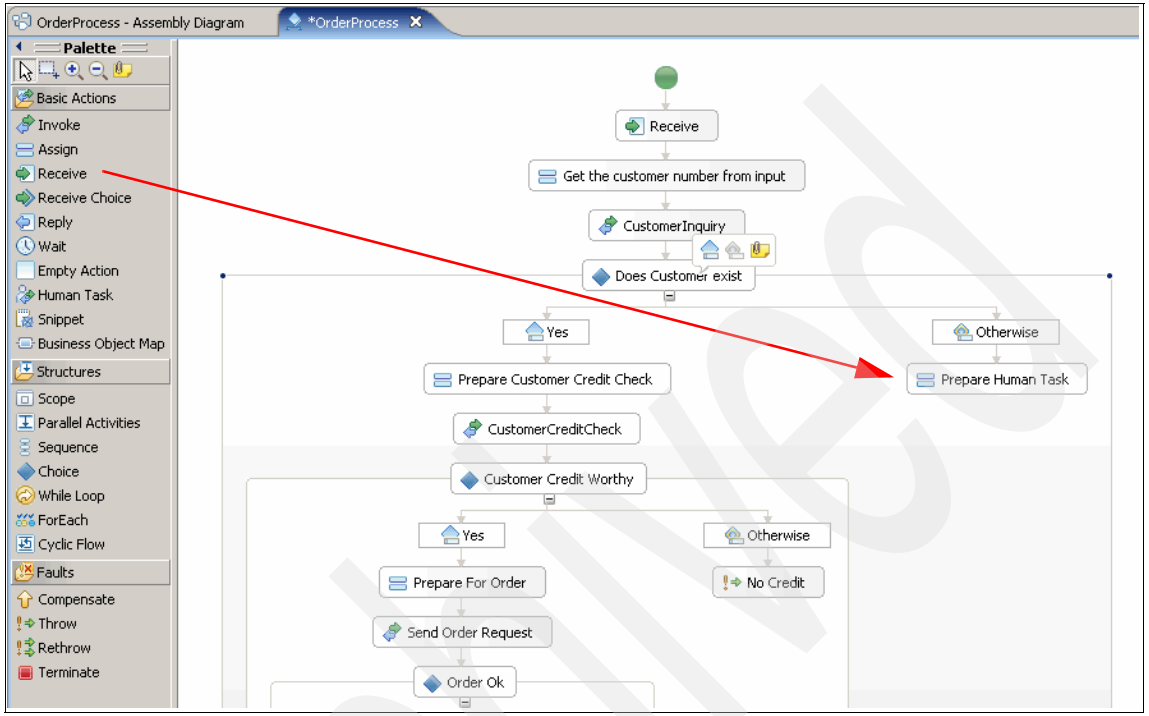

Figure 6-92 Adding an Assign node to for the Human task to the process flow

20. To handle the Human Task, new variables must be added. Click the plus icon in the **Variable** section to add a new variable called `HumanTaskInput` with a data type of **HumanTaskRequest**, as shown in Figure 6-93.

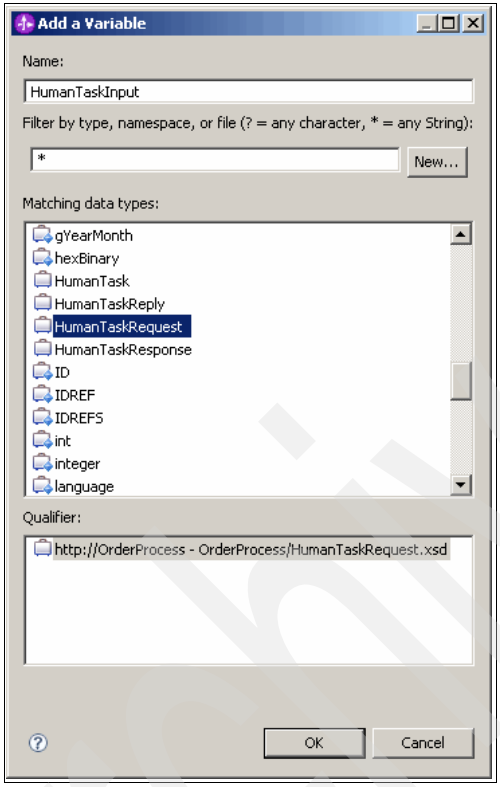

Figure 6-93 Adding an input variable to the process flow for the Human Task

21. Click the plus icon in the **Variable** section and add a variable for the reply called `HumanTaskOutput` with a data type of **HumanTaskOutput**, as shown in Figure 6-94 on page 155.

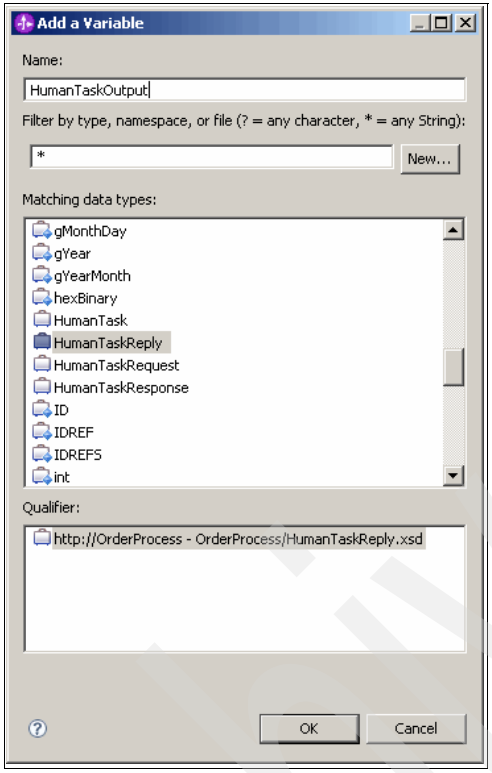

Figure 6-94 Adding an output variable to the process flow for the Human Task

22. The request that is being sent must be set properly for the **HumanTaskInput**. Perform the following steps to do this:

    a. Select the **Prepare Human Task** node and select **Properties** → **Details.**
    b. In the Assign From field click **Select From**. In the drop-down list expand the **CustomerOrdeRequest** and select **CustNo**.
    c. In the Assign To field click **Select To**. In the drop-down list expand the **HumanTaskInput** and select **CustNo**.

    This sets the Customer Number of the incoming request to the Customer Number of the Human Task. This is shown in Figure 6-95.

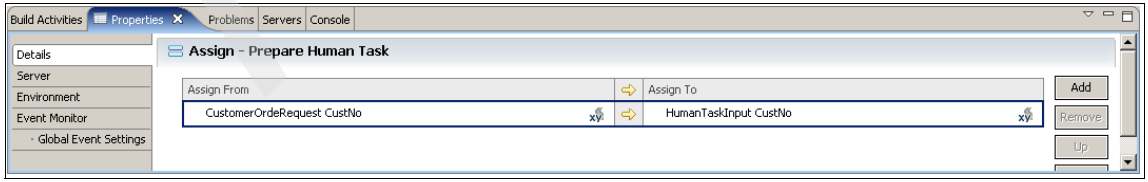

Figure 6-95 Mapping the request parameter to the Human Task

23. Add a **Human Task** node under the Prepare Human Task Assign node. Drag and drop a **Human Task** node from the Basic Actions palette, as shown in Figure 6-96.

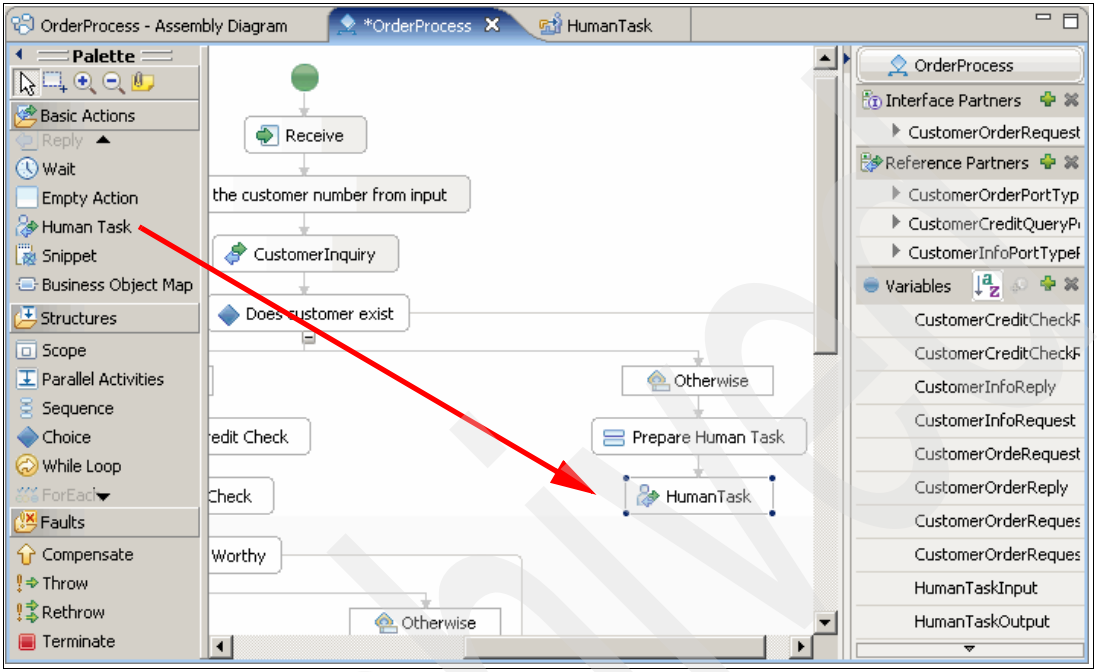

Figure 6-96 Adding a Human Task node

24. Rename the new Human Task to `HumanTask` and select the **HumanTask** interface that was created earlier from the interfaces list in the pop-up window (Figure 6-97). Click **OK**.

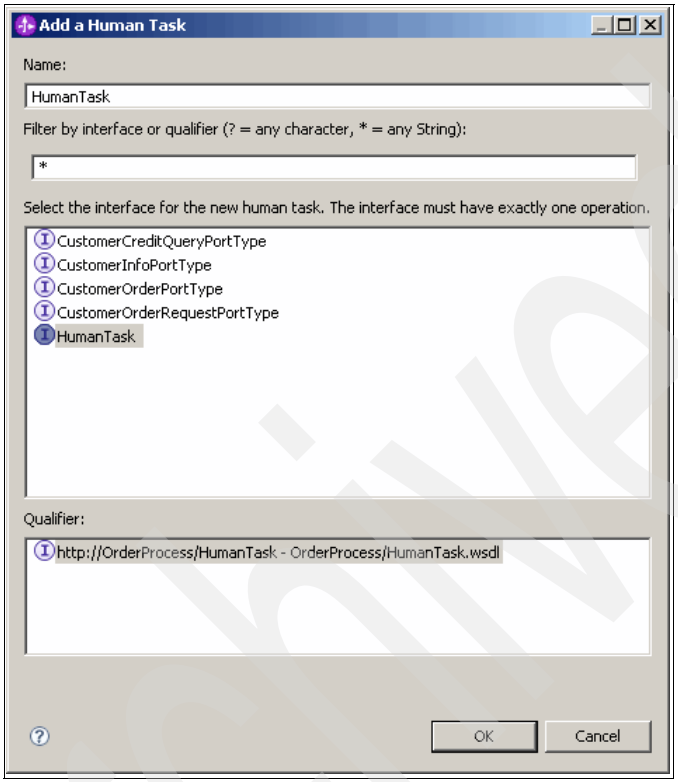

Figure 6-97 Defining the Interface for the HumanTask

The Human Task now only has one operation, called "**HumanTask,**" which we created in the interface section.

A new Human Task will open in the main editing window.

25. In the **Properties** tab select **Details**. In this example, anyone who is authorized can process the Human Task. We allow this by selecting **bpe/staff/everybodyconfiguration** in the **People Directory (JNDI name)** as shown in Figure 6-98.

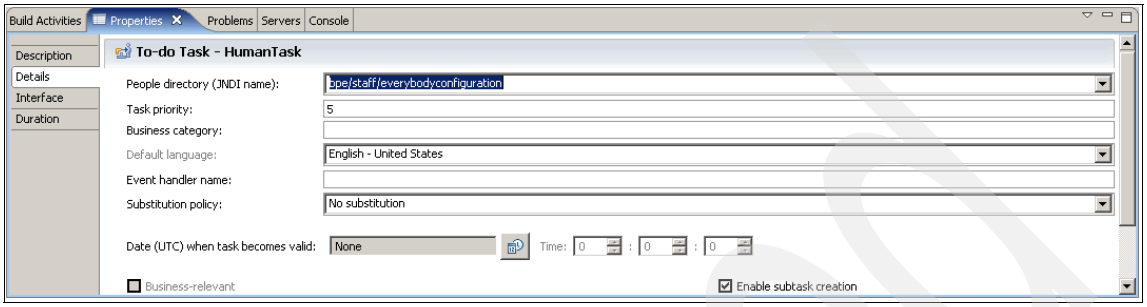

Figure 6-98 Changing the authorization details for the Human Task

26. The variables of the Human Task must be defined for the physical task. Select the **HumanTask** node from the Process Flow Diagram and select **Properties** → **Details**.

27. Set the input and output variables as shown in Figure 6-99.

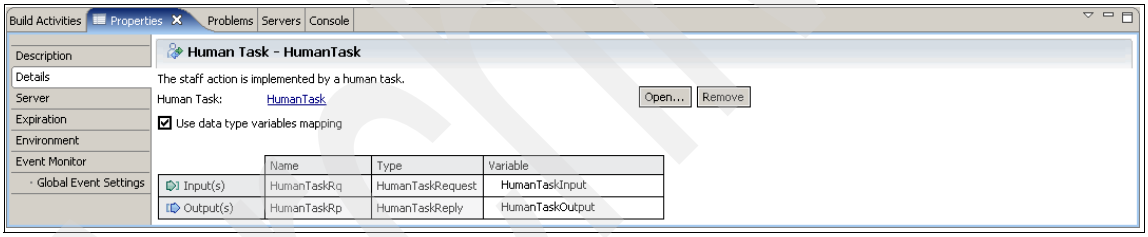

Figure 6-99 The Human Task input and output variables have been set

28. The response from the Human Task must be set and passed to the return of the overall process flow. Select an **Assign** node from the Basic Action folder in the palette, drag it under the **HumanTask** node and rename it `Return From Human Task`, as shown in Figure 6-100.

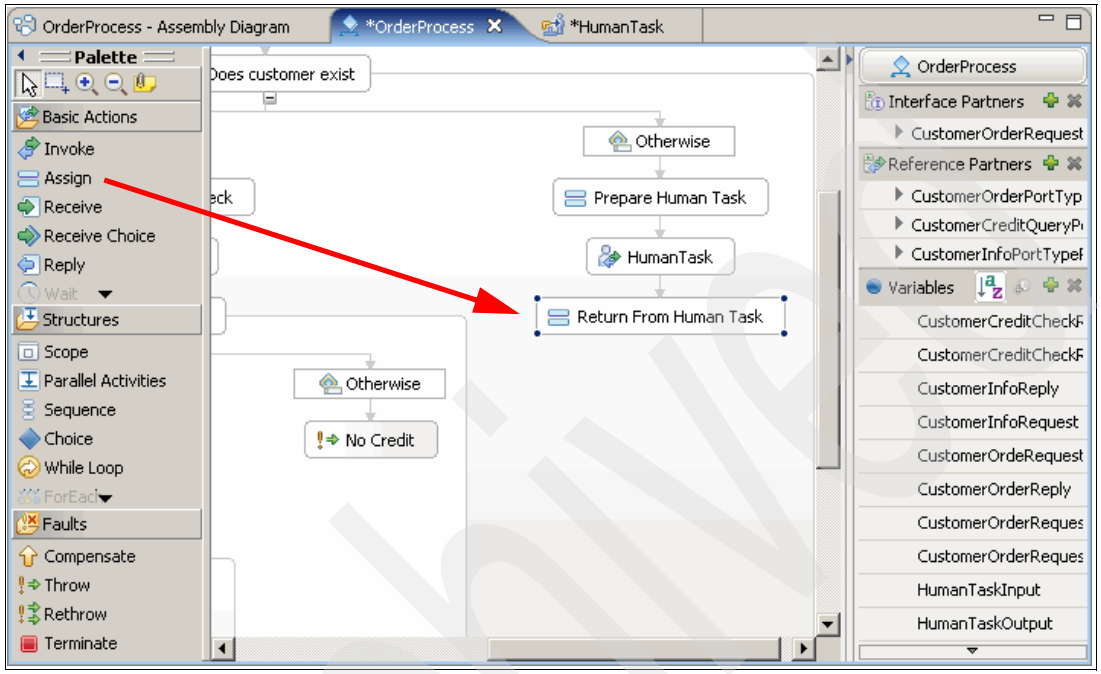

Figure 6-100 Adding an Assign node for the response from the Human Task

29. The reply that is being sent must be set properly for the **HumanTaskOutput**. This is done by selecting the **Return from Human Task** node and selecting **Properties** → **Details**. Set the variable assignment as shown in Figure 6-101.

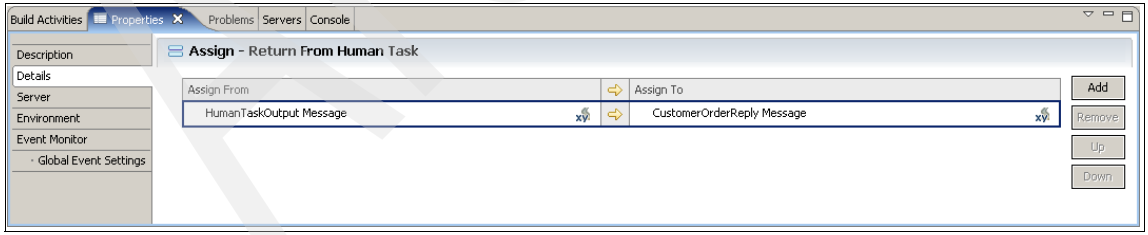

Figure 6-101 Setting the response message from the Human Task

30. Save your changes by selecting **File** → **Save All**.

    You will notice that the Human Task node contains an error. Let's find out what happened.

31. Go to the **Problems** tab and look at the error message.

    A task activity cannot be used in a microflow. In other words, to allow Human Tasks in the process, the process needs to be changed to allow *long running* tasks. This is done by right-clicking the error message and selecting **Quick Fix** as shown in Figure 6-102.

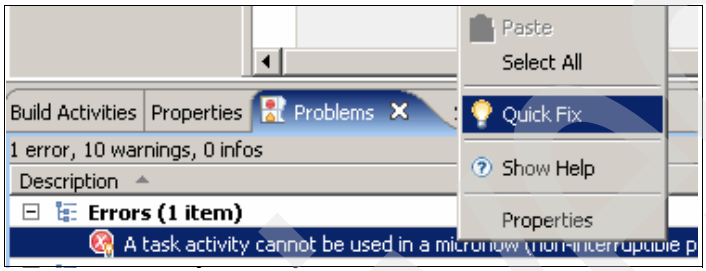

Figure 6-102 Fixing a problem in WID

32. A window will open with suggested fixes. Select the only fix available, as shown in Figure 6-103.

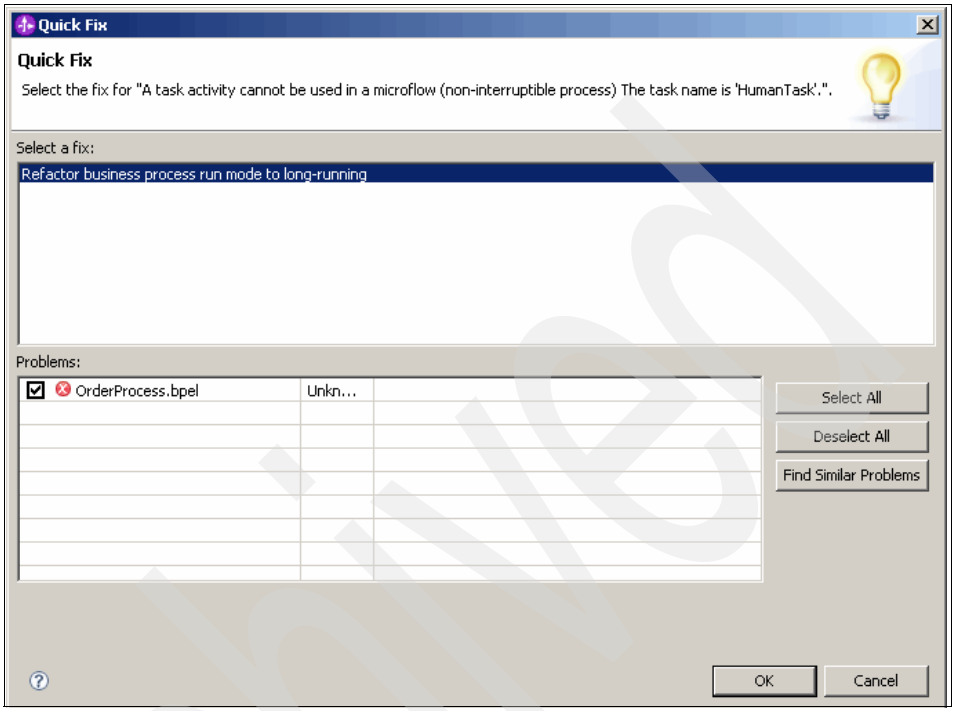

Figure 6-103 Quick Fix pop up window

33. Click **OK** and **Yes** in the next pop-up window. This will refactor the process to run in the long running mode instead of short running.

