HP Service Oriented Architecture Policy Enforcer

Integration Guide

Version: 3.00

Windows®, HP-UX, Linux

June 2008

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notices

Trademark Notices

Java™ and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation

UNIX® is a registered trademark of The Open Group

To view open source code, see the Installation\license\thirdparty\source directory on the product installation media.

Support

You can visit the HP Software support web site at:

www.hp.com/go/hpsoftwaresupport

This Web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the HP Software Support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:

www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

www.managementsoftware.hp.com/passport-registration.html

Contents

Table of Contents

1

Introduction

The *Integration Guide* is intended for users who want to take advantage of the HP Service Oriented Architecture Policy Enforcer (SOA PE) integration possibilities. SOA PE Integration is often performed to either customize how the SOA PE works or to reuse the SOA PE's assets within other enterprise applications.

Much attention has been given to ensure that the SOA PE's architecture is flexible and easy to integrate with. The architecture is pluggable and uses popular industry standards such as Java and XML. More important, the SOA PE is among the first distributed management software products to use a services-based architecture. For example, Web services are used to expose many parts of the SOA PE's management model. This practice not only ensures standard and open integration possibilities, it clearly sets the SOA PE within current and future SOA-based environments.

Document Overview

The *SOA PE Integration Guide* provides instructions for performing integrations with the SOA PE. The guide provides a brief contextual overview of the customization and integration possibilities. Generally, each chapter is dedicated to an integration possibility and is self contained. Therefore, the book need not be read in sequence.

Every effort has been made to explain general concepts. However, much of the content in this guide assumes that the user is already familiar with the SOA PE and has a basic level of experience. If you are new to the SOA PE product, it is recommended that you first become familiar with the product. A good starting point is the *SOA PE User Guide* and the *SOA PE Tutorials*. These and other documents are located in the distribution in the `/Documentation` directory.

Audience

The *Integration Guide* is primarily intended for solution architects, systems integrators, and application developers who are responsible for integrating and enabling management components in their environments.

2

Integration with SOA PE Overview

Integration Points

SOA PE provides several integration points. The integration points can facilitate the deployment of Web services management solution as well as provide an opportunity to repurpose some of the management data that is collected for Web services. In some cases, there is also the ability to customize the solution with management features that are not supported with the solution, but important to fulfill the management needs of a Web service deployment.

In general, the integration points have been organized into the categories listed below. This is purely an organizational scheme and does not suggest any functional boundaries within the product.

- SOA PE includes the following integration points:
  — Database – This integration point is used to create custom audit reports or to include audit information within other management applications.
  — Web services interfaces – This integration point is used to create new (or augment existing) enterprise management applications by reusing the SOA PE's management data and management model. The data and model are exposed as management Web services.
- SOA PE policy enforcement intermediary – The intermediary supports integration with Java APIs. This integration helps you create custom management handlers that address the specific Web service management requirements of an organization.
- Integration with other products – You can integrate SOA PE with the following products:
  — HP Business Process Insight (BPI)
  — HP SOA Systinet
  — HP Diagnostics

Prerequisites for Integration

These steps are used to manually create SOA PE's assets as well as customize the SOA PE and the policy enforcement intermediary startup. The prerequisites for integration are as follows:

- Customize startup behaviors.
- Automate repetitive tasks.
- Save time when updating SOA PE assets over multiple installations.
- Reuse current tool sets (IDE, Content Versioning System, and so on).
- Maintain current development processes.

Startup Tasks

SOA PE policy enforcement intermediary group has the ability to execute user-defined Java code at startup. This is useful for initializing any required services when the processes are started or executing any type of process initialization code.

Integrators are responsible for creating the user-defined Java code as well as configuring each server to use the custom code. Configuration of startup classes is done in the servers' XML configuration file (`server.xml`).

SOA PE

You can integrate with SOA PE by using published Web services interfaces or databases.

Web Services Integration

The most common and robust method for integrating with SOA PE is by using the SOA PE's published Web services interfaces. The Web services are SOAP based Web services that are defined using WSDL. The Web services follow standard Web services management protocols.

> Two Web services management specifications are: Web Services Distributed Management (WSDM) and Web Services for Management (WS-Management). At present, SOA PE software utilizes the Web Services catalog, which is an HP authored precursor to the standard WS management protocols. SOA PE software supports these specifications as they mature and stabilize.

Web services integrations allow system architects to leverage current management investments and provide a broader and more thorough view of the enterprise. In general, Web services integrations are used to:

- Link SOA PE with other enterprise management products to create composite and custom applications
- Create and/or reuse current management consoles to display the SOA PE's management data
- Create custom management consoles

Database Integrations

Web services audit trace messages are persisted to a central database. This information in conjunction with the model relationships which are also populated into the database (such as which Business Service contains which Web services) can be used to enable reporting and analytical applications such as SLA reporting, billing, non-repudiation, forecasting, and so on. Any application can potentially connect to the SOA PE database and make use of the data. For example, the database can be used to create audit reports using packages like Crystal Reports.