34. When prompted by a wizard, click **Preview** to view your changes. Expand the tree in the upper window to preview the changes (Figure 6-104 on page 162).

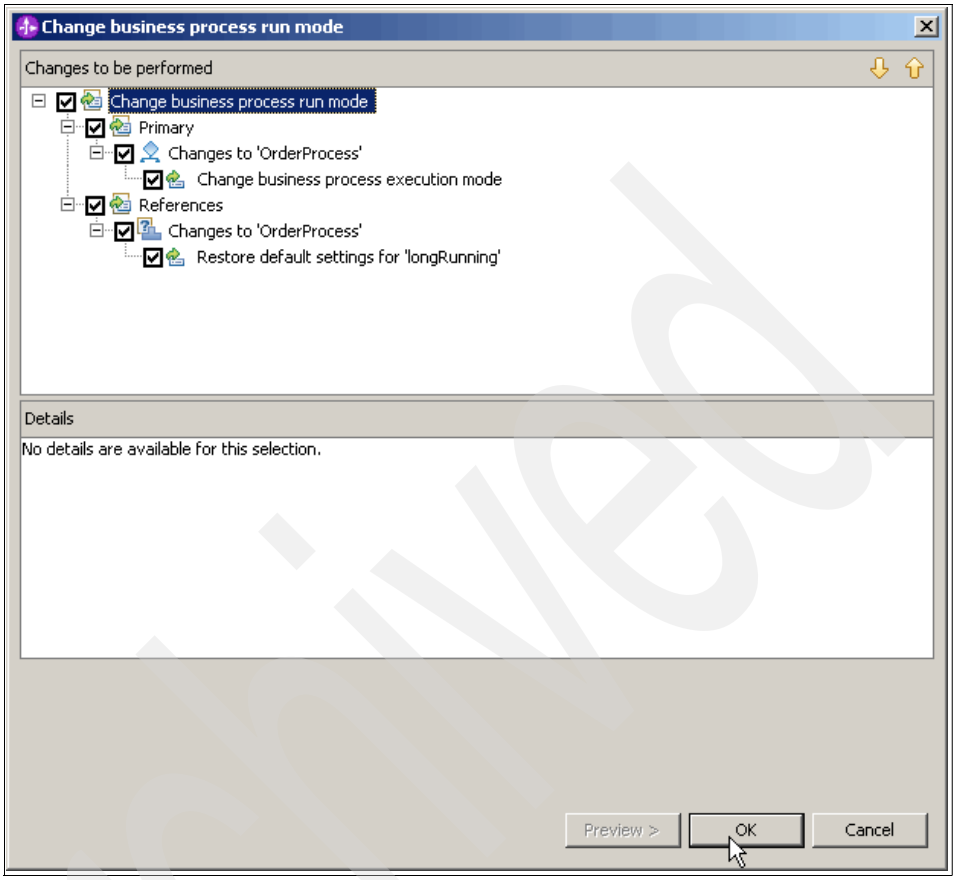

Figure 6-104 Preview of changes to the business process

35. Click **OK** when you are done inspecting the changes. The changes will be applied and the error will be gone.

36. It is a good idea now to close all open windows. If WID asks you to save something, click **Yes**.

6.5.1 Testing the Human Task with built-in client

Now that the Human Task has been successfully added to the process it is time to test it and show how the process flow will proceed when a customer is not recognized. The first test is done within WID Events.

1. From the **Business Integration Explorer**, right-click **OrderProcess** → **Test** → **Test Module** as shown in Figure 6-105.

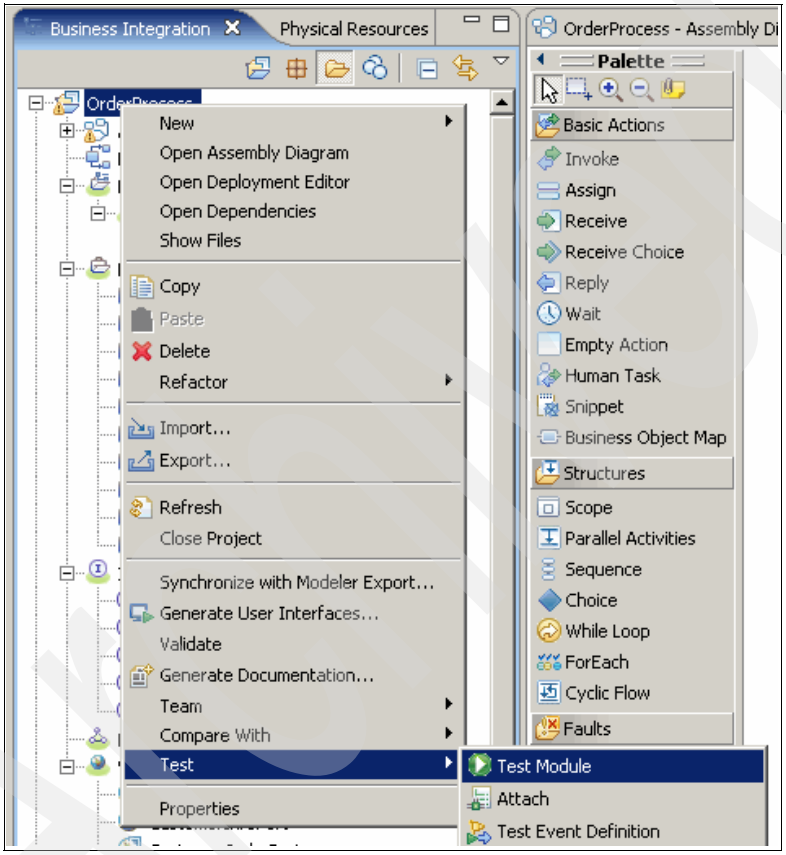

Figure 6-105 Testing the OrderProcess module

2. On the **Events** test frame select **OrderProcess** from the **Component** drop-down list shown in Figure 6-106. This will select the correct components to test and the proper request to be sent.

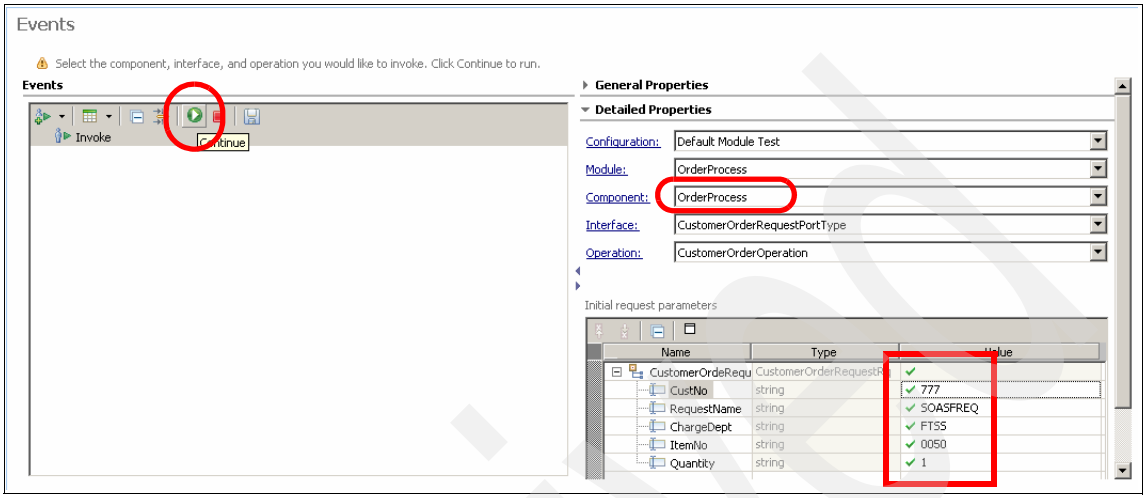

Figure 6-106 Selecting the correct request to test

3. In the **Initial Request Parameters** section add the value pairs specified in Table 6-3. Click **Continue**.

Table 6-3 Test variables

Name	Type	Value
CustNo	string	777
RequestName	string	SOASFREQ
ChargeDept	string	FTSS
ItemNo	string	0050
Quantity	string	1

4. A pop-up window will ask for the deployment location of the process. Select **WebSphere Process Server v6.1** and click **Finish** (Figure 6-107).

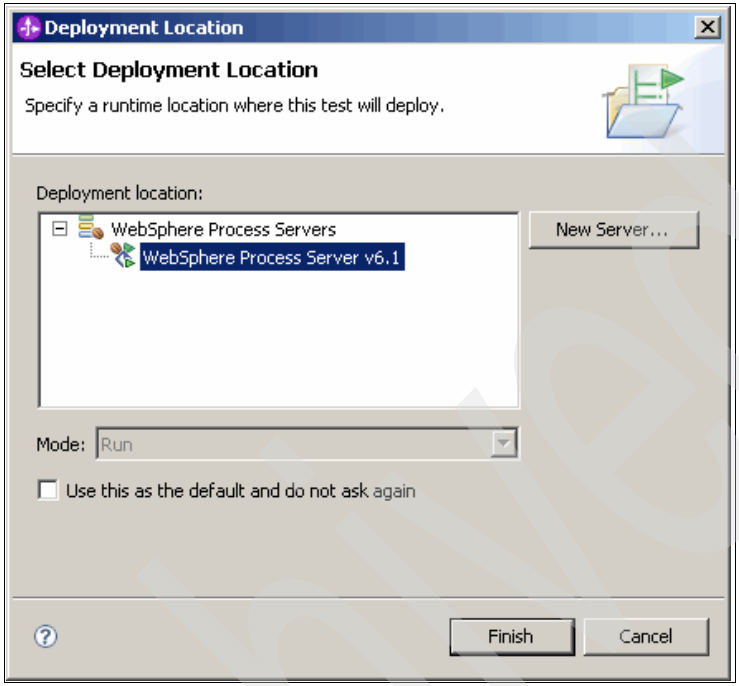

Figure 6-107 Testing the module on the WebSphere Process Server

5. Click **OK** to accept the user ID and password (admin / admin) in the User Login window.

   The application will now be deployed to the local WPS.

   Note: This can take quite a while, up to 10 minutes!

   On the **OrderProcess Events** frame the request will be invoked and you can now follow the process under the Events window. The customer number is not valid and will be placed in the Human Task queue for processing. The process is now on hold.

6. Launch the interface to the **Business Process Choreographer** by selecting **Servers** → **WebSphere Process Server v6.1.** Right-click **WebSphere Process Server v6.1** and select **Launch** → **Business Process Choreographer Explorer** as shown in Figure 6-108.

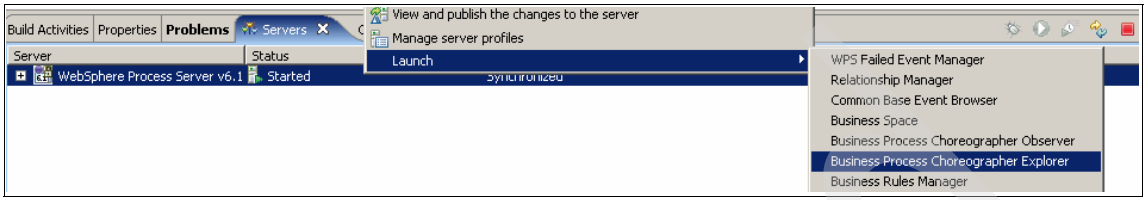

Figure 6-108 Launching the Business Process Choreographer

This will bring up a Web browser frame.

7. Type in the admin user ID and password (`admin` / `admin`) since security is turned on. Click **OK**. The Business Process Choreographer Explorer window shown in Figure 6-109 is opened.

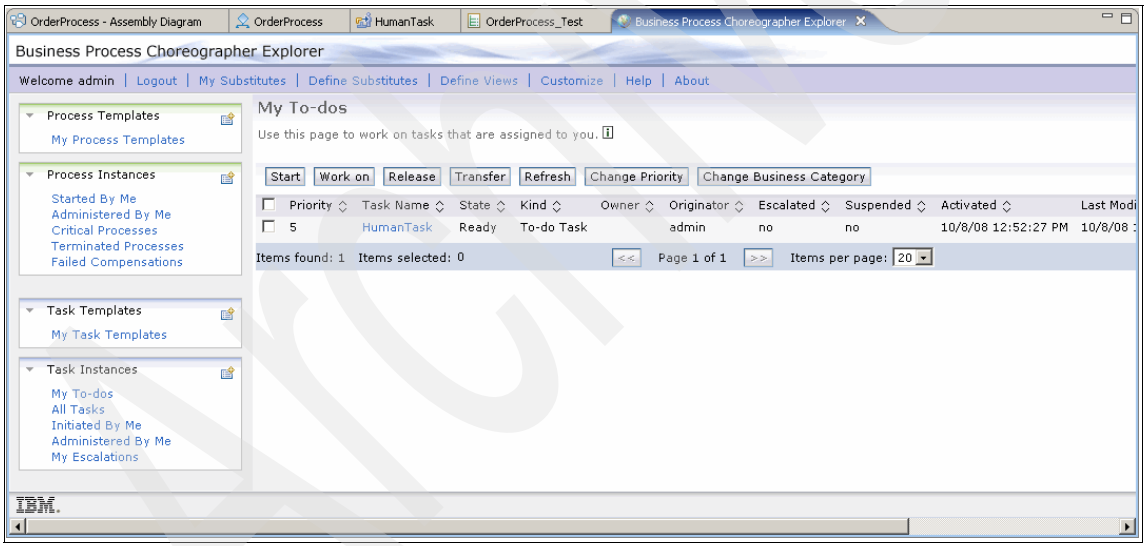

Figure 6-109 Business Process Choreographer Explorer

8. To see the details of your task waiting, check the **HumanTask** and select **Work on** as shown in Figure 6-110.

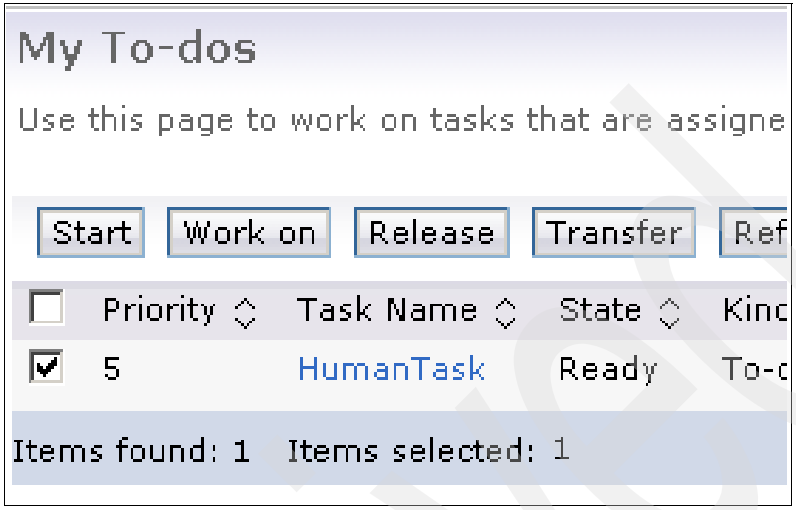

Figure 6-110 Selecting the HumanTask from the BPCE

9. The next window shows the input that was sent, which is the invalid Customer number and allows for a response message to be added as shown in Figure 6-111.

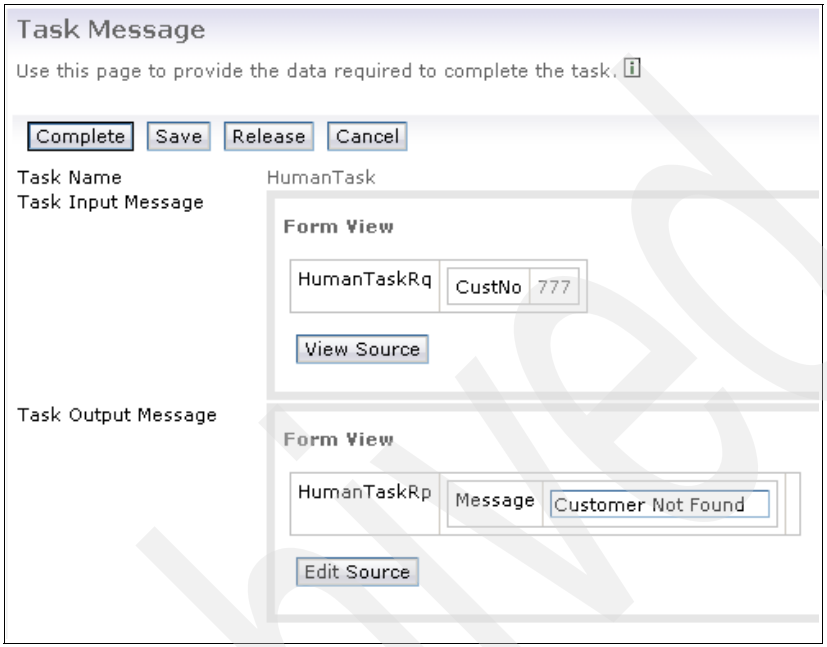

Figure 6-111 A Human Task that can now be worked on

10. Type in a message in the **Message** text box and click **Complete** to finish with the task, as show in Figure 6-111.

    Upon completion of the Human Task, the **OrderProcess Events** is updated with the response. Go back to the **OrderProcess_Test** window and click the **Return** event. You will see the message that was entered from the Human Task (Figure 6-112 on page 169).

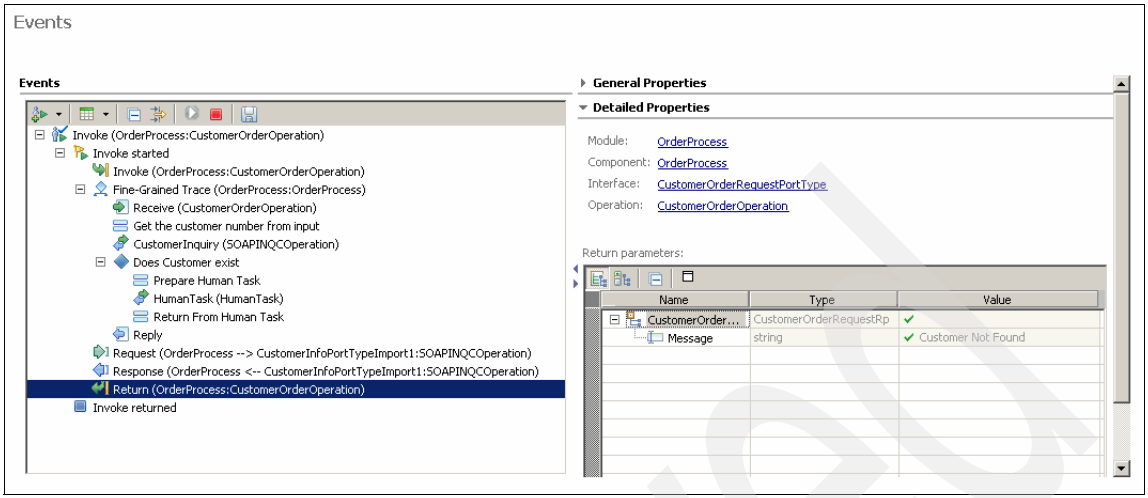

Figure 6-112 Checking the response message from the process

After the Human Task in the BPCE is completed, the process in WPS resumes and runs until the end.

You can now log out and close the BPCE window. Also close the Order test window (without saving).

6.6 Creating a Human Task client

If you do not want to use the standard BPCE, you can build a custom Human Task client. Here is how this can be done:

1. To generate the Human Task Client, right-click **OrderProcess** → **Generate User Interfaces** as shown in Figure 6-113.

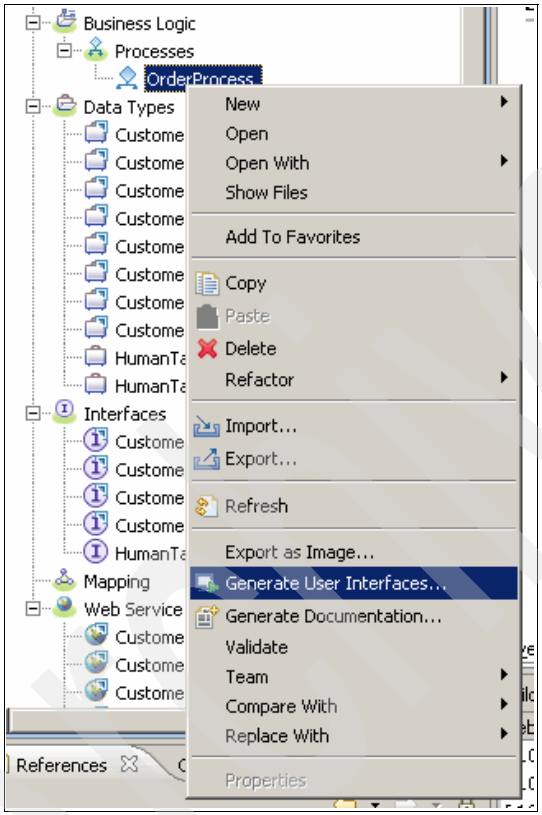

Figure 6-113 Generating the User Interface for the BCP

2. In the pop-up window select **JSF custom client** in the “Generator type” field and click **Select All** to select all Human Tasks, as shown in Figure 6-114. Click **Next**.

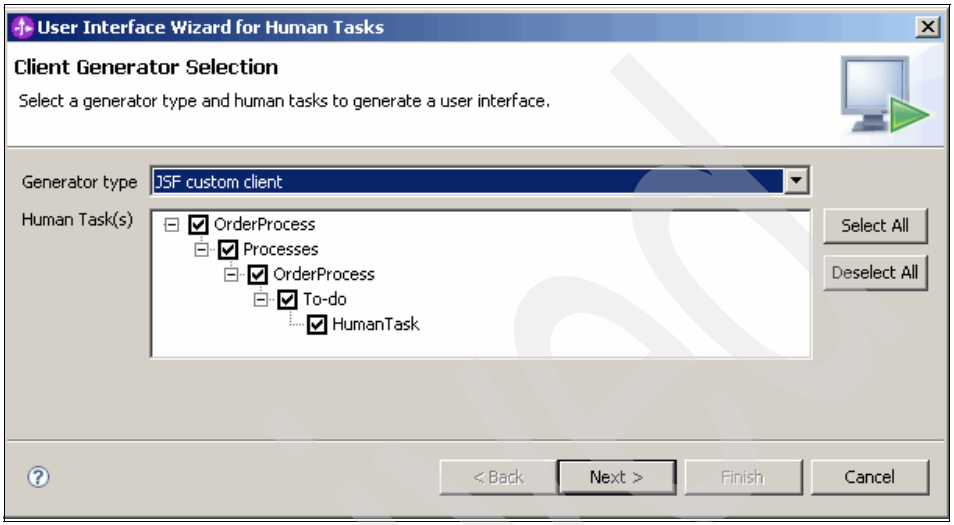

Figure 6-114 Selecting all the Human Tasks that will be allowed in this BPC client

3. The JSF client configuration window shows the styles that can be used, as well as where the client should connect to get information from WPS. Enter `OrderProcessHumanTaskClient` in the “Name of dynamic web project” field, and select **Cool Blue™ Style** from the Style drop-down list. Click **Finish** (Figure 6-115).

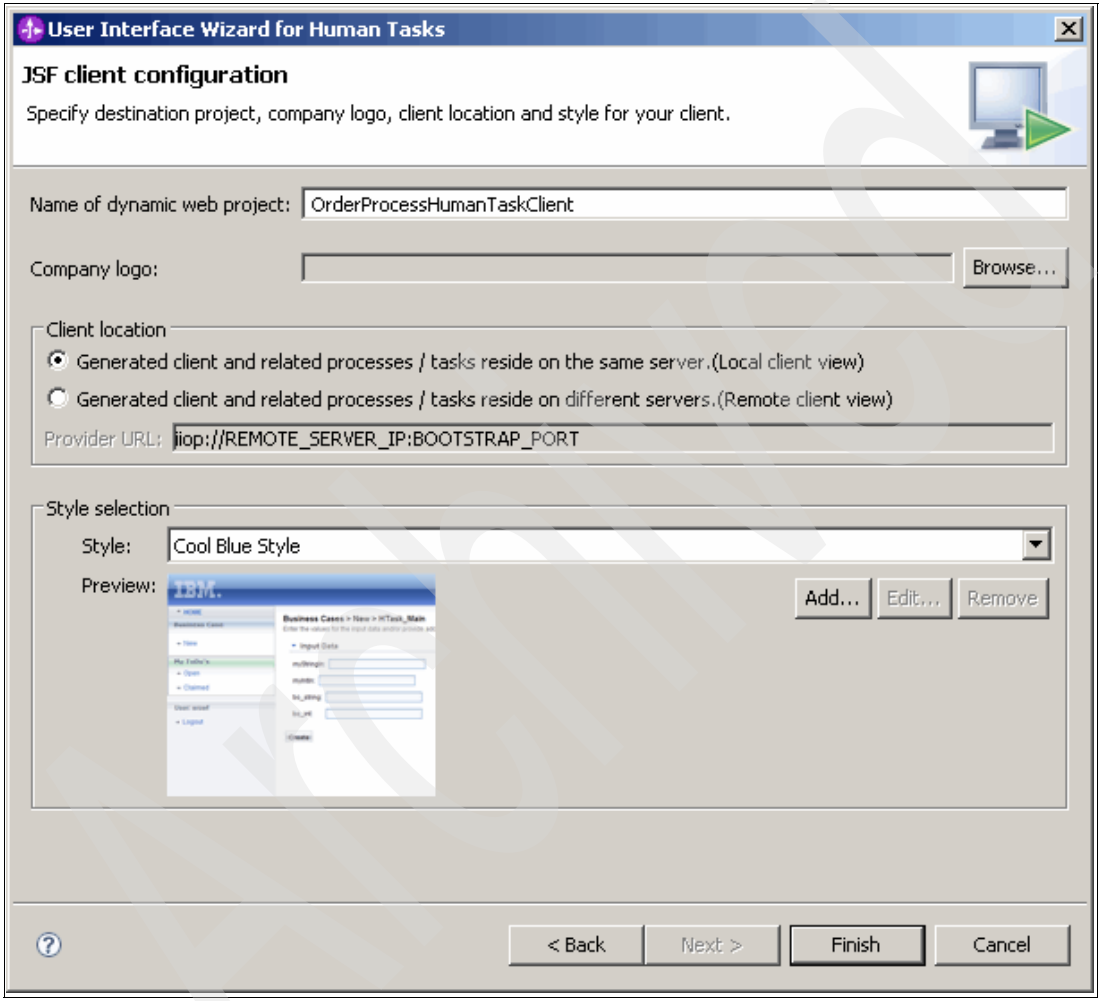

Figure 6-115 Selecting the options for the BPC client

4. A pop-up box indicates that the client has been generated and how to access the client. In this case the client is available on http://localhost:9080/OrderProcessHumanTaskClient. Click **OK** to continue (Figure 6-116).

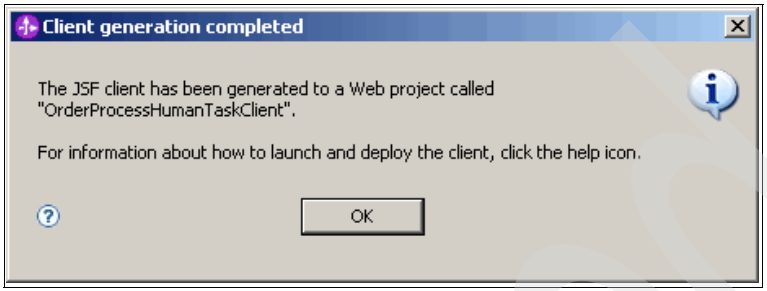

Figure 6-116 Confirmation that the client has been generated

5. The new client application must be added to the running WPS. This is done by clicking the **Servers** tab and right-clicking **WebSphere Process Server v6.1** → **Add and remove projects** as show in Figure 6-117.

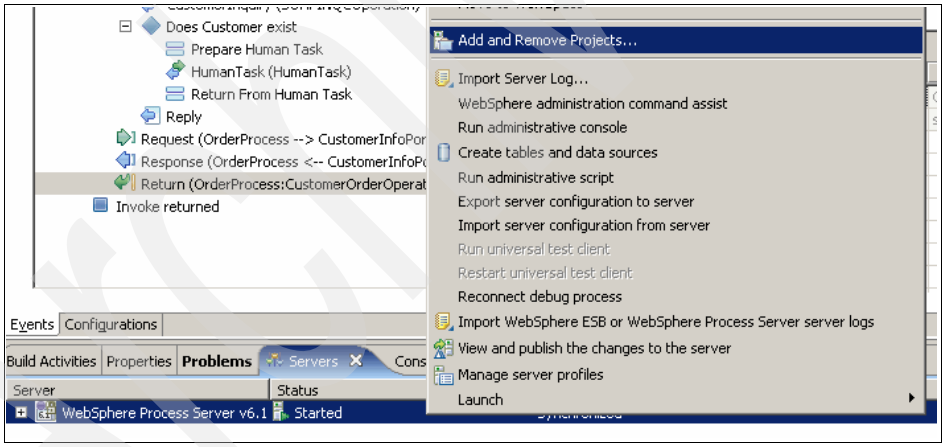

Figure 6-117 Adding and removing projects to the local WPS

6. On the Add and Remove Projects pop-up window select **OrderProcessHumanTaskClientEAR** and click **Add** to add the project to the server. Click **Finish** to continue (Figure 6-118).

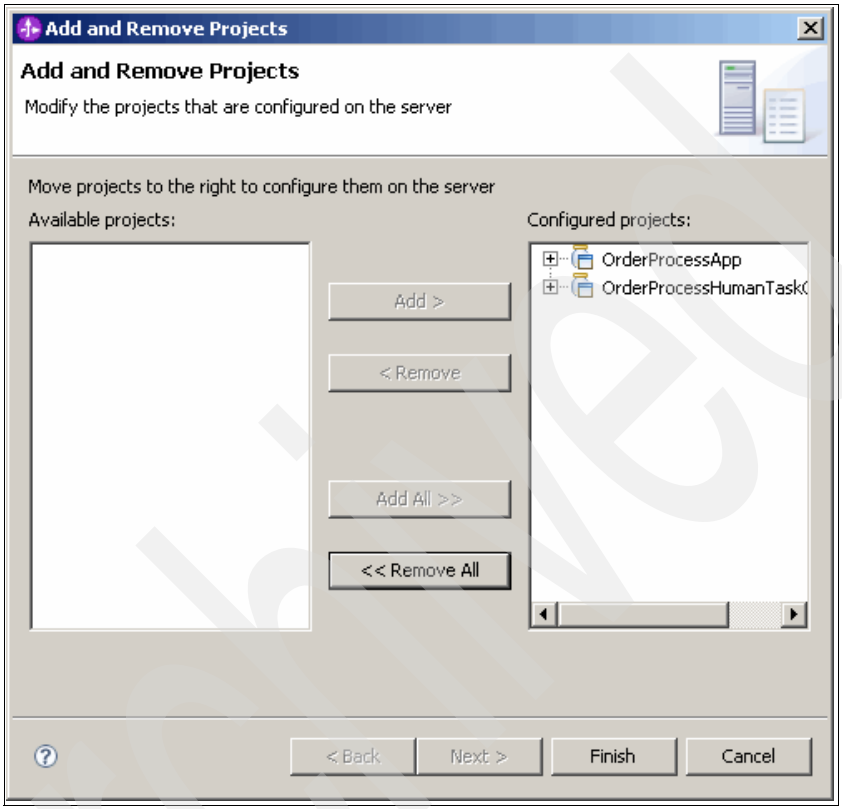

Figure 6-118 Adding the BPC client to the list of application deployed on WPS

7. If the WebSphere Process Server v6.1 is not started, right-click the server and click **Start**.

   Once the client application is added to the server and is running it can be tested within a Web browser. Open Internet Explorer® from the desktop, enter address `http://localhost:9080/OrderProcessHumanTaskClient` press Enter. When prompted for your user ID and password, enter `admin` / `admin`. A Web page will appear as show in Figure 6-119 on page 175.

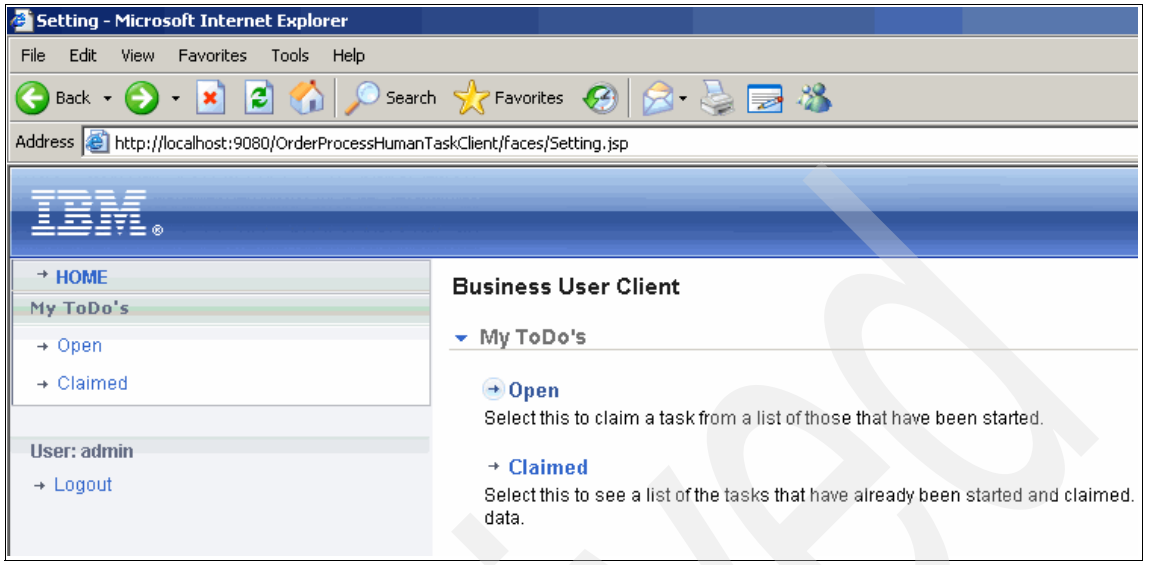

Figure 6-119 Entering the URL of the BPC client into a Web browser

8. To test the new client, follow steps 1 through 4 in 6.5.1, “Testing the Human Task with built-in client” on page 163, and simply rerun the previous test case.
9. Return to the Web page with the **OrderProcessHumanTaskClient** and select **My ToDo’s** → **Open.**
10. The next Web page shows a list of uncompleted tasks. Click the first uncompleted task as shown in Figure 6-120.

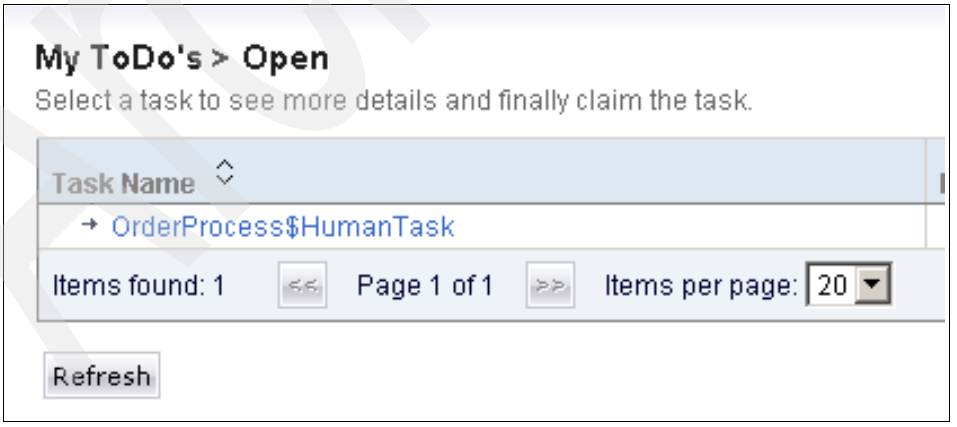

Figure 6-120 A list of uncompleted tasks for the user

11. In the next window click the **Claim** button to start working on the task.

12. Add a message to the “Message” box and click **Complete** (Figure 6-121).

Figure 6-121 Adding the message that will be returned and completing the task

13. The client is then returned to a list of tasks that are open and not completed. It will be empty in your case.

14. Return to the WID OrderProcess Event test frame and click the **Return** event. The message that was entered in the OrderProcessHumanTaskClient panel should appear in the Message field as shown in Figure 6-122.

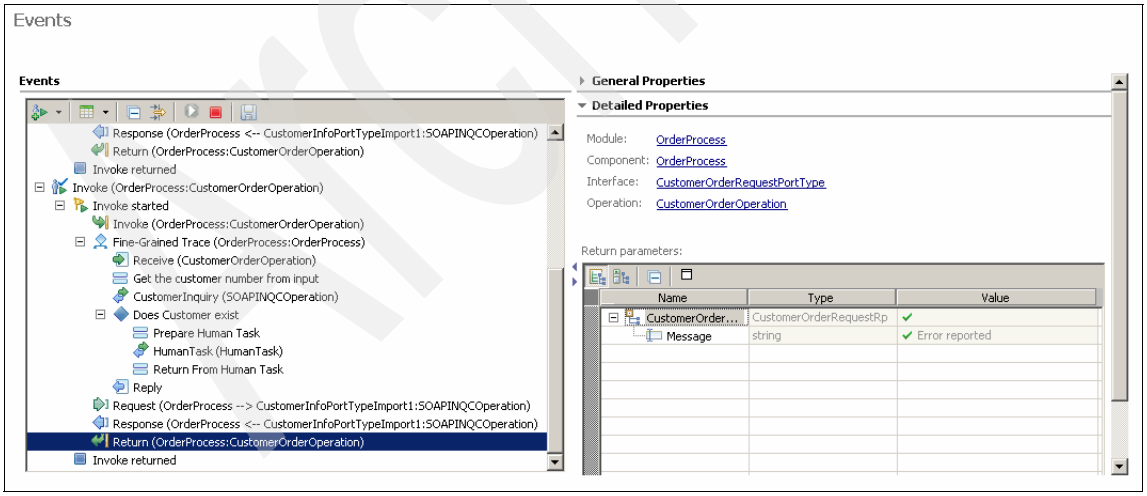

Figure 6-122 Checking the response from the process

15. Close the Order test window in WID without saving.

6.7 Creating a Web client front end

This section describes the steps necessary to quickly create a Web client and use the Web client to test. The client will be created using WID and can be easily modified. The project can then be deployed to WPS and used.

1. To generate a Web client from the WSDL that initiates the OrderProcess process, go to the Business Integration view and in the **OrderProcess** → **Web Service Ports** folder right-click **CustomerOrderRequestPortTypeExport1_CustomerOrderRequestPortTypeHttpPort** → **Generate client**. Select **Web Services** → **Generate Client** in the pop-up window (Figure 6-123).

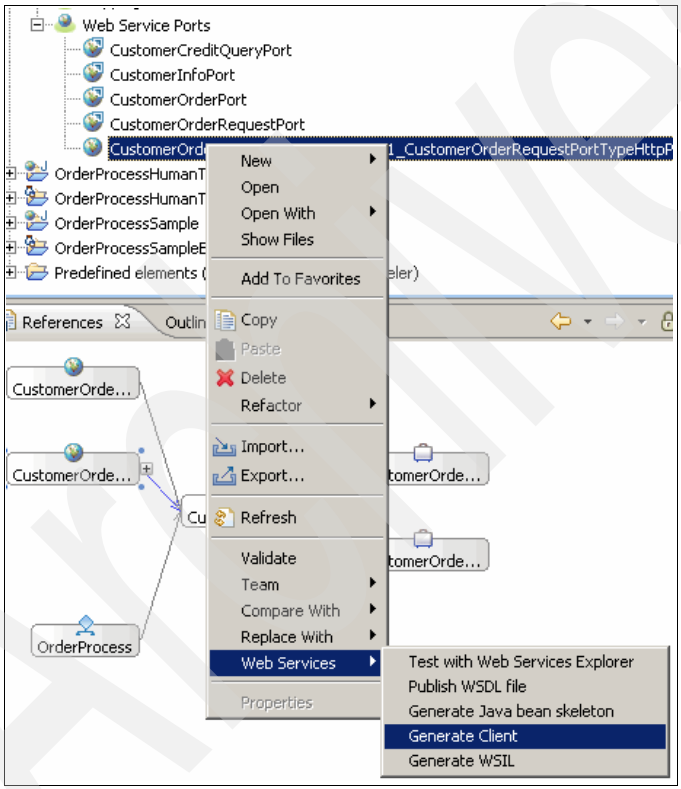

Figure 6-123 Generating a client

2. On the Web Services pop-up window make or verify the following settings, shown in Figure 6-124 on page 178:
   - Client type: **Java Proxy**
   - **Test Client** mode is selected (slide the vertical bar on the left of the window)

– Server: **WebSphere Process Server v6.1**
– Client project: **OrderProcess**

Click **Next**.

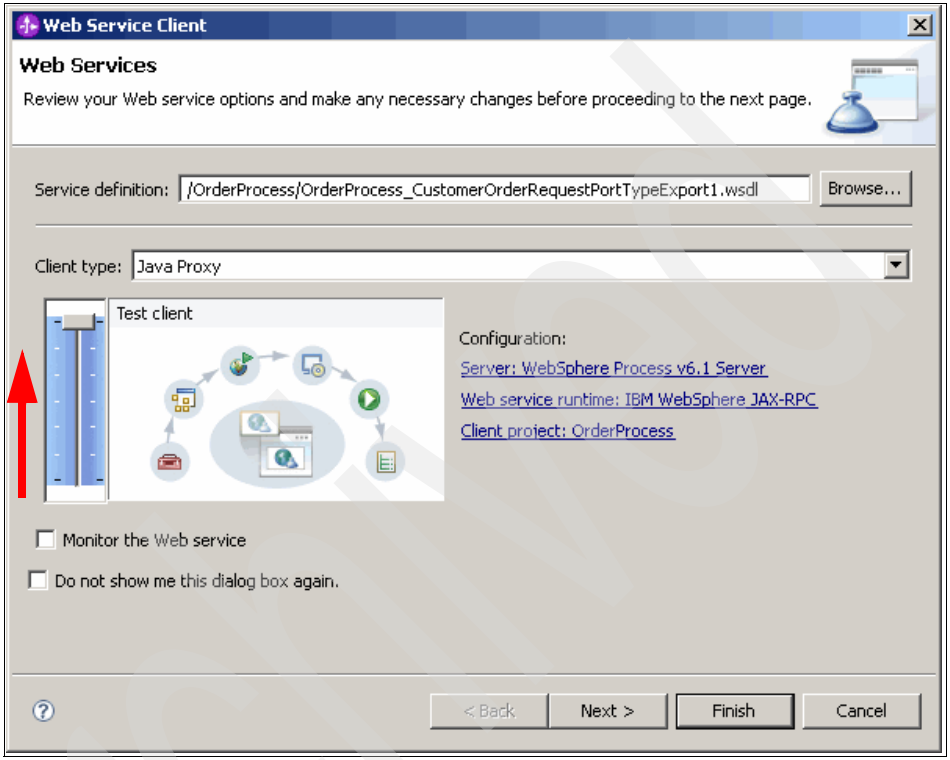

Figure 6-124 Adding the preferences for the Web client generation

3. On the next page accept the defaults and click **Next**, as shown in Figure 6-125. Security will not be enabled for this example.