Integrators are responsible for creating and maintaining database connections from within their applications. In addition, integrators are responsible for upgrading their applications as the SOA PE's database schema changes.

SOA PE Policy Enforcement Intermediary Group

The Broker Configurator tool is typically used to create intermediary groups, configure intermediary group management handlers, and deploy intermediary groups. These steps can become repetitive and time consuming depending on the number of intermediary groups that are being deployed. However, these tasks can be completed without using the Broker Configurator.

Integrators are responsible for creating the intermediary group definition file (a proprietary file written using XML), packaging the intermediary group as a Java Archive (jar file), and copying it to the appropriate directory on the policy enforcement intermediary group. Depending on the requirements, some or all of these tasks can be automated.

Java API Integrations

Policy handlers implement management logic that is used to interpose visibility and controls on Web services. These handlers are inserted in the HTTP or SOAP pipeline that is responsible for processing request and response messages. Intermediary groups use a set of standard handlers and can also be configured to use a set of simple or advanced handlers. However, an organization may have special management requirements that are not covered by any of the provided handlers.

In such cases, the SOA PE's Java API can be used to create custom handlers that can implement any management or processing logic that is required.

Integrators are responsible for:

- Creating the custom handler Java logic by extending a base class
- Compiling the handler
- Configuring the handler in the intermediary group definition file
- Packaging the handler in the intermediary group jar file

Security Customization

The SOA PE's Java API integration point is used to create custom security implementations that allow an organization to enforce security policies that are not covered by the default security features provided by the policy enforcement intermediary. This includes custom authentication based on user profile, custom security handlers, and an XML Introspection service.

3

Prerequisites for Integration

Before integration, you must make sure that you perform the following activities:

- Customize startup behaviors
- Automate repetitive tasks
- Save time when updating SOA PE assets over multiple installations
- Reuse current tool sets (IDE, Content Versioning System, and so on)
- Maintain current development processes

Startup Tasks for SOA PE Server and SOA PE Intermediary

SOA PE and SOA PE Intermediary have the ability to execute user-defined Java code at startup. This is useful for initializing any required services when the process is started, or for executing any other type of process initialization code. To have either SOA PE or intermediary execute startup code, a class must be created that implements the `com.hp.wsm.sn.server.StartupTask` interface. This startup class should be registered in the `service.xml` file of either SOA PE or SOA PE Intermediary. This file is present in the `/conf` directory.

StartupTask Interface

A startup class must implement the `com.hp.wsm.sn.server.StartupTask` interface in order to be executed. The `StartupTask` interface is as follows:

```
public interface StartupTask {
  public void startup(MipServer server) throws
  StartupTaskFailureException,
  LicenseException.LicenseRunTimeError,
  LicenseException.LicenseNotFound;
}
```

The class must be available on the classpath. When the server executes the startup task, it creates a new instance of the class and executes the startup method, passing in the current `com.hp.wsm.sn.server.MipServer` instance. The `MipServer` class is used for internal processing and is generally not used by user-defined startup tasks. The `MipServer` interface is not documented and is subject to change.

The startup class should throw a subclass of the `com.hp.wsm.sn.server.StartupTaskFailureException` class if a critical problem occurred and the server should shutdown.

Configuration File (server.xml)

The `server.xml` file is located in the *<install_dir>*/conf/broker and *<install_dir>*/conf/networkservices directories, respectively. Underneath the root `<server>` element, it has a `<startup>` element. This is the element in which the list of startup classes is contained. Each startup class should be specified within `<classname>` tags. For example:

```
<server>
   <startup>
     <classname>
         com.hp.wsm.sn.networkservices.NetworkServicesStartupTask
      </classname>
   </startup>
</server>
```

The startup tasks are executed in the order in which they appear in the startup list. User-defined startup tasks should follow the existing `com.hp.wsm.sn` startup tasks. While it is possible to put a startup task before the existing `NetworkServicesStartupTask` or `BrokerStartupTask`, it is not recommended.

Manually Implementing Intermediary Web Services

When using a SOA PE Intermediary-based deployment scenario, the Broker Configurator is typically used to create intermediary Web services. This includes configuring an intermediary Web service's management handlers. As an option, users can manually create intermediary Web services and configure their management handlers. Intermediary Web services are manually created and deployed when:

- You do not want to use the Broker Configurator.

- You want to develop and manage intermediary Web services using your own development environment (IDE, Content Versioning System, development processes).

- You want to create custom intermediary Web services.

The instructions in this section demonstrate how to manually create an intermediary Web service and deploy it to the Policy Enforcement Point (PEP).

Creating an Intermediary Web Service JAR

The artifacts of an intermediary Web service are packaged as a JAR file. A good method for learning about intermediary Web services is to create an intermediary Web service using the Broker Configurator, and then inspecting the intermediary Web service JAR file or using the files in the JAR as a template for creating your own intermediary Web services. When using the Broker Configurator, the JAR file is written to the *<install_dir>*/conf/broker.

The archive contains two files which can be manually created:

- `service.wsdl` – This file contains a service's definition without the address (endpoints) for the service. The service can be defined using either SOAP semantics, or can be created using straight XML.