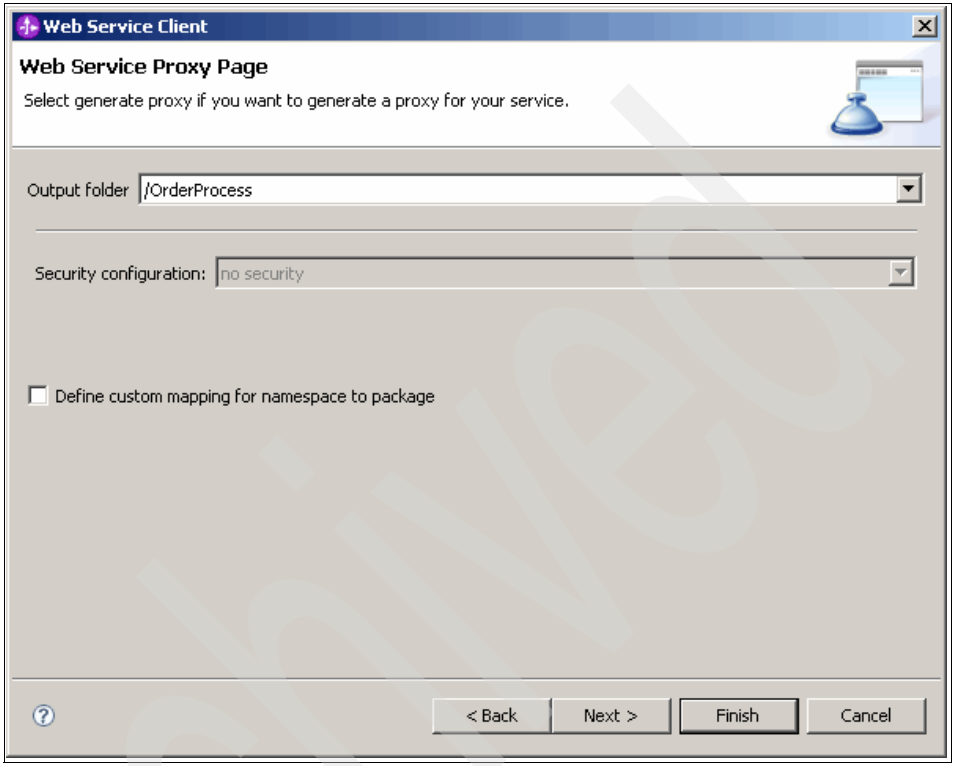

Figure 6-125 Accepting the defaults for the WSDL selected

4. In the Web Service Client Test window, shown in Figure 6-126 on page 180, verify the following settings:
   - **Test the generated proxy** is checked
   - All client **Methods** are checked
   - **Run test on server** is checked

   Click **Finish**. This will create a test client with all available methods accessible for the client.

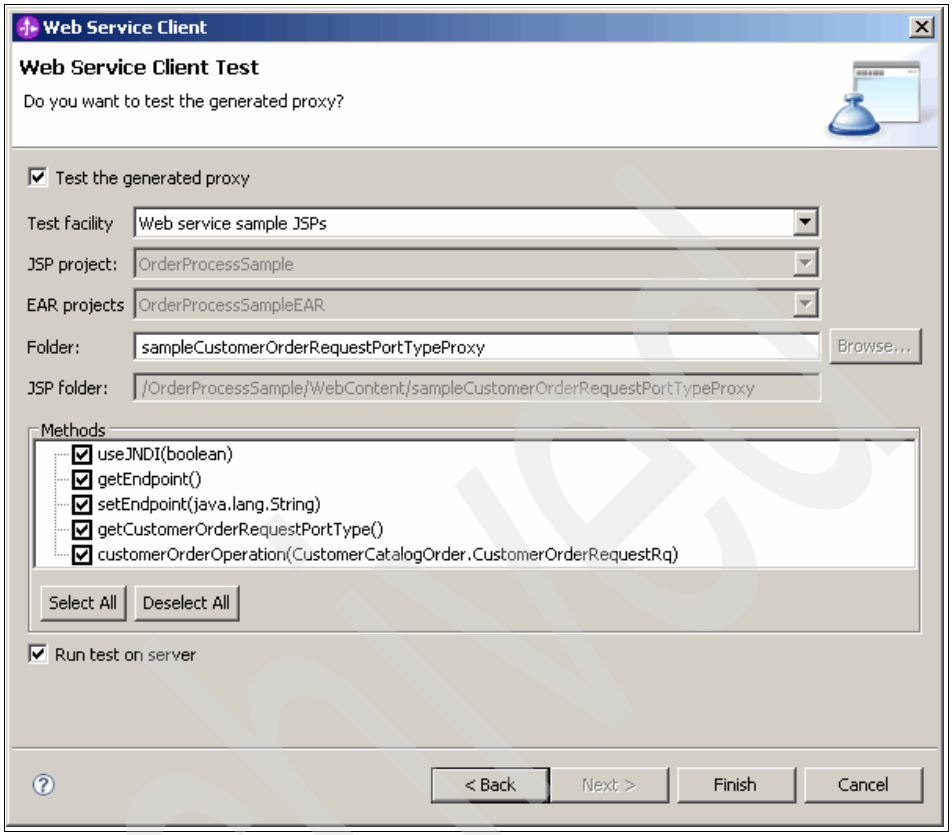

Figure 6-126 Setting the correct methods to be exposed as Web Services

5. After some time a Web page appears with the client showing as in Figure 6-127.

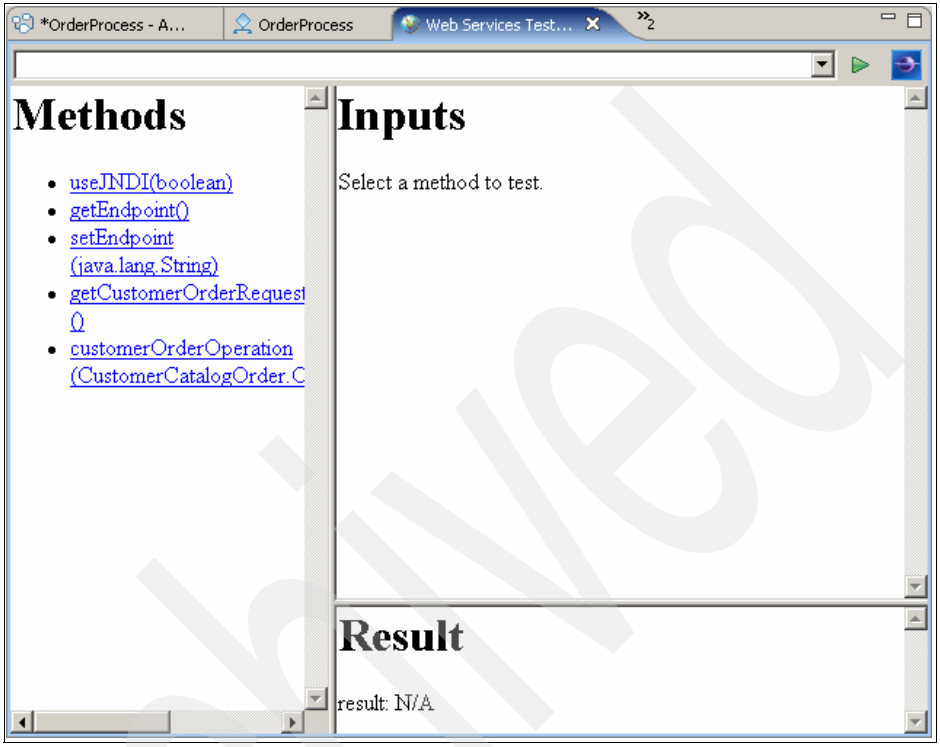

Figure 6-127 Web client Web page

6. Click **customerOrderOperation**, enter the request information, and click **Invoke** to send the request as shown in Figure 6-128. The request information can be found in 6.5.1, “Testing the Human Task with built-in client” on page 163.

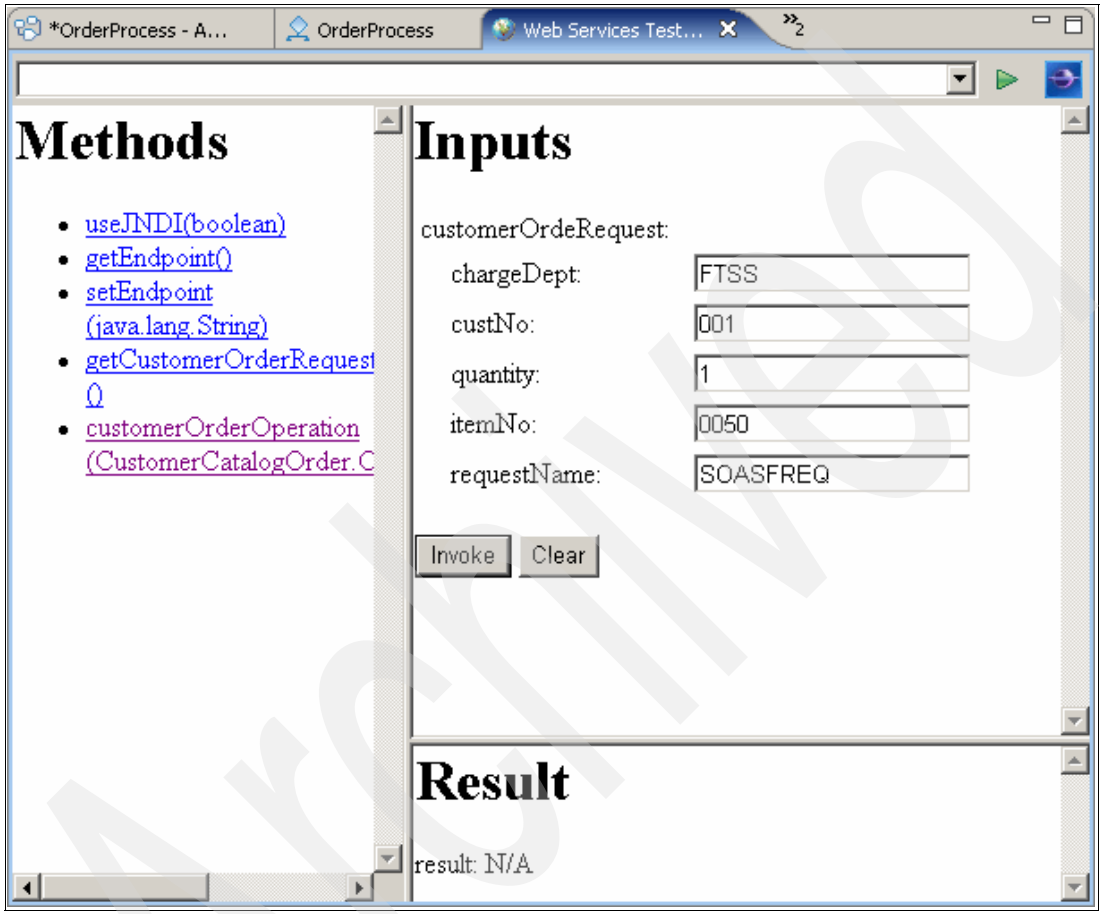

Figure 6-128 Enter the request information into the Web client

7. The reply should be returned in the **Result** frame as shown in Figure 6-129.

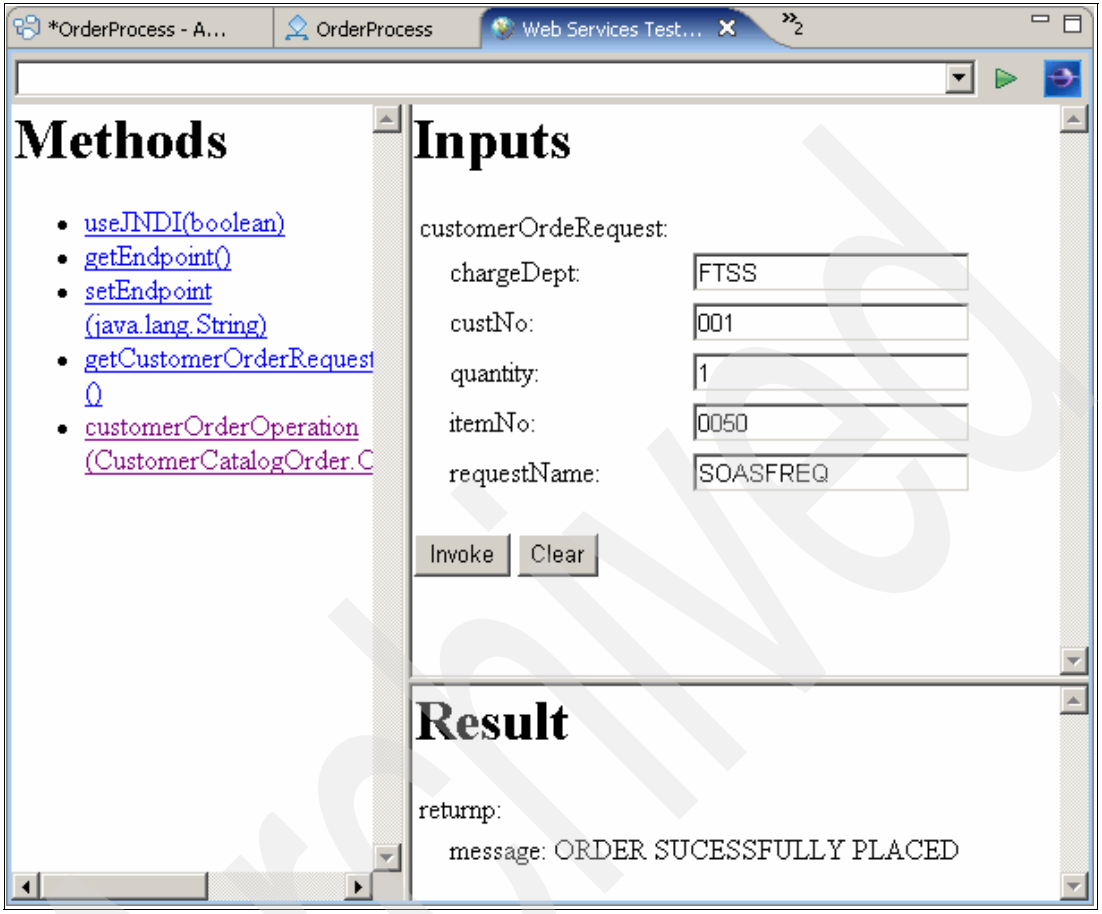

Figure 6-129 Result from the Web client request is returned

6.8 Additional material

A workspace is included with the entire solution. Refer to Appendix A, “Additional material” on page 229 for details.

7

Deploying and running the process in WPS on z/OS

In this chapter we describe how to deploy the finished process to WPS on z/OS and how to perform a simple test.

7.1 Deploying the business process to WPS on z/OS

Once the business process has been tested on the local WPS server on Windows® we are ready to deploy it to WPS on z/OS.

First, an ear file containing the project must be exported from WID. Perform the following steps:

1. Switch to the Business Integration Perspective.
2. Right-click the project name and select **Export**.
3. From the pop-up window, select **J2EE** → **EAR file**, as shown in Figure 7-1.

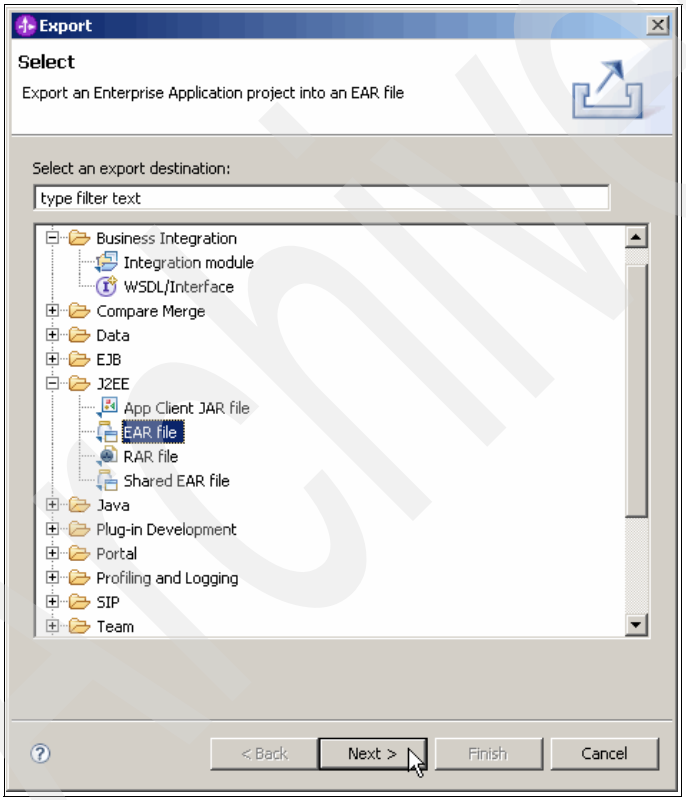

Figure 7-1 Exporting an EAR file

4. Select the destination for the EAR file.

Important: Before exporting, decide what security you have configured in WebSphere Process Server. The application that you export must match the target security. Configure the human tasks to use the security of the target server, that is, set the JNDI name of the staff plug-in configuration in the Properties view Details tab. In our scenario, security is not enabled.

7.1.1 Exporting the Human Task client application

In “Creating a Human Task client” on page 170 we described how to create a client application to manage the Human Tasks. We called that application OrderProcessHumanTaskClient. If you wish to test the Human Task with this client application, you must deploy it, too. The application contains a war file only and does not use any JDBC™ or JMS references.

Export the war file from WID and deploy it to WPS, accepting all defaults.

7.2 Our WPS configuration

The process server must be configured with certain resources to be able to run the Order process. Those resources include a JDBC resource and JMS resources.

7.2.1 Defining the JDBC resource

The data source used for our process is based on a DB2 database, with DB2 Location name DB9H. It is listening on port 38130 on host wtscz1.itso.ibm.com. See Figure 7-2 for the details of our data source.

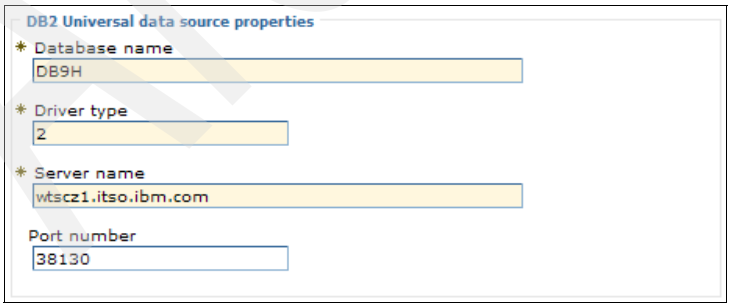

Figure 7-2 DB2 data source

You need to have a data source specified for your database of choice with the appropriate scope: Server, Node, or cell. We specified our data source at the Node level and selected the DB2 Universal JDBC Driver Provider. See Figure 7-3 for a list of data sources in our environment.

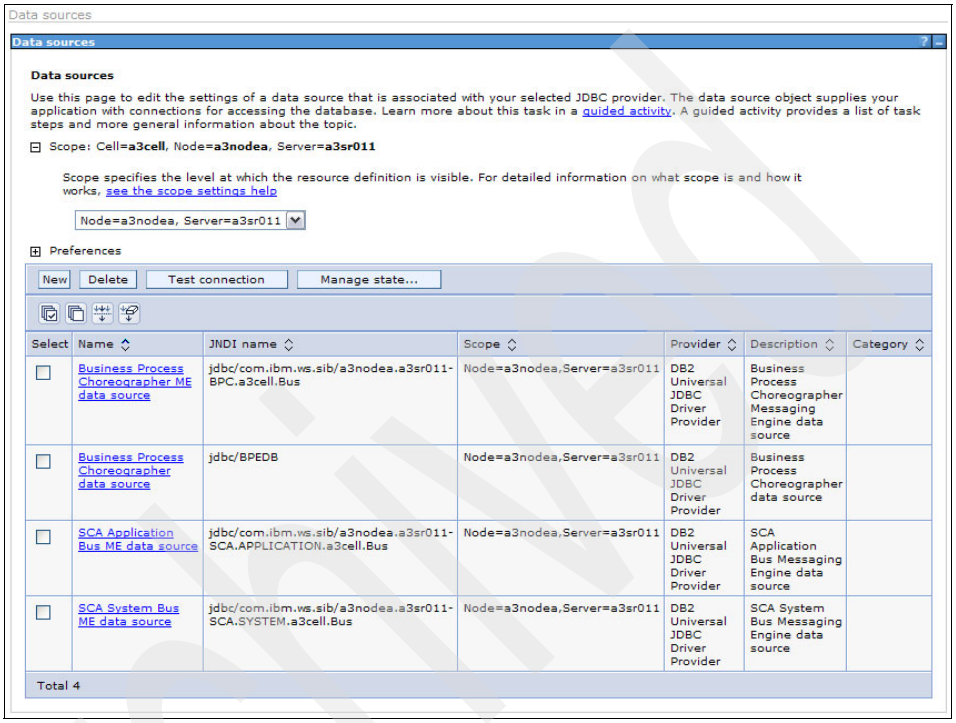

Figure 7-3 Data sources in our WPS server

7.2.2 Defining the JMS resources

While building the Order process application in WID, EJBs have been generated and those EJBs access the SIBus in WebSphere Application Server. Internal mediation as a result of the process flow is executed over the SIBus. For this reason, JMS destinations are required to define the access points on the SIBus.