- `service.xml` – This file contains a service's endpoints and also the management capabilities that will be interposed for the service.

Writing the Intermediary Web Service Definition

The intermediary Web service definition (`service.xml`) is an XML-based file and can be created using a text editor or an XML editor. Among other things, the definition contains a service's endpoints and references to any management handlers. The file can contain handlers that are included with the WSM solution, or can contain any custom handlers you create. The `service.xml` contains the following elements:

<service>

This is the root element of the intermediary Web service definition. The following attributes are included as part of the `<service>` element.

- `class`: Defines the class used to create the intermediary Web service. There are four possible classes that you can use:

- — `com.hp.wsm.sn.router.xml.SimpleSoapServiceFactory`: This class is used to create an intermediary service for SOAP-based services as defined in the service.wsdl file. The intermediary Web service can contain a predefined set of handlers often referred to as simple handlers.

- — `com.hp.wsm.sn.router.xml.SimpleXmlServiceFactory`: This class is used to create an intermediary Web service for XML-based services as defined in the service.wsdl file. The intermediary Web service can contain a predefined set of handlers often referred to as simple handlers.

- — `com.hp.wsm.sn.router.xml.CustomSoapServiceFactory`: This class is used to create an intermediary service for SOAP-based services as defined in the service.wsdl file. The intermediary Web service can contain a broad set of handlers and can also contain any custom handlers you create.

- — `com.hp.wsm.sn.router.xml.CustomXmlServiceFactory`: This class is used to create an intermediary Web service for XML-based services as defined in the service.wsdl file. The intermediary Web service can contain a broad set of handlers and can also contain any custom handlers you create.

- • `name`: The name of the intermediary Web service

- • `version`: A version for the intermediary Web service

- • `namespace`: A list of namespaces entered as *prefix=url*

<transport>

The `<transport>` element is contained within the `<service>` element and is used to define the transport layer used to access the intermediary Web service. It contains a reference to the transport provider as well as the http context that is used to access the intermediary Web service and whether the transport layer is secured.

The following example demonstrates the transport definition:

```
<transport
    provider="com.hp.wsm.sn.router.http.HttpTransportProvider">
    <ns1:property name="http.context" value="/financeServiceProxy"/>
    <ns1:property name="http.secure" value="false"/>
</transport>
```

<routing>

The `<routing>` element is contained within the `<service>` element and is used to bind the intermediary service to a Web service endpoint. The dispatcher is used to send the request to final endpoint. The following example demonstrates the routing definition:

```
<routing>
    <entry binding="ns2:FinanceServiceSoapSoapBinding">
        <ns3:endpoint
            type="com.hp.wsm.sn.common.dispatcher.
                endpoint.http.HttpEndpointProvider">
            <ns1:property name="priority" value="primary"/>
            <ns4:address>
                http://15.40.235.105:8080/axis/services/FinanceServiceSoap
            </ns4:address>
        </ns3:endpoint>
    </entry>
</routing>
```

Handler Definitions

Each handler that you want to include for the intermediary Web service must be referenced in the intermediary service definition and be contained within the `<service>` element. Any properties for the handler must be included as attributes. The following example demonstrates the audit handler definition:

```
<audit
    classname="com.hp.wsm.sn.router.xml.handlers.audit.
                AuditHandlerFactory"
    includeProfiling="true"
    payload-filter="ALL"
    payload-option="REQUEST-RESPONSE"
</audit>
```

> A reference of all the handlers can be found in the *SOA PE User Guide*.

Creating a Custom Handler

The following example demonstrates a custom handler:

```
Package com.mycompany.CustomHandler

import com.hp.wsm.sn.router.common.message.MessageServiceException;
import com.hp.wsm.sn.router.xml.XmlOperation;
import com.hp.wsm.sn.router.xml.handlers.BaseXmlHandler;
import java.io.IOException;

public class CustomHandler extends BaseXmlHandler {
  public void onRequest(XmlOperation operation) throws
    MessageServiceException {
    try {
        // Custom code goes here.
    }
    catch (IOException e) {
      throw new MessageServiceException.RequestReadException(e);
    }
  }
}
```

> An `onResponse` method is also available when processing a response message.
>
> An operation can be retrieved by using `getOperation()` or `getOperationAsString()` on the `XmlOperation` object.

Accessing Attachments for SOAP with Attachments

The SOA PE Intermediary provides an API that can be used in custom handlers to access attachments that are part of a SOAP with Attachments (SwA) message. This API is part of the `com.hp.wsm.sn.router.xml.SoapOperation` class and is therefore only available for SOAP services.

To access the soap attachments of a service request in a custom intermediary Web service handler:

1   Create a Handler class which extends `BaseXmlHandler` and overwrites the methods:

```
public void onRequest(XmlOperation operation)
public void onResponse(XmlOperation operation)
```

2   In the `onRequest` and/or `onResponse` method, verify that the `XmlOperation` parameter is an instance of `SoapOperation`.

3   If it is a `SoapOperation`, the attachments can be accessed by casting the `XmlOperation` to a `SoapOperation` and calling:

```
public Attachments getRequestAttachments()
public Attachments getResponseAttachments()
```

The `com.hp.wsm.sn.router.soap.Attachments` class that is returned from these operations has methods to retrieve body parts by index:

```
public BodyPart get(int i)
```

Or by name:

```
public BodyPart getAttachmentByPartName(String partName)
```

The returned type is `javax.mail.BodyPart`. Below is an example of a handler that accesses attachments from a SwA message:

```
package …

import com.hp.wsm.sn.router.xml.handlers.BaseXmlHandler;
import com.hp.wsm.sn.router.xml.XmlOperation;
import com.hp.wsm.sn.router.xml.SoapOperation;
import com.hp.wsm.sn.router.common.message.
   MessageServiceException;
import com.hp.wsm.sn.router.soap.Attachments;

public class SoapAttachmentHandler extends BaseXmlHandler {
   public void onRequest(XmlOperation operation)
      throws MessageServiceException {

// check that the operation is an instance of SoapOperation

         if (operation instanceof SoapOperation) {
            SoapOperation soapOp = (SoapOperation)operation;
            Attachments attachments =
               soapOp.getRequestAttachments();

// add code to manipulate the request attachment parts.

         }
   }

   public void onResponse(XmlOperation operation)
      throws MessageServiceException.InternalError {

// check that the operation is an instance of SoapOperation

          if (operation instanceof SoapOperation) {
            SoapOperation soapOp = (SoapOperation)operation;
            Attachments attachments =
               soapOp.getResponseAttachments();

// add code to manipulate the response attachment parts.

         }
   }
}
```

Adding Custom Handlers to an Intermediary Web Service Definition

Custom intermediary Web services allow you to add your own custom handlers to an intermediary Web service's handler chain. In order to add a custom handler, you must first create the custom intermediary Web service and then edit the service's definition file located in the intermediary Web service jar file.

To add a custom handler:

1   Uncompress *<install_dir>*\conf\broker*<intermediary_web_service_name>*.jar.

2   Using a text (or XML) editor, open `service.xml`.

3   Under the `<service>` element, add a `<handler>` element and include the fully qualified class name. For example:

```
<handler classname="com.company.HandlerClass" />
```

4   If the handler requires any properties, add them as elements under the handler class. For example:

```
<handler classname="com.company.HandlerClass" >
    <property1>foo</property1>
    <property2>
       <property name="foo" value="bar" />
    </property2>
    <ns1:property3>foo</ns1:property3>
</handler>
```

> If the property uses a namespace, you must declare the namespace as an attribute of the `<service>` element before using the namespace (`xmlns:ns1="com.company"`).

5   Save and close `service.xml`.

6   Place the custom handler class and any dependent classes in the same directory as `service.xml`.

7   Re-jar the intermediary Web service including the custom handler class and any dependent classes.

Deploying an Intermediary Web Service Jar

Intermediary Web service JARs are placed in the *<install_dir>*/conf/broker directory. Any intermediary Web service JARs that are in this directory are automatically deployed by the SOA PE Intermediary. When the JAR is deployed, it is automatically extracted to *<install_dir>*/conf/broker and placed in a directory that is named using the IP address of the host computer.

> Never edit the extracted files as they will be overwritten when a new JAR is deployed.

4

Database Integration

This chapter describes how to create SLA audit reports using the SOA PE Server's audit database. The chapter provides a brief overview of the integration architecture and includes a reference of the schema and data dictionary of the audit database. An example of an SLA audit report is also provided.

Overview

SOA PE has the ability to capture audit information about Web service messages into a central audit database. When a policy enforcement intermediary group is registered with SOA PE, the Audit Service registers an event callback listener. Subsequently, the intermediary posts Audit Trace message lists back to the Audit Service at a configured interval. The Audit Service inserts these trace messages into the Audit database. This information in conjunction with the model relationships which are also populated into the database (such as which Business Service contains which Web services) can be used to enable reporting and analytical applications such as SLA reporting, billing, non-repudiation, forecasting, etc.

This chapter provides instructions for producing custom SLA reports based on the management data that is stored in the Audit database.

Database Schema Reference

There are two tables that are used to create SLA Audit reports:

- MESSAGE_TRACE
- MESSAGE

MESSAGE_TRACE Table

Name	Type	Description
sequenceId	xsd:string	An identifier that links together

Name	Type	Description
		`MessageTrace` instances that represent hops between nodes (i.e. intermediaries) for the same message call. This information is currently passed between nodes in the HTTP header `SequenceHeader`. If a node receives a message that has a `SequenceHeader`, it should use that value to populate the `sequenceId` field of the `MessageTrace` for that message. If a node receives a message that does not have a `SequenceHeader`, the node should generate a new globally unique `sequenceId`. It should use this value to populate the `sequenceId` field of the `MessageTrace` for that message. In either case, the node should also add a `SequenceHeader` HTTP header to any outgoing messages associated with this call and populate it with the appropriate `sequenceId`.
traceSourceId	xsd:string	A unique identifier that identifies the source of the message trace – the intermediary that is sending the message trace to the `CollectionService`. The value is the host and port of the router sending the trace.
traceSourceRole	xsd:string	A string indicating the role being played by the sender of this message trace with regard to the message being traced.
serviceName	xsd:string	The name of the service being invoked. The value of this field is the name of the service element in the WSDL for this service.
serviceVersion	xsd:string	The version of the service being invoked. The value of this field is the namespace of the service element in the WSDL for this service.
portType	xsd:string	The WSDL `portType` being invoked.
operation	xsd:string	The WSDL `operation` being invoked.
binding	xsd:string	The WSDL `binding` being invoked.
serviceType	xsd:string	An indicator if the service is managed internally (proxy) or is outside the service network. This field is restricted to the values `internal` and `external`.

Name	Type	Description
transportType	xsd:string	The transport on which the message traveled. This is typically http or jms.
consumerSecurityPrincipal	xsd:string	The Principal invoking the service.
providerSecurityPrincipal	xsd:string	The Principal used by the services network on behalf of the consumer.
senderURI	xsd:string	A URI identifying the message source.
receiverURI	xsd:string	A URI identifying the message destination.
duration	xsd:long	The duration in milliseconds of the call from the time the request was received by this node to when it was completed.
timestamp	xsd:dateTime	The time at which the call was received. This should be in UTC format ('Z' extension) with Millisecond (.sss) precision.
errorType	xsd:string	An indicator for the type of error that occurred, if any. This field is restricted to the values none, application, transport, timeout, marshalling, and soap fault.
requestSize	xsd:int	The size of the request in bytes.
responseSize	xsd:int	The size of the response in bytes

MESSAGE Table

This table contains actual message payloads.

Name	Type	Description
sequenceId	xsd:string	The UUID representing this message.
requestMessage	CLOB	The payload body of a request message.
responseMessage	CLOB	The payload body of a response message.

Sample SLA Reports

You can build SLA reports from the SOA PE Audit database using a reporting package like Crystal Reports. Refer to the Audit database schema and data dictionary to understand the source of this information.

```
From: MM/DD/YY HH:MM:SS
To: MM/DD/YY HH:MM:SS
Compute over interval:  MONTH / WEEK / DAY / HOUR
```

```
Consumer: <securityPrincipal>
    Consumed Service: <serviceNS>:<serviceName>
    Number of Requests made:
    Number of Requests processed successfully:
    Number of Requests failed:
    %age Availability over interval: (Number of Requests processed
        successfully / Number of Requests made) X 100
    Max Response time:
    Average Response time:
```

Sample SLA Reports for Month of Nov 2004

From: 11/01/04 00:00:00

To: 11/30/04 00:00:00

Compute over : DAY

Consumer: joebob

Consumed Service: http://wsm.hp.com/finance:financeServiceProxy

Time Window	Number of Requests	Number of Successes	Number of failures	% Availability	Ave Response (ms)	Max Response (ms)
11/01/04 – 11/02/04	35	24	11	68.57	274	360
11/02/04 – 11/03/04	45	44	1	97.77	248	423
…						

Consumed Service: http://wsm.hp.com/mobile:mobileServiceProxy

Time Window	Number of Requests	Number of Successes	Number of failures	% Availability	Ave Response (ms)	Max Response (ms)
11/01/04 – 11/02/04	29	29	0	100	128	175
11/02/04- 11/03/04	13	13	0	100	110	155
…						

5

HP BPI Integration

HP Business Process Insight (BPI) is one of the HP Software products that you can integrate with SOA PE. You can configure instances of the HP BPI SOA PE adapter to integrate with your SOA PE installations. You can then configure HP BPI to link SOA PE business events to the business flows that you define. SOA PE enables you to manage your service oriented architecture (SOA) resources to ensure their reliability and optimize their performance. The combination of SOA PE and HP Business Process Insight provides you with the ability to monitor the health and performance of services running within a Service Oriented Architecture. This chapter provides an overview of the integration, configuring, and troubleshooting instructions when integrating HP BPI with SOA PE. Refer to the HP Business Process Insight (BPI) manuals for more information about detailed instructions and conceptual information regarding HP BPI.

Integration Instructions

The HP BPI-SOA PE integration works as follows:

- Within the HP BPI Modeler, you are able to specify the SOA PE Business Services on which your business flow depends.

- You then deploy your flow to the HP BPI Business Impact Engine.

- As these Business Services change state, this status information is collected from SOA PE and your HP BPI flow reflects any resultant business impact.

For SOA PE to be able to communicate with HP BPI you need to install and run the HP BPI SOA PE Adapter component. This SOA PE Adapter component can be installed on any machine that has access to your network, but typically you would install the SOA PE Adapter component on the SOA PE server. Refer to the BPI Installation Guide for more details.

Let's consider the architecture where you have installed HP BPI and the SOA PE adapter on different servers where:

- You install and configure the HP BPI SOA PE Adapter to talk to your SOA PE installation.