As shown in Figure 7-4, there are two JMS Connection Factory references specified in the deployment descriptor of the EJB module of the Order process application.

```
<resource-ref id="ResourceRef_1214336070000" >
        <res-ref-name> jms/BPECFC</res-ref-name>
        <res-type> javax.jms.ConnectionFactory</res-type>
        <res-auth> Application</res-auth>
        <res-sharing-scope> Shareable</res-sharing-scope>
</resource-ref>
<resource-ref id="ResourceRef_1214336070015" >
        <res-ref-name> jms/BPECF</res-ref-name>
        <res-type> javax.jms.ConnectionFactory</res-type>
        <res-auth> Application</res-auth>
        <res-sharing-scope> Shareable</res-sharing-scope>
```

Figure 7-4 JMS references for Order process

Those JMS Connection factories must be specified first using the Integrated Solutions Console, under option **Resources** → **JMS** → **Connection Factories**. The provider type must be **Default Messaging Provider**.

Our JMS Connection factories are specified as shown in Figure 7-5 on page 190 and Figure 7-6 on page 191.

Connection factories > BPECFC

A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Configuration

General Properties

Administration

Scope
Node=a3nodea,Server=a3sr011

Provider
Default messaging provider

* Name
BPECFC

* JNDI name
jms/BPECFC

Description

Category

Connection

* Bus name
BPC.a3cell.Bus

Target
a3nodea.a3sr011

Target type
Bus member name

Target significance
Preferred

Target inbound transport chain

Provider endpoints

Connection proximity
Bus

Additional Properties

- Connection pool properties

Related Items

- JAAS - J2C authentication data
- Buses

Figure 7-5 JMS Connection Factory BPECFC

Connection factories > BPECF

A JMS connection factory is used to create connections to the associated JMS provider of JMS destinations, for both point-to-point and publish/subscribe messaging. Use connection factory administrative objects to manage JMS connection factories for the default messaging provider.

Configuration

General Properties

Administration

Scope
Node=a3nodea,Server=a3sr011

Provider
Default messaging provider

* Name
BPECF

* JNDI name
jms/BPECF

Description

Category

Connection

* Bus name
BPC.a3cell.Bus

Target
a3nodea.a3sr011

Target type
Bus member name

Target significance
Preferred

Target inbound transport chain

Provider endpoints

Connection proximity
Bus

Additional Properties

- Connection pool properties

Related Items

- JAAS - J2C authentication data
- Buses

Figure 7-6 JMS Connection factory BPECF

7.3 Deploying the Order process application to WPS

Enterprise applications can be installed using the WebSphere Integrated Console or a wsadmin script. The steps to install the application using the WebSphere Integrated Console are as follows:

1. Select **Applications** → **Install application**.
2. In the next window, click **Browse** to locate the EAR file.
3. In the subsequent panels, we basically accept the defaults to prepare for and perform the application installation.
   a. Select installation options (select **Pre-compile JSP**).
   b. Map modules to servers.
   c. Provide listener bindings for message-driven beans.
   d. Provide JNDI names for Beans.
   e. Map resource references to resources.
   f. Map virtual hosts for Web modules.
   g. Ensure that all unprotected 2.x methods have the correct level of protection.
   h. Summary.
4. Click **Finish** and the application is installed. Wait for the message “Click Save to Master Configuration” and click **Save**.

Starting the application

After installation, the application will be in stopped status. Start the application by selecting **Applications** → **Enterprise Applications**. Check the Order process application and click **Start**.

7.4 Testing on z/OS

Once the Order process application is deployed and started, you can start testing. Note that in addition to a deployed WPS Order process application, you also need the following back-end components installed and working:

- Customer inquiry service installed into CICS.
- Order placement service flow installed into CICS.
- Message flow in WebSphere Message Broker, as described in Chapter 4, “Credit check service” on page 55.
- DB2 stored procedure for the credit check installed into DB2.

You can use other back-end services of your choice and adjust the Order process and user interface, as long as the mappings and service bindings match your back-end services.

1. Use or start a new browser and enter the URL of the OrderProcess application, as shown in Figure 7-7. Use the URL of your own WPS server.

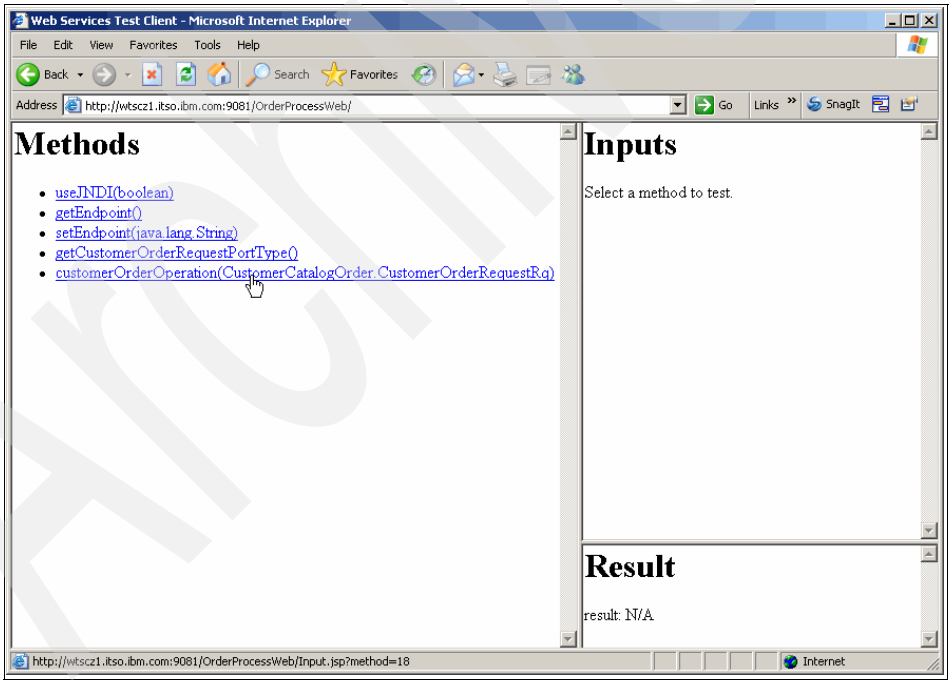

Figure 7-7 Starting the OrderProcess client

2. Click **customerOrderOperation** under Methods. The window shown in Figure 7-8 will appear.

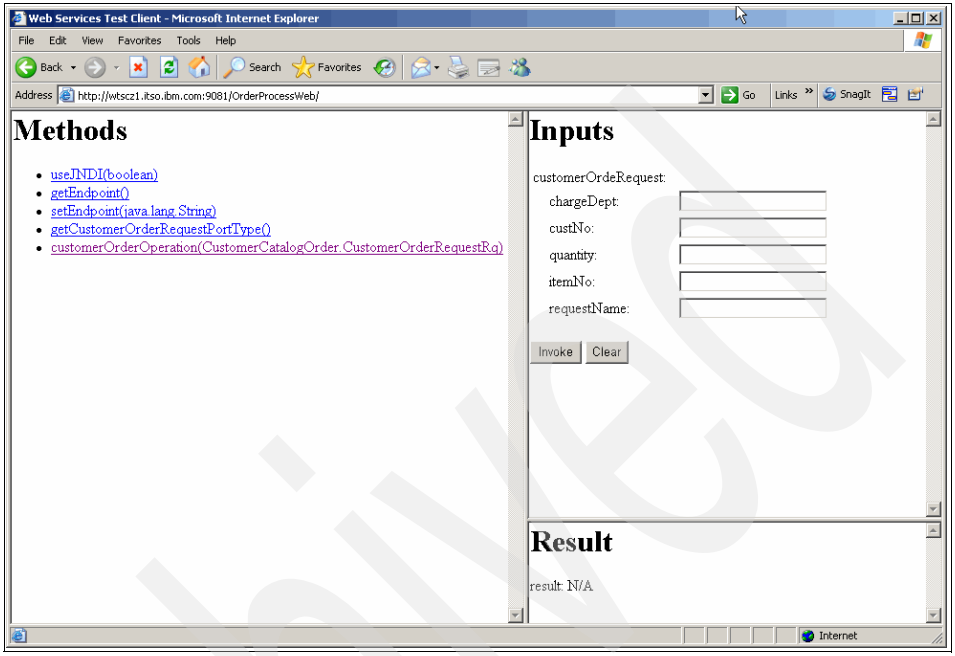

Figure 7-8 Input window

3. Enter values in the **Inputs** pane, as shown in Figure 7-9. Click **Invoke**.

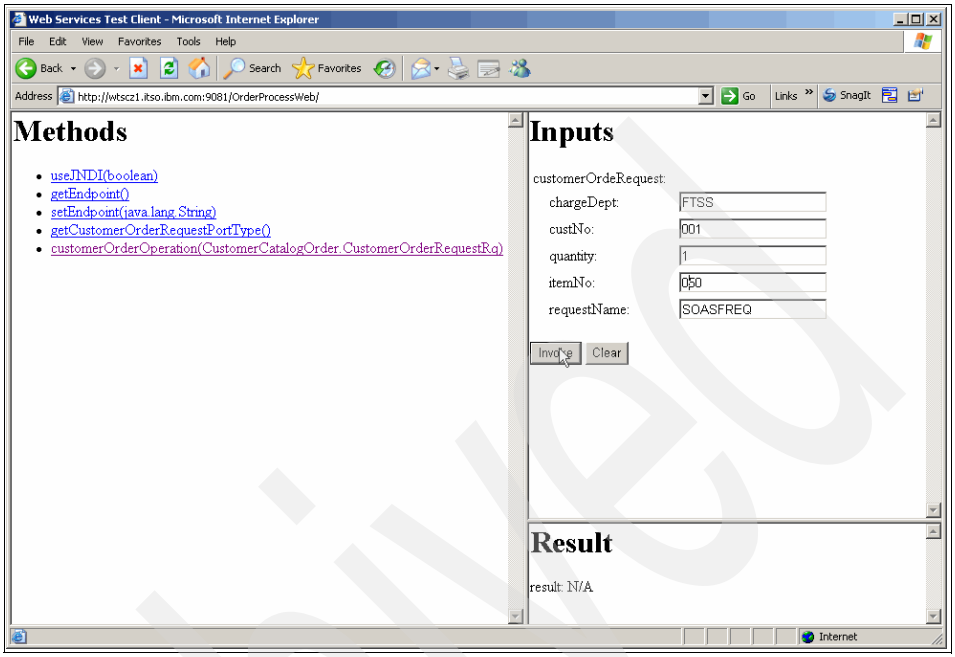

Figure 7-9 Entering input values

The next window shows the result of your order (Figure 7-10).

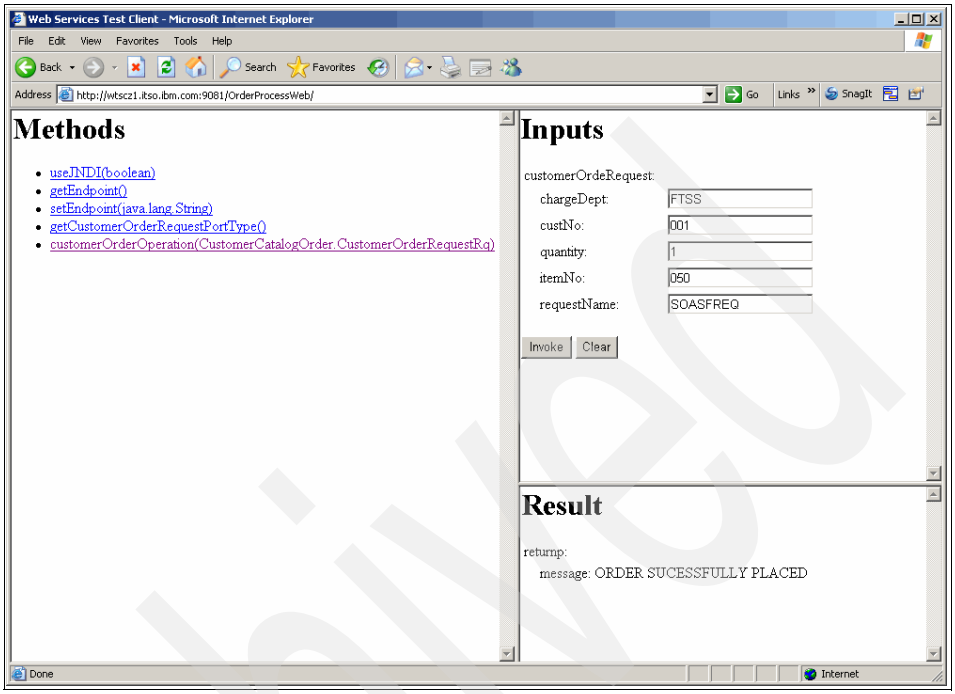

Figure 7-10 Order successfully placed

4. For the next test use a customer number that does not exist. As you may remember, a Human Task is modeled for this situation and a task will show up for the user to resolve this situation. Use customer number **077** in this case, as shown in Figure 7-11. Click **Invoke** again.

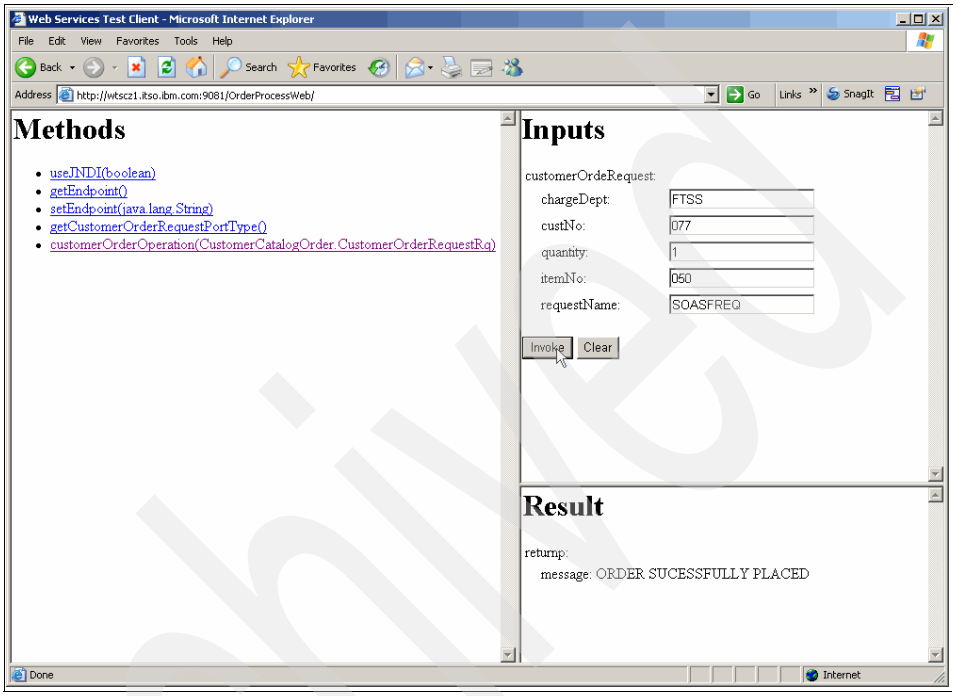

Figure 7-11 Input values with an invalid customer number

The result should look as shown in Figure 7-12.

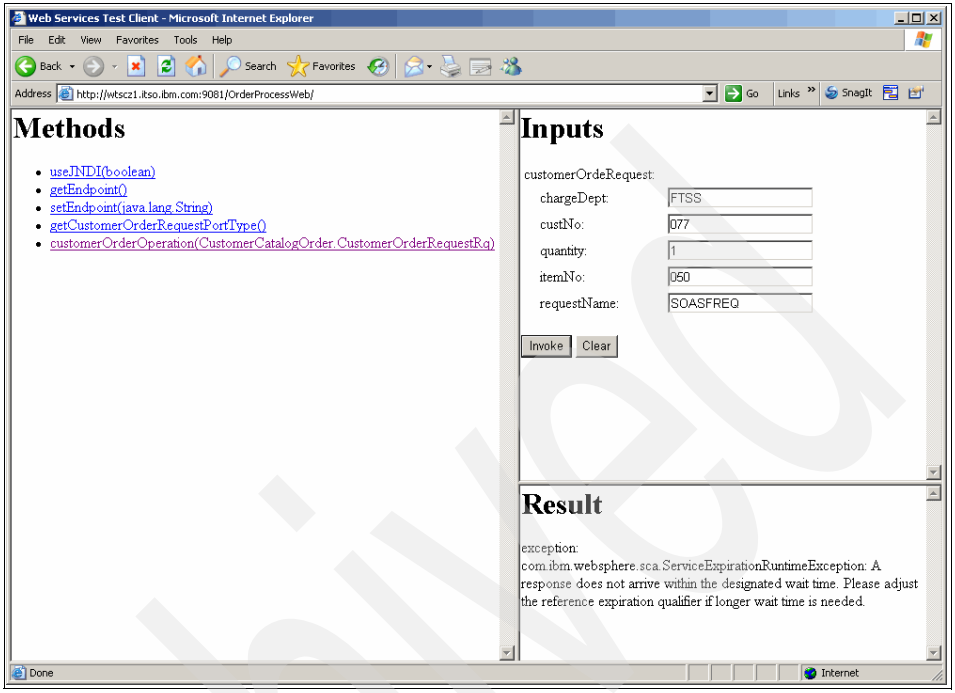

Figure 7-12 Invalid customer message

You will see an exception indicating that the process has timed out. The reason for this is that there is a Human Task that needs to be accomplished.

5. Start another browser with the Human Task client, as shown in Figure 7-13, and using the URL adjusted to your local setup.

Note: You can only execute this activity if you have previously deployed the Human Task client application WAR file.