- Within the HP BPI Administration Console, you configure a connection to your HP BPI SOA PE Adapter.

- You make sure the HP BPI Service Adapters component is running, as this is the component that handles the BPI side of the SOA PE connection. This communication uses the HP Software Web Services interface.

- When defining a flow within the HP BPI Modeler you specify the names of the SOA PE Business Services that you wish to use. The HP BPI Modeler calls the HP BPI Service Adapters component (which, in turn, calls the HP BPI SOA PE Adapter) to determine whether these services exist as Business Services within SOA PE.

- You deploy your flow to the HP BPI Business Impact Engine.

- The BPI Service Adapters component polls the HP BPI SOA PE Adapter to get status details about the SOA PE Business Services. Any status changes for these Business Services are passed through to the HP BPI Business Impact Engine.

Configuration Instructions

Configuring the HP BPI SOA PE integration requires some set up on both the SOA PE server and the HP BPI server. Even if you have installed HP BPI on the same server as SOA PE, these steps are still required:

On the HP BPI Server

You configure the SOA PE connection details using the HP BPI Administration Console:

1   Select **Operational Service Sources** from the Navigation tree in the Administration Console. The right-hand pane shows a list of Operational Service Sources.

2   From the right-hand pane, select the **Add** button to add a new Service Source for the SOA PE adapter instance that you have created. You are presented with the Service Oriented Architecture Manager Source Properties dialog box.

3   Enter values for the properties of the SOA PE Service Source. The properties are fully described in the HP Business Process Insight Administration Guide:

— Service Source Name

— Description (Optional)

— Product Name, which is set as Service Oriented Architecture Manager. The interface allows for additional Service Sources to be added in future versions of BPI.

— Host name

— Port

— Status Event Poll Interval In addition, you can configure a Web Proxy for your Web Services connection if you have on

4   Click the **OK** button when you modifications are complete.

5   Make sure that the **Enabled** check box is selected in the column next to the new service source entry.

6   Click the **Apply** button to apply your changes

7   Move to the Component Status screen and stop and restart all the BPI components. The configuration is now complete and you can access SOA PE business services from HP BPI.

On the SOA PE Server

You need to install the HP BPI SOA PE Adapter and configure it to talk to your SOA PE installation. Once the BPI SOA PE Adapter is installed and running, it is available to HP BPI.

Prerequisites for HP BPI- SOA PE Adapter Installation

The SOA PE adapter needs be installed only if you want to use SOA PE as a source of operational events. The adapter can be installed on any Windows machine and configured to access the machine where SOA PE is running.

You install the SOA PE Adapter files from the zip archive provided on the HP BPI distribution media.

You can configure and start only one instance of the SOA PE adapter for each installation. Starting multiple instances of the adapter from the same installation can have unpredictable results.

If you want to run multiple instances of the adapter on one machine, you can achieve this by installing the adapter multiple times and starting one adapter instance from each installation.

You need the following information available to install and set up the adapter on the SOA PE machine.

Information	Notes
The name of the SOA PE Web Service that Manager Web Service that contains the catalog of web services that it exposes.	This is the SOA PE Web service that starts with the following string: WsmfServiceCatalog... The SOA PE Web services are listed at: http://hostname:port/wsmf/services where: - hostname, is the host name of the machine where SOA PE is running. This is described as the next parameter in this table. - port, is the port number used by SOA PE to publish its Web Services. This is described more fully later in this table.
Hostname of the machine where SOA PE is running	If no hostname is specified, a value of local host is assumed.
Port number used by SOA PE to publish its Web Services	If no port number is specified, a value of 5002 is used.

Port Number used to publish the SOA PE adapter as a Web Service. This is the Axis port number as installed with the adapter.	If no port number is specified, a value of 18097 is used.

Installing the HP BPI- SOA PE Adapter

Complete the following steps to install the adapter:

1 Locate the zip archive file on the SOA PE distribution media:

```
cd-root\i386\soam-adaptor.zip
```

2 Copy the zip archive file to the machine where you want to install the SOA PE adapter.

3 Create a new directory for the SOA PE Adapter zip archive file.

4 Unpack the zip archive file into the directory that you have created in the previous step.

5 Set the path for the Java Home directory for the adapter in the environment variable `SOAMADAPTER_JAVA_HOME`. Open a Command Window and enter:

```
set SOAMADAPTER_JAVA_HOME=java-install-dir
```
where `java-install-dir` is the location of your Java installation, for

example: `c:\program files\java\jdk1.5.0_08`

6 From a Windows Command Window, locate the following script to configure and start the adapter:

7 From the directory where the script is located, run the following script in a new Command Window to configure and start the adapter as follows:

```
runAdapter.bat
start runAdapter -csoa-svs-catalog -h hostname -swsvs-port -aadport
```

where:

`-c soa-svs-catalog` (required parameter)

`-c` takes a parameter `soa-svs-catalog`, which is the name of the SOA PE Web Services Catalog. This is the catalog name that you identified starting with the string `WsmfServiceCatalog`.

`-h hostname` (optional parameter)

`-h` takes a parameter, which is the fully qualified host name of the machine where SOA PE is running. This parameter is optional and if it is not specified, a value of localhost is assumed.

`-s wsvs-port` (optional parameter)

`-s` takes a parameter, which is the port number used by SOA PE to publish its Web services. This parameter is optional and if it is not specified a value of 5002 is used.

`-a adport` (optional parameter)

`-a` takes a parameter, which is the port number used by Axis to publish the SOA PE Adapter as a Web Service. This parameter is optional and if it is not specified, a value of 18097is used.

An instance of the SOA PE Adapter is now installed, configured and started. Do not close the Command Window where you started the adapter, or you shut down this adapter instance.

You also need to configure the adapter as an operational service source for HP BPI. You do this using the Administration Console as described in the HP Business Process Insight Administration Guide.

You can check that the adapter is configured and running at any time using the following URL:

`http://hostname:18097/axis/services/SOAMAdapter?wsdl` where hostname is the name of the machine where the adapter is installed. If the browser returns an error page, then the adapter is not running.

Stopping the SOA PE Adapter

You stop the adapter using `CTRL/C` in the Command Window where you started the adapter.

SOA PE Adapter Log Files

The SOA PE Adapter logs errors and warnings in the following log file:

`adapter-install-dir/data/log`

Troubleshooting HP BPI Integration

If the BPI Dashboard returns an error similar to the following when you attempt to link to a SOA PE-defined Service, it is likely to be because you are using a version of the Business Process Dashboard that does not recognize the Service definition:

`File: dash1-1_error.gif`

In this example, you are running a Business Process Dashboard based on HP BPI version 01.01. The integration with SOA PE was introduced in HP BPI version 2.10. As a result, the HP BPI Dashboard version 01.01 does not understand the SOA PE Service and is not able to render it.

If you want to show SOA PE Services, you need to upgrade your Business Process Dashboard to version 02.10.

6

Service Performance Monitoring

Integration with HP Diagnostics

SOA PE integrates with HP Diagnostics to monitor service performance. When you install SOA PE Broker, the installer installs the Diagnostics probe. The Diagnostics probe enables you to integrate SOA PE Broker with the HP Diagnostics Server and use Diagnostics for service performance monitoring and collection of service performance metrics. The Diagnostics probe runs on port number 40000. You can configure the Broker to use the Diagnostics Server after installing the SOA PE Broker.

The following diagram illustrates how the Diagnostics probe works with SOA PE Broker.
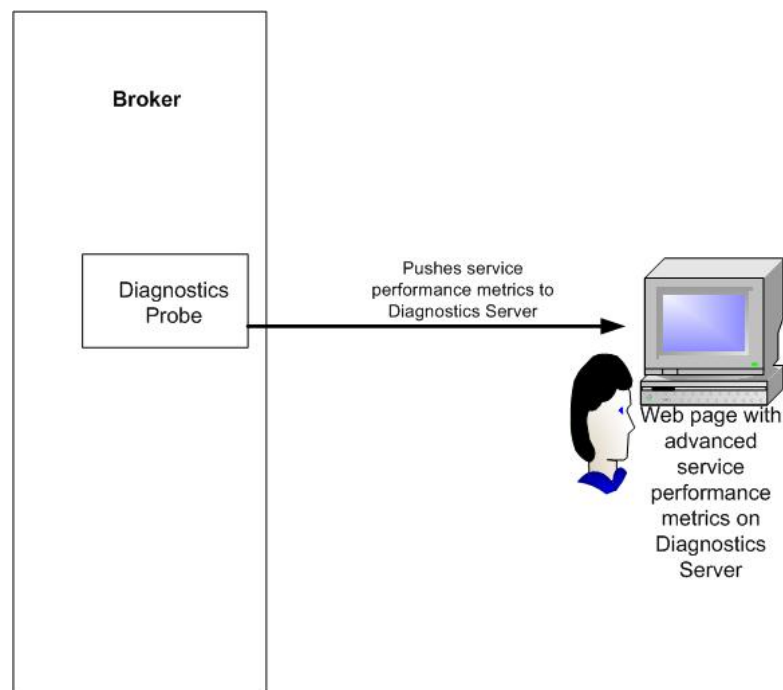
In the diagram, the SOA PE Broker is configured to integrate with Diagnostics Server. Therefore, the Diagnostics probe sends the service performance metrics from the Broker to the Diagnostics Server. This integration enables you to view the service performance metrics. See the HP Diagnostics documentation for more information about HP Diagnostics.

Configuring a Java Agent with the Diagnostics Server

You can refer to the *HP Diagnostics Enterprise Edition* documentation for configuring a Java agent with the Diagnostics Server.

## Viewing Service Performance Using the Diagnostics Probe

After installing SOA PE and configuring the policy enforcement points, you can view the service performance metrics from SOA PE user interface by performing the following steps:

1   Log on to the SOA PE user interface and click **Policy Enforcement Points** from the **View** drop-down menu on the left pane. The Policy Enforcement Intermediary Group page opens.