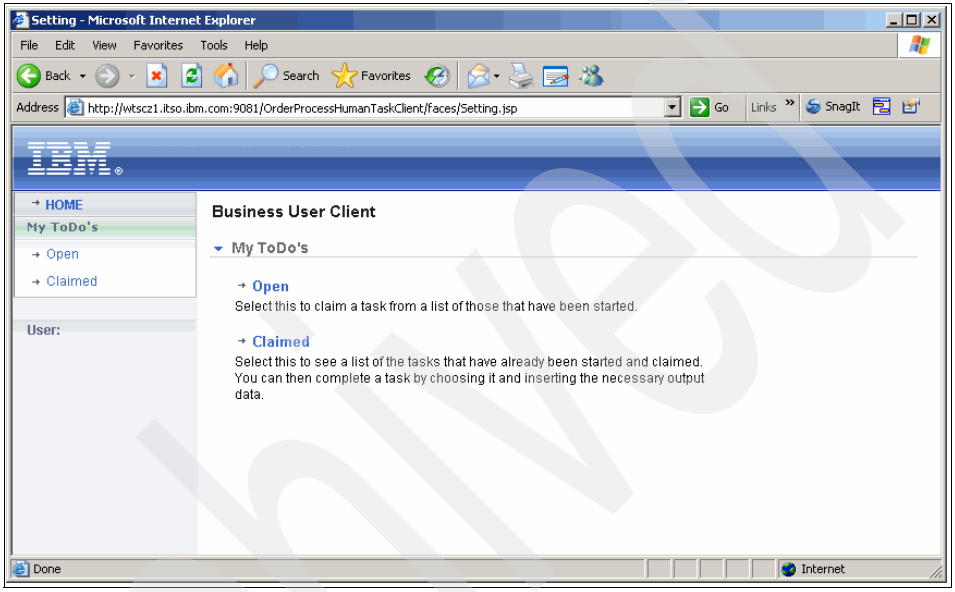

Figure 7-13 Initial BPC client window

6. Click **My ToDo's** → **Open**. This will return the window shown in Figure 7-14.

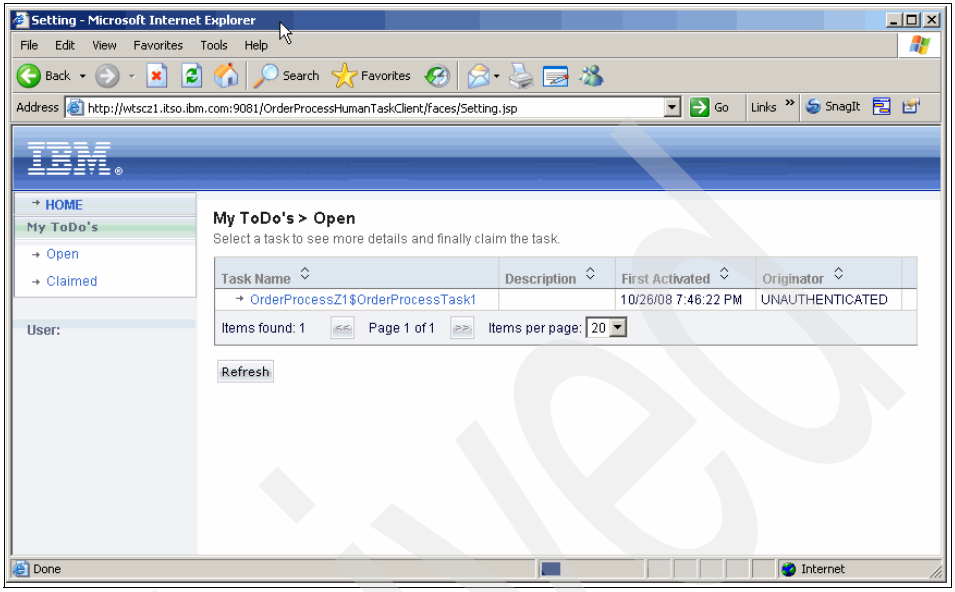

Figure 7-14 Task to be completed

7. Click the highlighted task name. The next window will look like Figure 7-15.

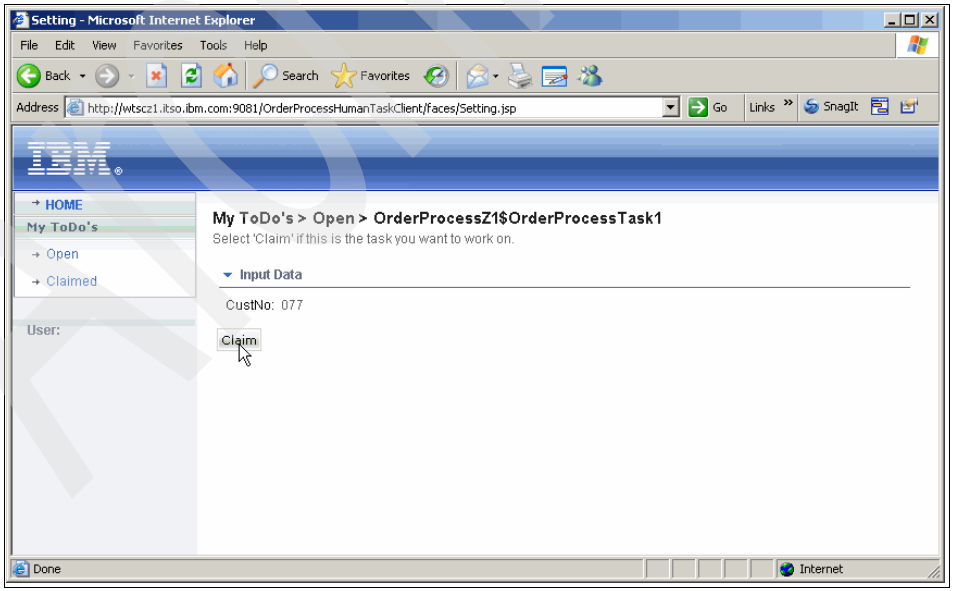

Figure 7-15 Claiming the task

8. Click **Claim**, type some message text, and click **Complete**, as shown in Figure 7-16.

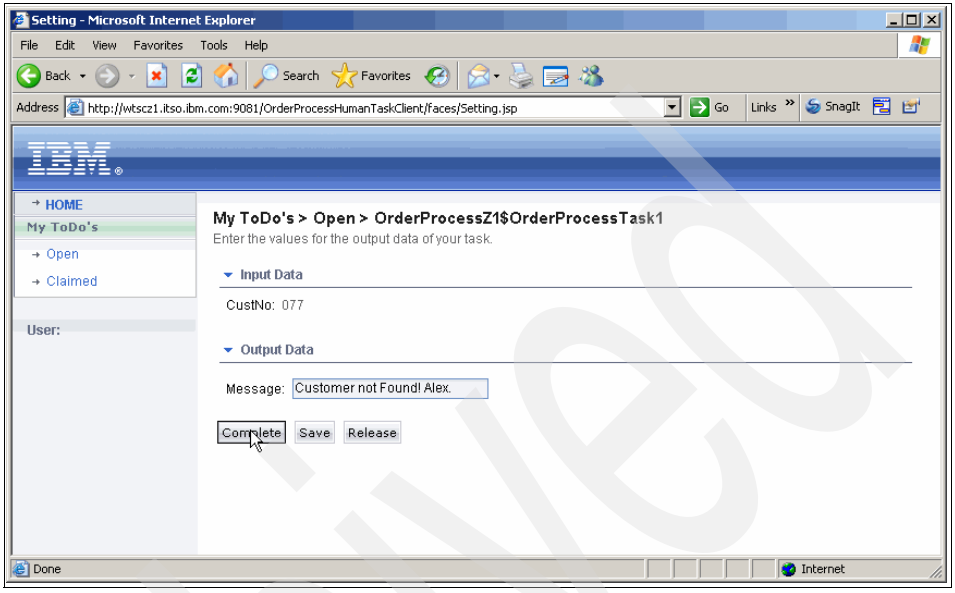

Figure 7-16 Completing a Human Task

The task should now be complete. Close the browser window.

9. If you wish, you can repeat this test scenario using the Business Process Choreographer Explorer. The URL should be of this format:

```
http://<your ICP/IP address>:<your portnumber>/bpc/
```

Repeat step 4 and then use the BPCE to complete the Human Task, using the information in “Testing the Human Task with built-in client” on page 163.

Important: If you claim and complete the Human Task within a reasonable time, the time-out exception in the OrderProcess will not occur and you will see the response message coming back that you typed in when claiming the Human Task. If you wait too long with claiming the Human Task, the OrderProcess browser window will time out.

8

Adding a Portal user interface

Now that the Order process is modeled and all back-end services are in place, we can create a simple Portal user interface. We have already seen in “Creating a Web client front end” on page 177 that it is not difficult to generate a Web-based user interface. Creating a portlet is not that much different. We will use the WSDL file that describes the Web Services interface of the Order process that we implemented in WPS. We describe the following tasks:

- Developing a Portlet user interface for the Order process
- Deploying the Web service Portlet

Important: If you construct the scenario in your own environment, be aware that you might need to choose you own naming conventions, as well as use your own variables for TCP/IP addresses, port number, host names, and so on. Pay special attention to keeping these consistent. The names and variables we use in this publication are only meant as examples.

8.1 Developing a Portlet user interface for the Order process

In this section we describe the steps we used to develop a Portlet user interface for the Order process using Rational Application Developer (RAD).

8.1.1 Creating a Project in RAD

1. Start Rational Application Developer Version 7. From the Windows Start menu, select **All Programs** → **IBM Software Development Platform** → **IBM Rational Application Developer** → **IBM Rational Application Developer**.
2. Enter a name for the Workspace as shown in Figure 8-1 (use **Browse** if needed) and click **OK**.

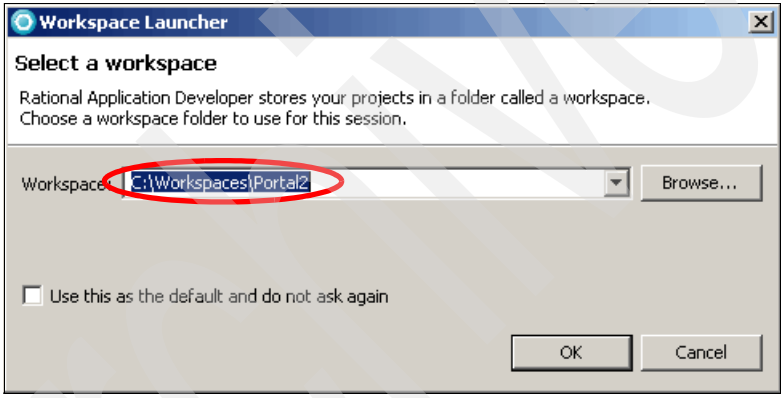

Figure 8-1 Select a Workspace window

Rational Application Developer should open up in the **Web** perspective. If this is not the case, perform the following steps to open the Web perspective:

a. From **Rational Application Developer**, select **Window** → **Open Perspective** → **Other**.

b. Check **Show All**.

c. Choose **Web** and click **OK**, as shown in Figure 8-2 on page 205.

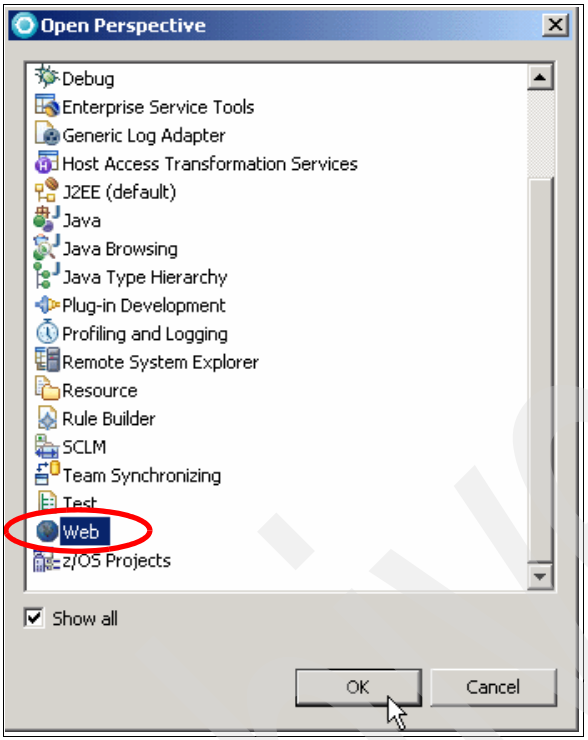

Figure 8-2 Select Web perspective window

3. Create a new project. From the **RAD** main menu, select **File** → **New** → **Project** as shown in Figure 8-3.

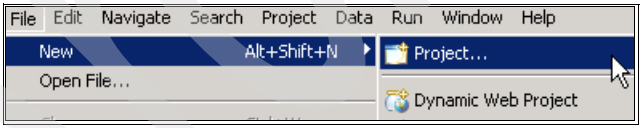

Figure 8-3 RAD Main menu: New Project

4. On the Select a wizard screen, select **Portlet Project** and click **Next** (Figure 8-4).

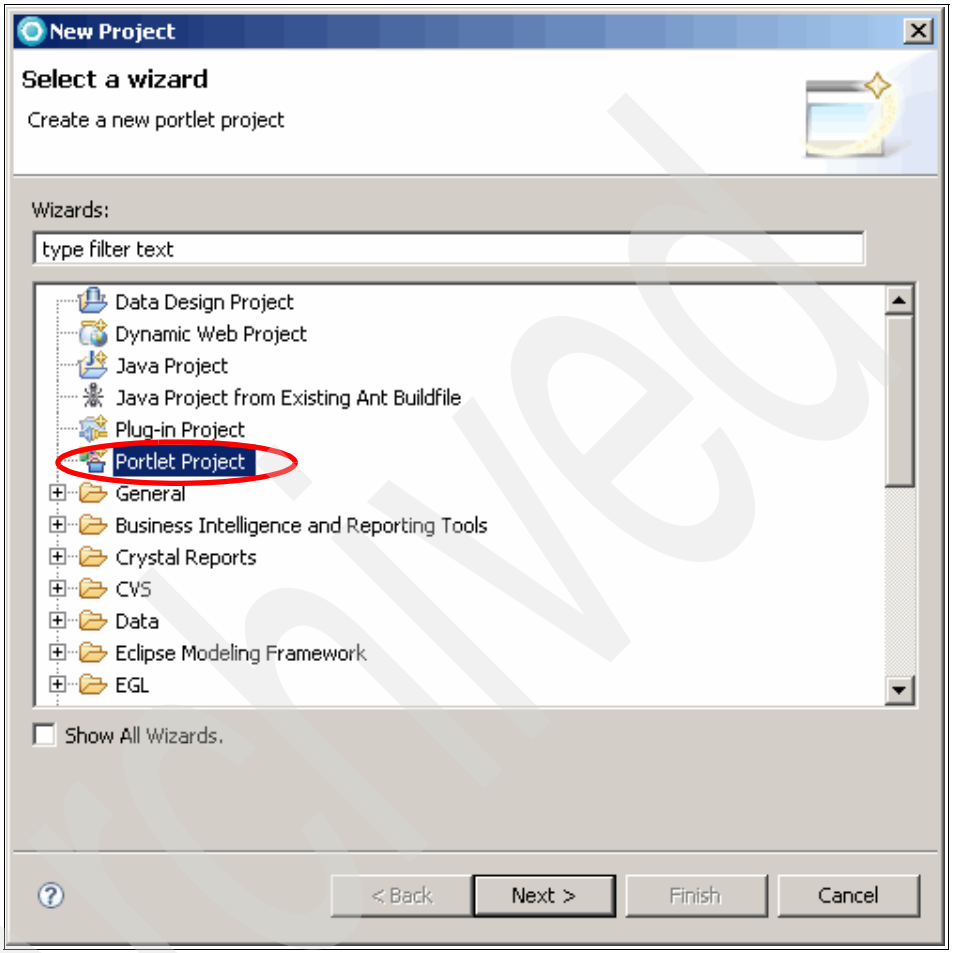

Figure 8-4 Selecting Portlet Project

5. If the Confirm Enablement screen appears, click **OK** to enable Portlet Development (Figure 8-5).

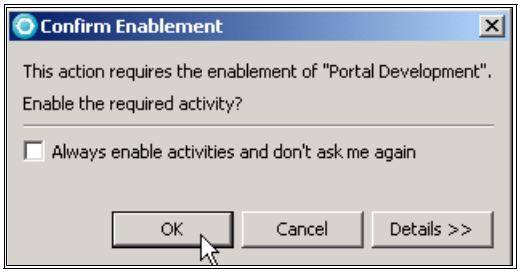

Figure 8-5 Enable Portal Tools

6. On the Portlet Project panel, shown in Figure 8-6 on page 208, make the following entries and selections:
   - Project name: Enter a name of your choice.
   - Target Runtime: Select **WebSphere Portal v6.0 stub** from the drop-down list because we will deploy to a Portal V6.0 server on z/OS.
   - EAR Membership: Unselected.
   - Portlet API: Select **IBM Portlet**.
   - All other values will be defaulted based on the project name entered. Make sure all check boxes and other settings are as shown in Figure 8-6.

   Click **Next**.

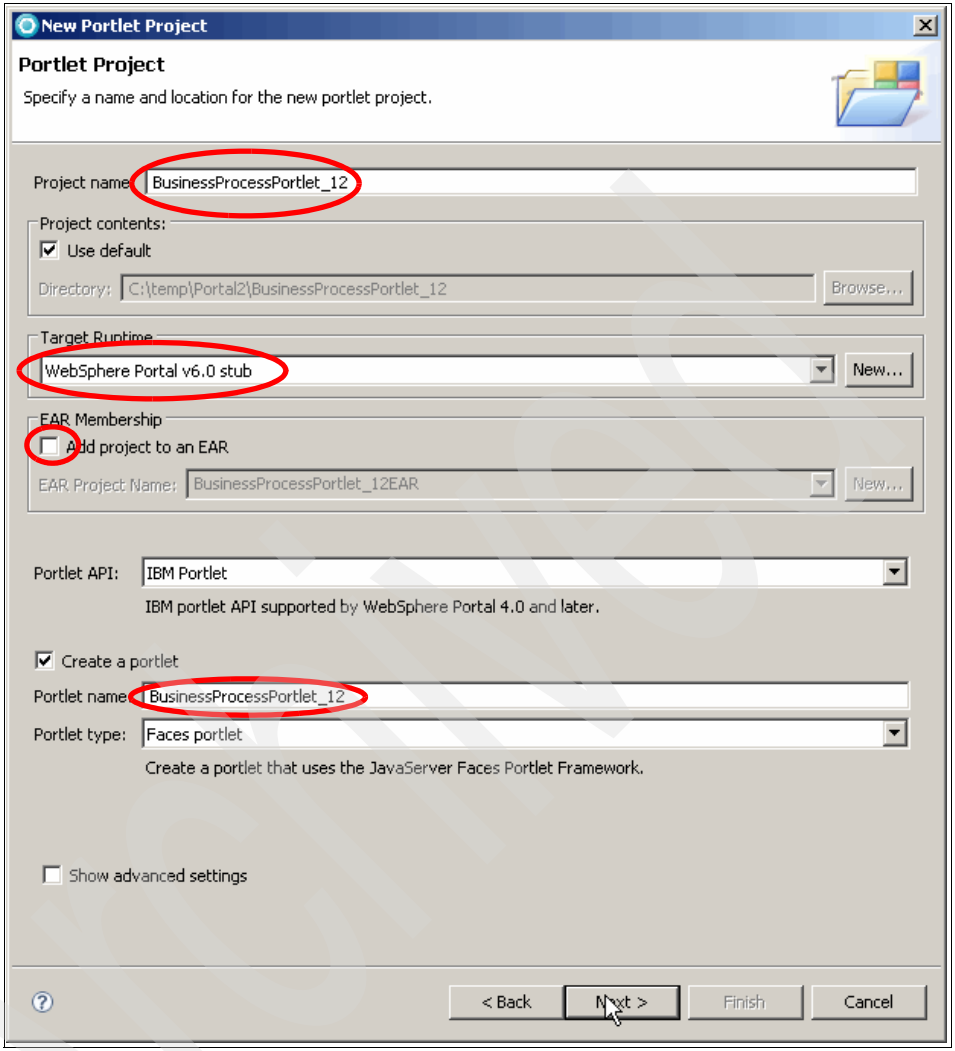

Figure 8-6 Entering the Portlet Project values

7. On the Portlet Settings panel (Figure 8-7), keep all default values and click **Finish.**

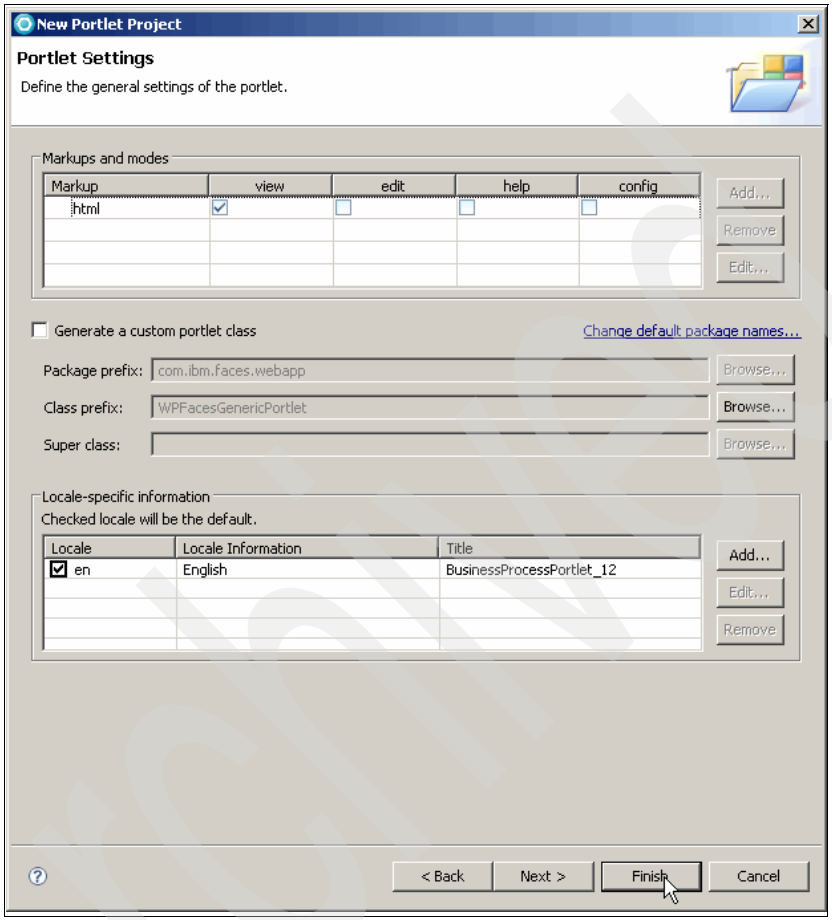

Figure 8-7 Portlet Settings panel

8. If you receive the Open Associated Perspective window (Figure 8-8), click **Yes** to open the Web perspective now.

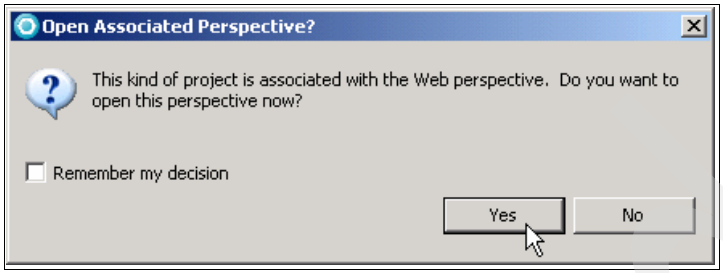

Figure 8-8 Opening Web perspective now

You should now see your newly created project in the **Project Explorer** tab, as shown in Figure 8-9.

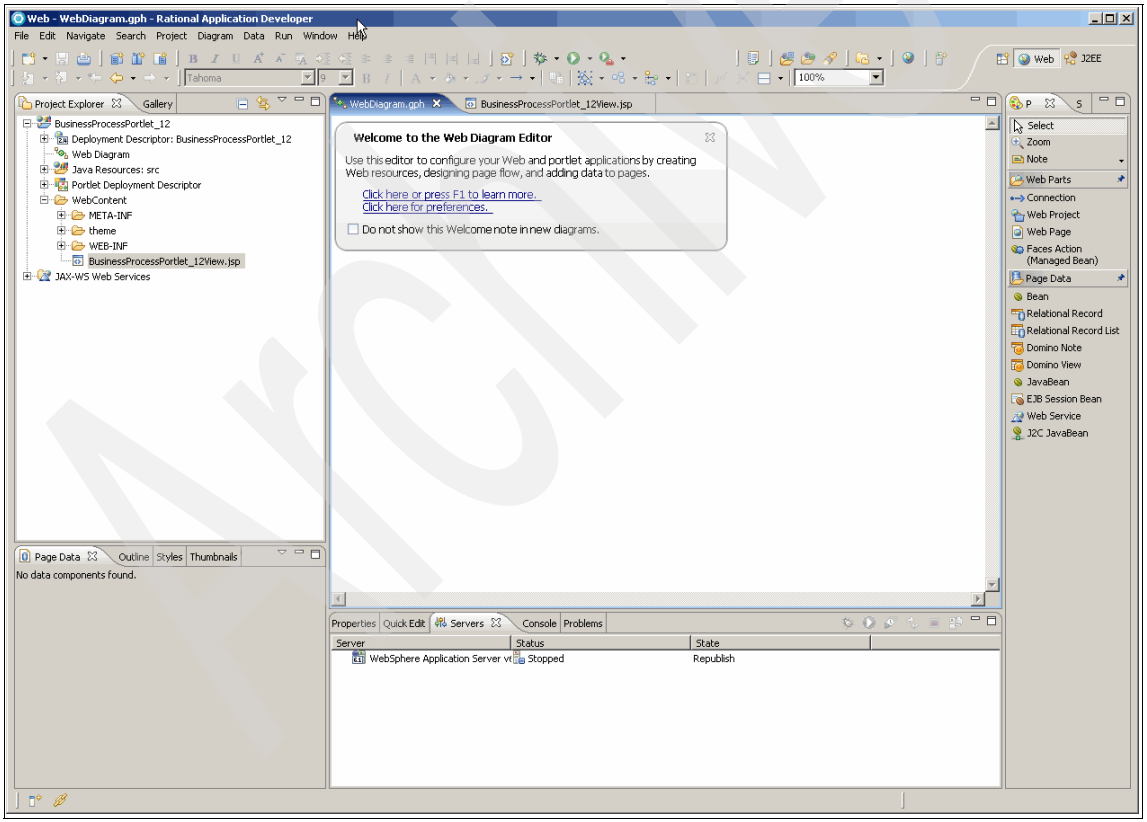

Figure 8-9 Closing webDiagram.gph

9. When the Web Diagram Editor window appears, click **x** on the **WebDiagram.gph** tab to bring the window with the generated JSP to the foreground.

8.1.2 Importing the WSDL of the Order process

Now we import the WSDL of the Order process. In our example this was named **OrderProcess_CustomerOrderRequestPortTypeExport1.wsdl**. The WSDL represents the operations and parameters that can be used to drive the Order process from our new Portlet.

1. In the **Project Explorer**, expand the created Portlet, right-click the **WebContent** folder, and select **Import** as shown in Figure 8-10.

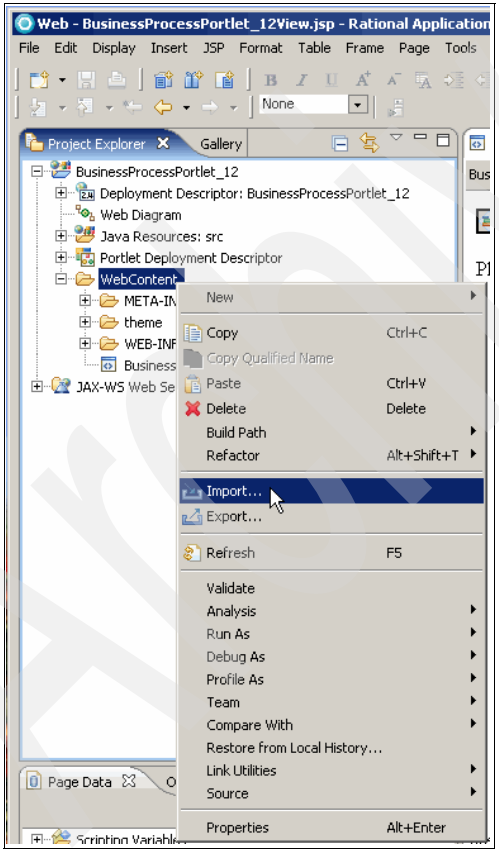

Figure 8-10 Selecting Import for the purpose of importing a WSDL

2. On the next screen expand **General**, select **File System**, and click **Next**, as shown in Figure 8-11.

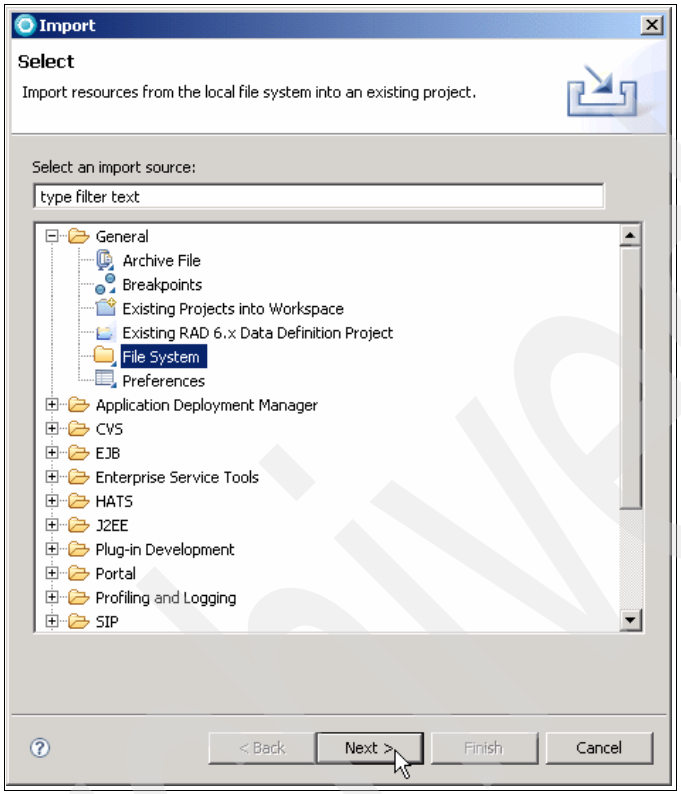

Figure 8-11 Import from the File System

3. On the **File System** screen, enter the location of the WSDL by using the **Browse** button and selecting the corret entry. Click **Finish** (Figure 8-12 on page 213).

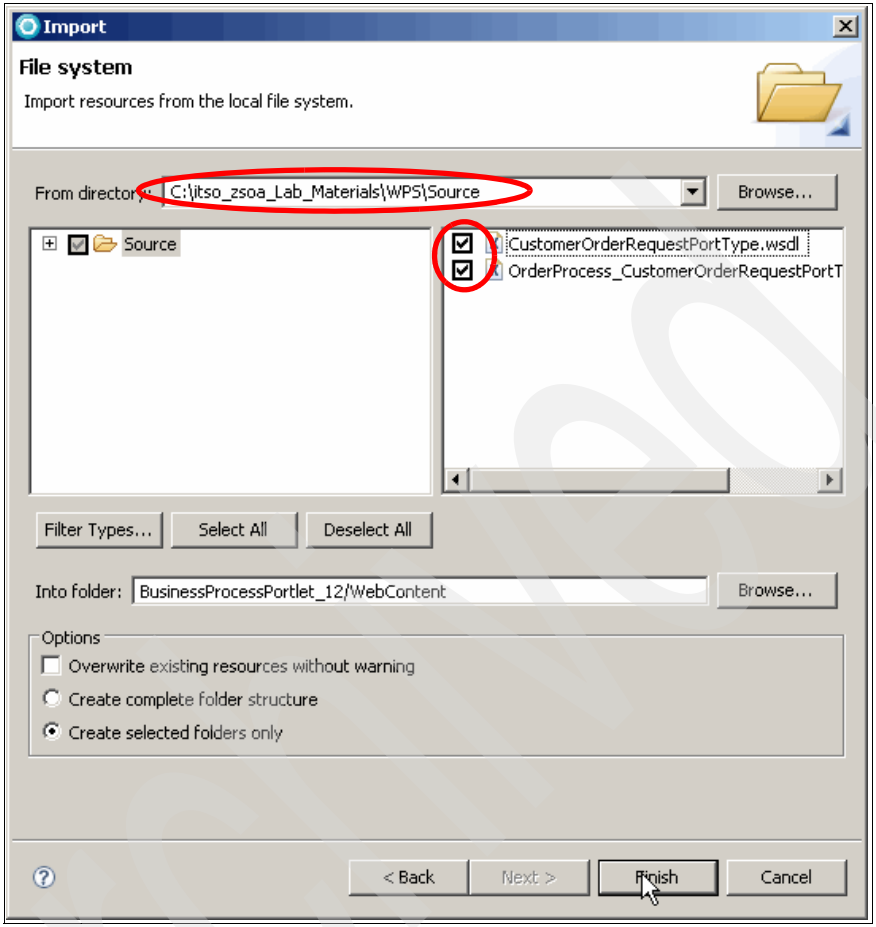

Figure 8-12 Selecting the WSDL to import into project workspace

Figure 8-13 shows the imported WSDL file in the Project Explorer.

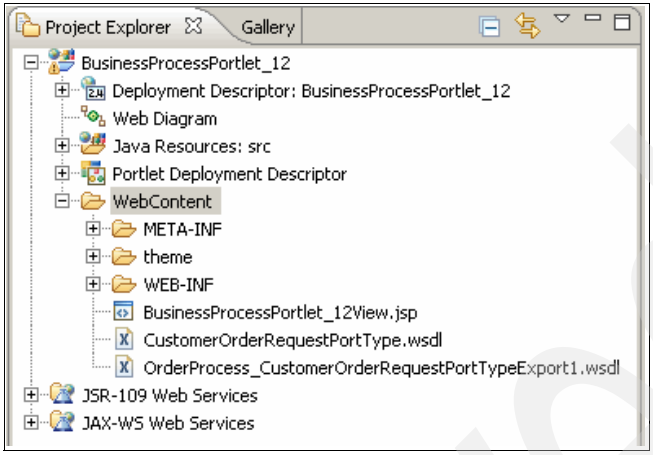

Figure 8-13 Imported WSDL in the Project Explorer

8.1.3 Adding the Web service to a JSP

We are now ready to modify the generated Business Process client JSP. Using the Page Designer in Design mode, we will add a Web Service to our *Java Server Face (JSF)*.

1. In the Page Editor, rename Place Content Here to `Order Details` by overtyping it.
2. From the **Palette tab** located on the right of your screen, expand **Data,** and drag **Web Service** onto the JSP page under **Order Details**.

3. On the “Web Service Discovery Home” screen, click **Web services from your workspace** as shown in Figure 8-14 on page 215.

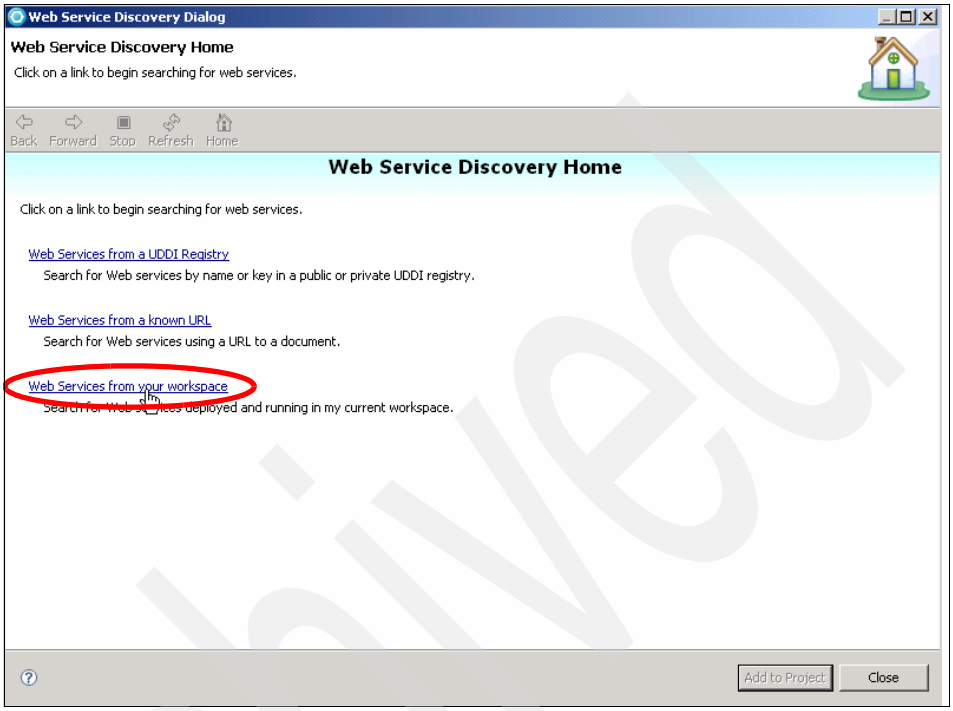

Figure 8-14 Selecting Web Services from your workspace

4. In the next window make sure that **Workspace WSDL documents** is selected from the drop-down list. Click on the imported Web service (named **CustomerOrderRequestPortTypeExport1** in our example) as shown in Figure 8-15 on page 216.

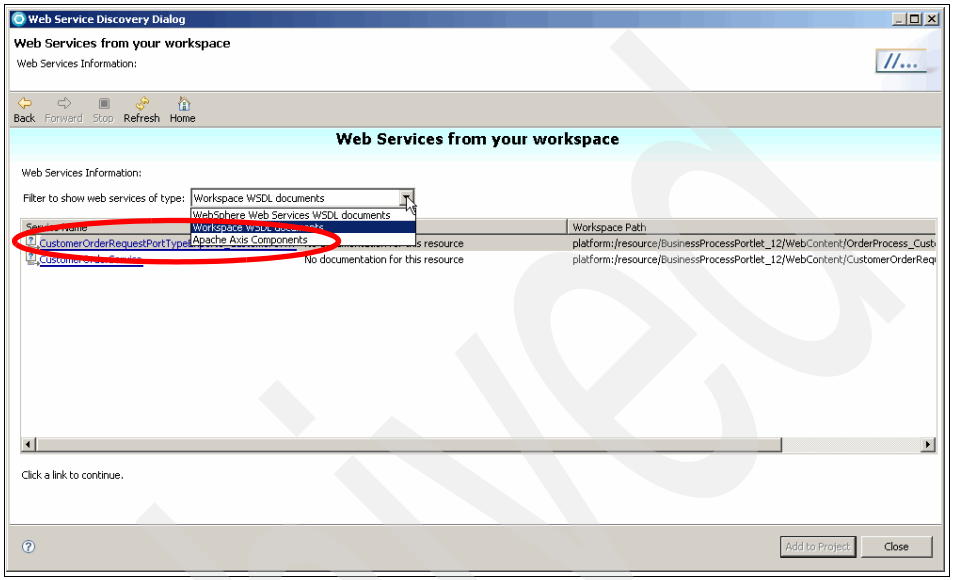

Figure 8-15 Selecting the imported Web service

5. On the next window, “Web Services from a known URL”, from the **URL** drop-down list select the imported WSDL file, expand **Service**, select **Port:CustomerOrderRequestPortTypeExport1**, and click **Add to Project** as shown in Figure 8-16.

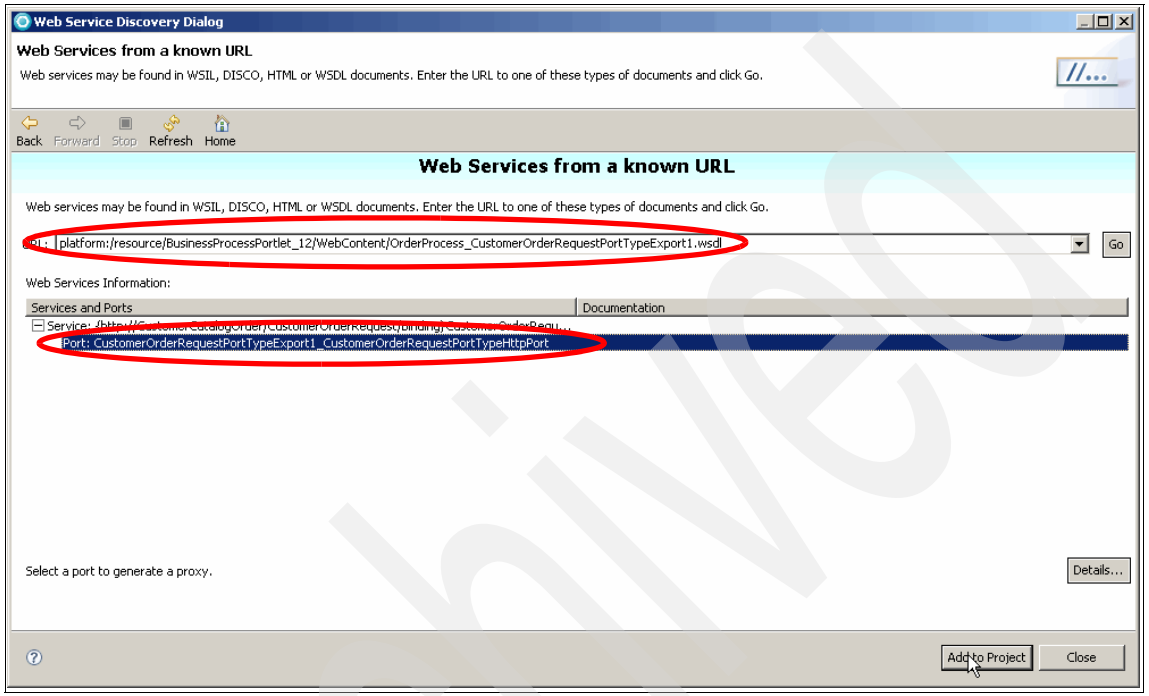

Figure 8-16 Selecting SOAP_HTTP_Port to generate poxy

6. If you receive a **Warning** window about enabling automatic file overwriting click **Yes All** as shown in Figure 8-17.

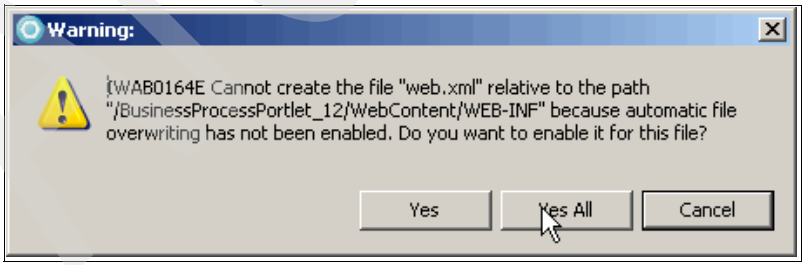

Figure 8-17 Enabling automatic file overwriting

7. On the Web Service panel, under **Select a Web Service** select **CustomerOrderRequestPortTypeProxy**.

   From the **Select a method** drop-down list select **customerOrderOperation,** and click **Next** as shown in Figure 8-18.

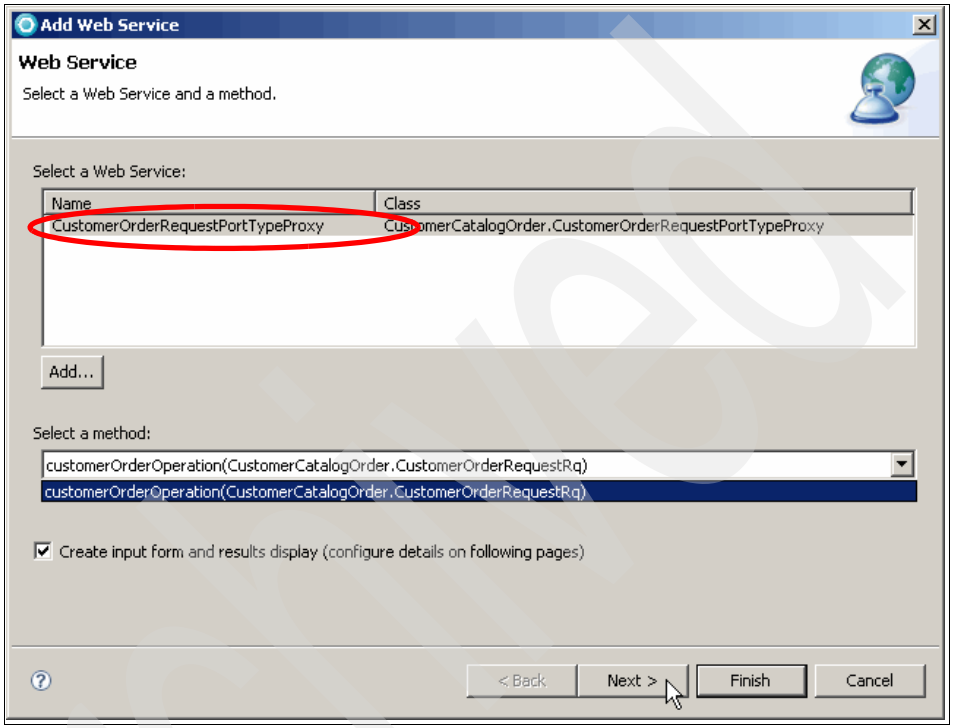

Figure 8-18 Selecting a Web service and method

You want users of this application to input a number of variables on the initial page. The variables are the same ones you used in Chapter 6, “Creating the Business Process” on page 85. This is accomplished through the design of our input form using the Input Form wizard.

8. In the **Input Form** screen, make sure all proposed fields are checked and click **Next** (Figure 8-19).

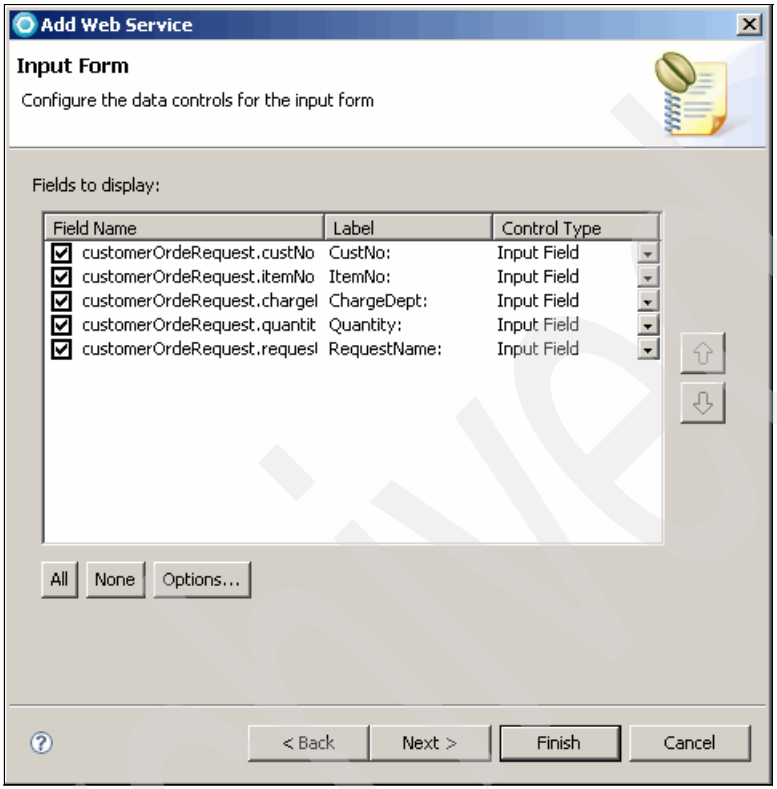

Figure 8-19 Designing the Input Form

In the output you will design a return message to be displayed. Do this through the design of the results form using the Results Form wizard.

9. In the **Results Form** panel accept the defaults and click **Finish** (Figure 8-20).

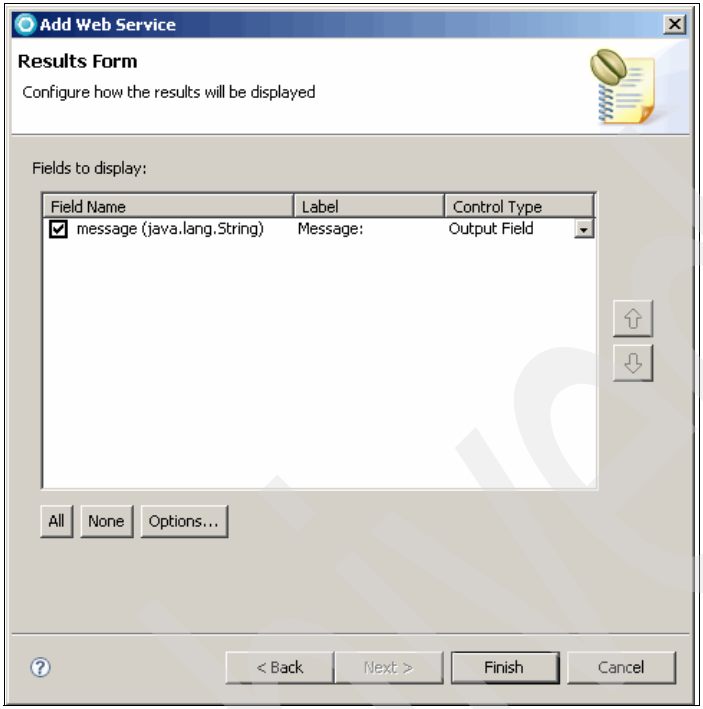

Figure 8-20 Designing the Results Form

Your screen should now appear similiar to Figure 8-21 on page 221.

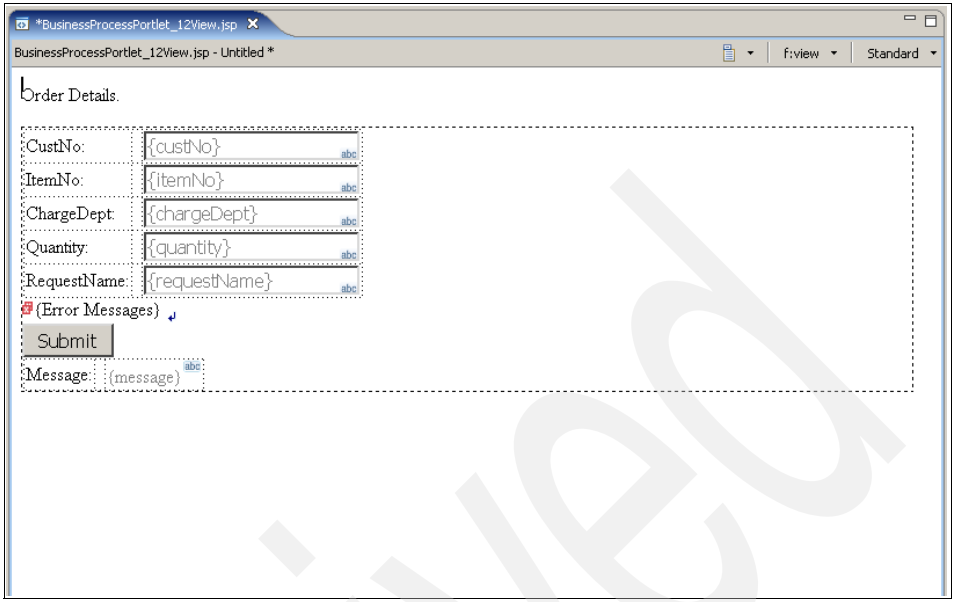

Figure 8-21 Modified JSP

10. **Save all** and close the JSP window.

8.1.4 Exporting the war file

1. In the **Project Explorer** pane, right-click the Portlet and select **Export** → **WAR file** as shown in Figure 8-22.

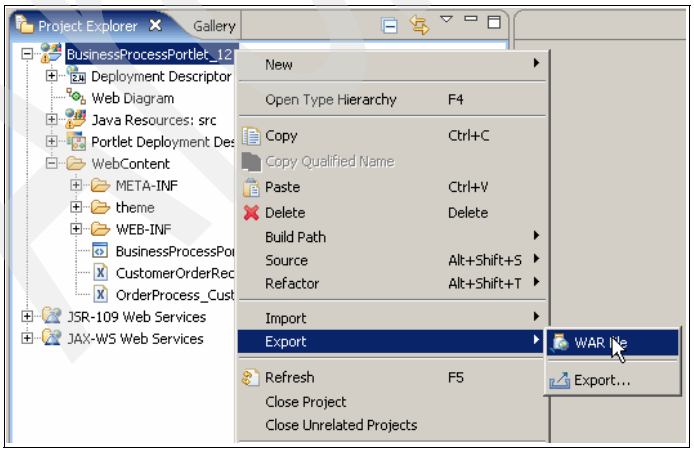

Figure 8-22 Selecting Export - WAR file

2. Use the Web module pull-down list to select the Portlet project as shown in Figure 8-23.
3. In the Destination field enter the location of the war file by clicking the **Browse** button and selecting the appropriate entry.

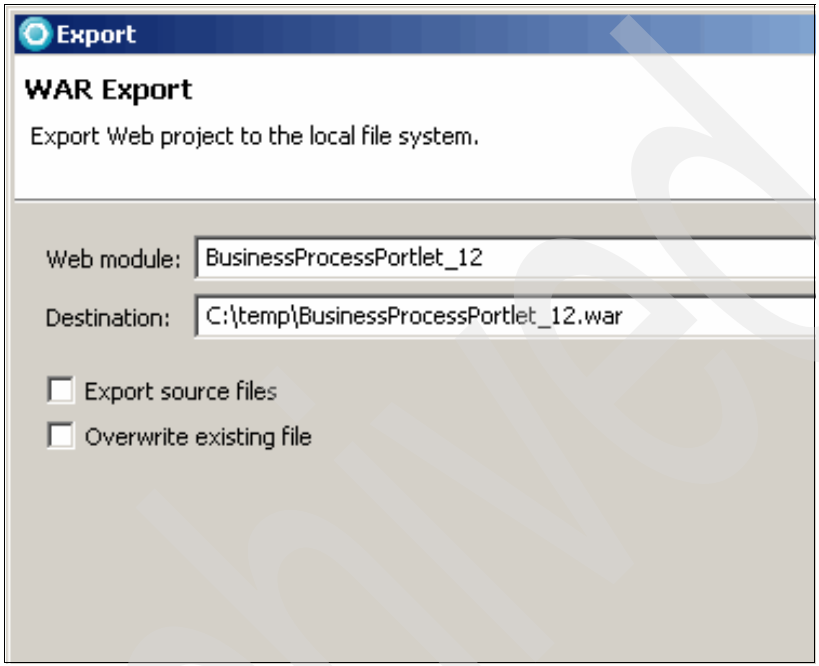

Figure 8-23 Exporting the war file to File System

8.2 Deploying the Web service Portlet

In the next steps we show how to deploy the Web service Portlet to the Portal on z/OS.

1. Click **Administration** in the Product Links Portlet (Figure 8-24).

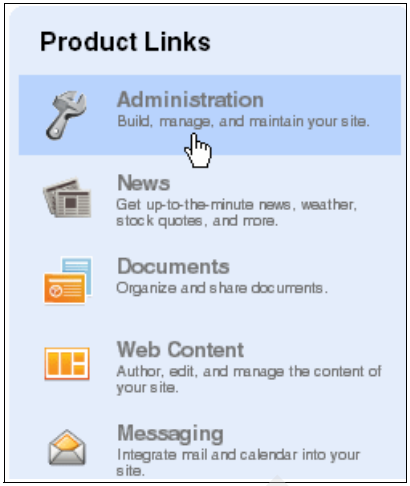

Figure 8-24 Selecting Administration in Product Links Portlet

2. On the left navigation pane of the WebSphere Portal Welcome to Administration page, expand **Portlet Management** and click **Web Modules** as shown in Figure 8-25.

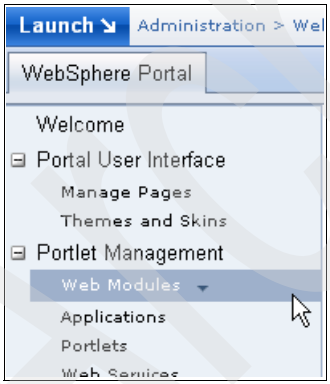

Figure 8-25 Selecting Web Modules from Portlet Management

3. On the Manage Web Modules page click **Install**.
4. Click **Browse** to select the war file you exported in the previous steps. Click **Open**.

5. Click **Next** on the Manage Web Modules page as shown in Figure 8-26.

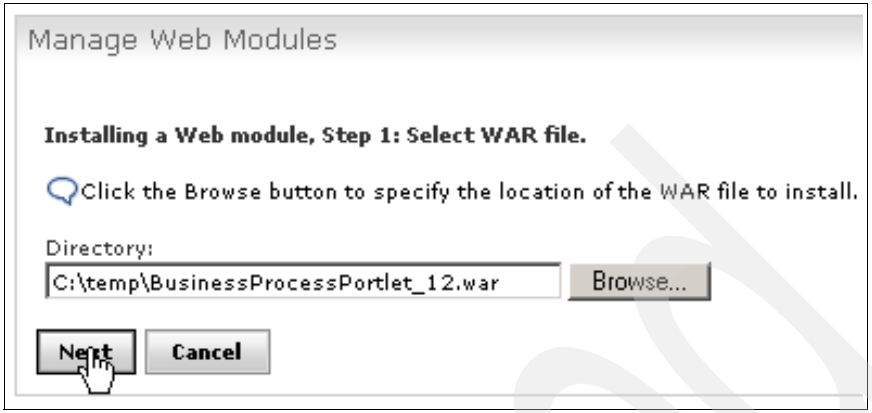

Figure 8-26 Selecting the WAR file

6. The Manage Web Modules screen now displays the contents of the Portlet war file. Click **Finish** to complete the install.

   You should receive the following message:

   ```
   EJPAQ1330W: the Web module was successfully installed but must be
   manually started.
   ```

8.2.1 Starting the Portlet

Now that you have installed the Portlet application it must be started manually. You start the Web module by activating it using the following steps.

1. On the Manage Web Modules page use the Search field to locate your Portlet, as shown in Figure 8-27. We typed a “B” because the Web module name in our example starts with “B”.

Figure 8-27 Searching for Web Modules

2. You will notice that your Web module has a **Stopped** status as shown in Figure 8-28 on page 225. Click the Activate (lightening bolt) icon to start the Web module.

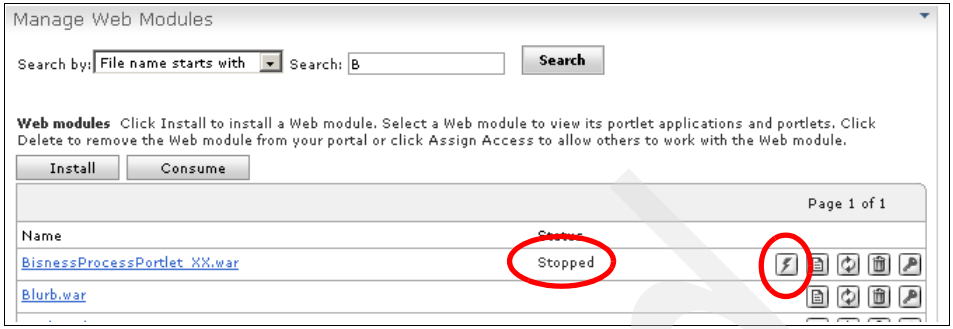

Figure 8-28 Starting the Web module

You should receive a message similar to:

```
EJPAQ1202I: Web module BusinessProcessPortlet_XX has been
successfully started.
```

8.2.2 Testing the Portlet

It is recommended to add the deployed Portlet to a page, but we do not describe these steps here. Once the Portlet is started and added to a page, it can be tested. The Portlet should resemble Figure 8-29.

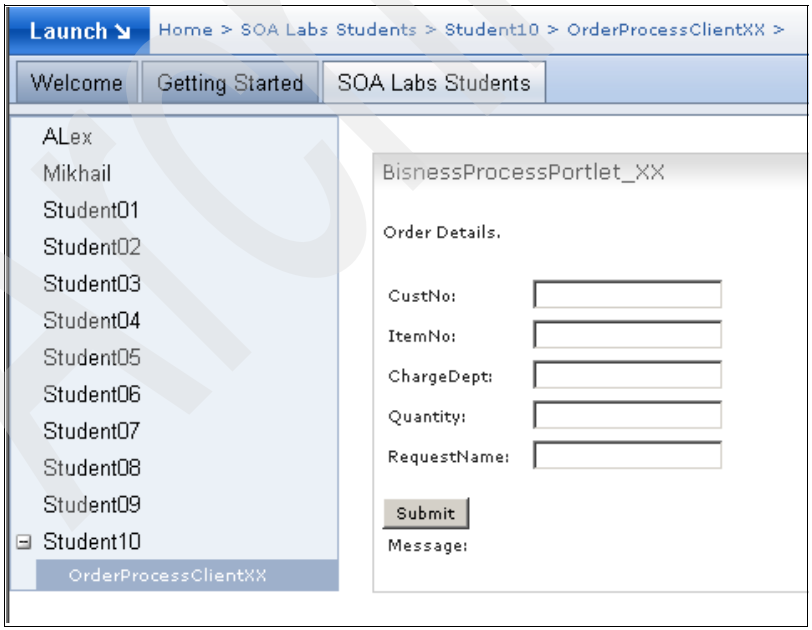

Figure 8-29 Portlet input values

1. Enter the following values:
   - CustNo: `001`
   - ItemNo: `0050`
   - ChargeDept: `ITSO`
   - Quantity: `1`
   - RequestName: `SOASFREQ` (Note - uppercase!)

   Click **Submit.** See Figure 8-30.

Figure 8-30 Input form with values

2. A Result Form similar to Figure 8-31 should appear.

Figure 8-31 Result Form with output

3. Optionally, you can perform another test with an invalid customer number and trigger the Human Task. You can then use the techniques discussed in “Testing the Human Task with built-in client” on page 163 to resolve the issue using the BPC.

Note: What has happened? You used a Portlet in WebSphere Portal to drive a process in WebSphere Process Server (the process designed in Chapter 6, “Creating the Business Process” on page 85). This process used an ESB (WebSphere Message Broker) to perform a credit check and a CICS service flow to place an order.

8.3 Additional material

A workspace is included in the additional material with the Portlet application. See Appendix A, “Additional material” on page 229.

A

Additional material

This document refers to additional material that can be downloaded from the Internet as described in this section.

Locating the Web material

The Web material associated with this paper is available in softcopy on the Internet from the IBM Redbooks Web server. Point your Web browser at:

`ftp://www.redbooks.ibm.com/redbooks/SG244391`

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 IBMRedpaper form number, REDP4391.

The zip files described in the following sections are available.

Customer service inquiry

For the Customer inquiry service, we have supplied the following zip files with additional materials:

- **Workspace - RDz - Customer inquiry service.zip**, containing an RDz workspace with the generated Customer inquiry Web service.
- **zOS - Customer inquiry service.zip**, containing the z/OS components for CICS and DB2.

Credit check service

For the Credit check service, we have supplied the following zip files with additional materials:

- **Workspace - WMB Toolkit - Credit check service.zip**, containing an WMB Toolkit workspace with the WMB message flow for the Credit check service
- **zOS - Credit check service.zip**, containing the z/OS components for DB2.

Order placement service

For the Order placement service, we have supplied the following zip files with additional materials:

- **Workspace - RDz - Order placement service.zip**, containing an RDz workspace with the generated Order placement Web service.
- **zOS - Order placement service.zip**, containing the z/OS components for CICS and DB2.

Order process

For the Order process developed in WID, we have supplied the following zip file:

- **Workspace - WID - Order process.zip**, containing a WID workspace with the generated Order process.

Order process Portal user interface

For the Order process user interface, we have supplied the following zip file:

- **Workspace - RAD - Order portlet.zip**, containing a RAD workspace with the generated Portlet user interface.

System requirements for downloading the Web material

There are no specific requirements for downloading the material with regard to space, memory, and so on. However, to successfully use each of the development tools discussed in this document specific requirements might apply. Refer to the specific product documentation for the development tool you plan to use.

How to use the Web material

There are two types of zip files: zip files with workspace to use in development tools and zip files with z/OS source components.

Zip files with workspaces

For each workspace zip file, create a subdirectory (folder) on your workstation, and unzip the contents of the zip file into this folder.

To start a development tool with any of the workspaces downloaded, start the development tool and select the workspace based on the folder you used to unzip the zip file to.

Note: Not every workspace can be used in every development tool:

- A WMB Toolkit workspace can only be used in WMB Toolkit.
- An RDz workspace can only be used in RDz.
- A RAD workspace can be used in both RAD and RDz.
- A WID workspace can only be used in WID.

This is because each workspace uses specific functions that are only available in specific tools.

Zip files with z/OS components

Each z/OS component, a job COBOL source, and so on, is kept in a Windows Text file inside the zip file. If you wish to use a z/OS component on z/OS, you can upload each Text file to a member of a PDS on z/OS in ASCII mode. You can also import the components into a project in RDz and perform uploads to z/OS from there.

Related publications

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

IBM Redbooks publications

For information about ordering these publications, see “How to get Redbooks publications” on page 233. Note that some of the documents referenced here may be available in softcopy only.

- *Patterns: Building Serial and Parallel Processes for IBM WebSphere Process Server V6*, SG24-7205
- *Implementing an ESB using IBM WebSphere Message Broker V6 and WebSphere ESB V6 on z/OS*, SG24-7335
- *Using IBM WebSphere Message Broker as an ESB with WebSphere Process Server,* SG24-7527

How to get Redbooks publications

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

An End-to-End SOA Integration Scenario using IBM WebSphere Process Server on z/OS

Redpaper™

Development and deployment of a business process

Reuse of CICS and DB2 assets as services

Using WebSphere Message Broker as ESB

This paper demonstrates an end-to-end SOA integration scenario on z/OS following a simple business scenario. The paper discusses the architecture, introduces the products used and describes how the scenario was developed.

Key products used are WebSphere Portal, WebSphere Process Server, WebSphere Message broker and CICS.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

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

For more information:
ibm.com/redbooks

REDP-4391-00