2   Click the policy enforcement intermediary group from the displayed list. The Contained Policy Enforcement Intermediary Instances page opens.

3   Click **View Metrics** to view to the Diagnostics page. This displays the Diagnostics page shown below.

Note: The default credentials for diagnostics is as follows:

— User ID: admin

— Password: admin

4   Click **Begin Profiling** present on the left-hand corner of the Diagnostics page if this is the first time you are viewing the performance status of the policy enforcement intermediary instance. This step is necessary for the service performance data collection to start.
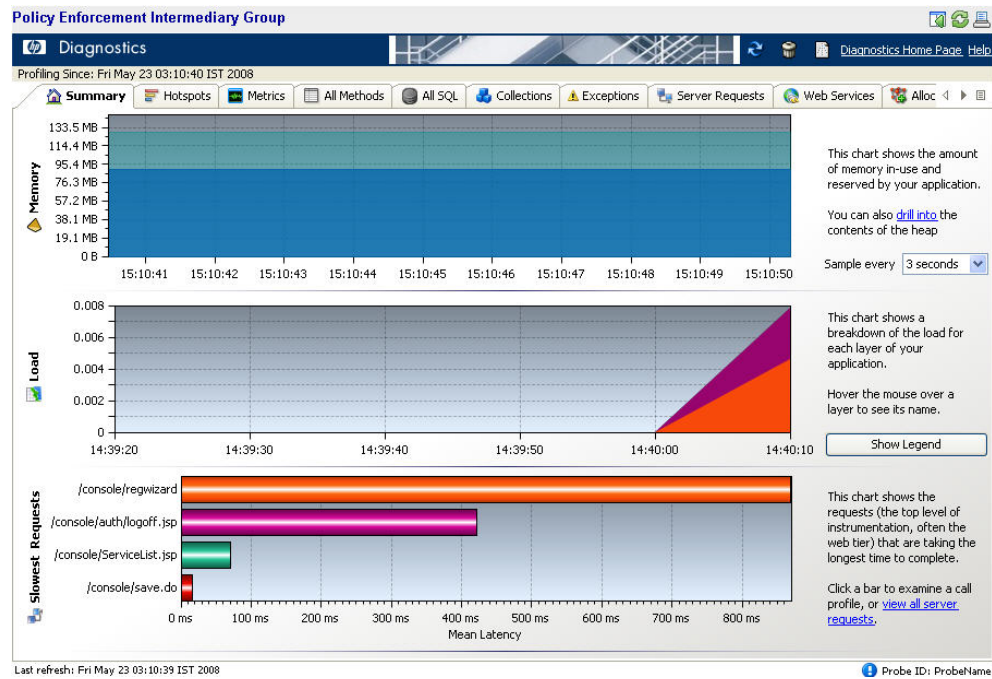
Enabling or Disabling the Probe Agent

You can do as follows to enable or disable the probe agent on various operating system platforms after installing SOA PE.

Note: By default the probe agent is enabled in SOA PE.

Microsoft Windows

To disable the Diagnostics probe, you must make sure that the command `rem` is not specified at the beginning of the set MIP_OPTS= line in the mipserver.bat file present in the <install_dir>\bin\win32 directory.

You can add the command `rem` at the beginning of the line to enable the Diagnostics probe.

UNIX and Linux

To disable the Diagnostics probe, you must make sure that the command # is not specified at the beginning of the `MIP_OPTS=` line in the mipserver.bat file present in the <install_dir>\bin\unix directory.

You can add the command # at the beginning of the line to enable the Diagnostics probe.

Windows Service

If you have installed the SOA PE Server or the Broker as a windows service, you can do as follows to enable/disable the Diagnostics probe for the service.

To disable the Diagnostics probe, you must make sure that the command rem is not specified at the beginning of the set `SM_MIP_OPTS=` line in the service-manager.bat file available under the <install_dir>\bin\win32\services directory. You can add the command `rem` at the beginning of the line to enable the Diagnostics probe.

Troubleshooting Diagnostics Integration

- What do I do if clicking the View Metrics link on the Policy Enforcement Intermediary Group page does not display the Diagnostics page?

  Make sure that the Diagnostics probe is running on port 40000. You can verify this in the `<entry name="com.hp.probe.jetty.port">40000</entry>` line in the mipServer.xml file in the `<install_dir>\conf\networkservices` directory. <install_dir> signifies the directory in which you installed SOA PE.

- How do I make sure that my Diagnostics Probe is working correctly?

  Verify that you are able to see the inbound or outbound calls, Web services information, instance tree, exceptions, SOA PE Broker layer information, and so on in the profiler mode on the Diagnostics page. You can also verify the `log/<probe_id>/detailreport.txt` file to identify the SOA PE Broker classes and methods that are instrumented successfully.

- How do I configure the Diagnostics profiler port if the configured port is already in use?

  - Update the entry `<entry name="com.hp.probe.jetty.port">40000</entry>` in the mipServer.xml file present at the following location to `<install_dir>\conf\networkservices \mipserver.xml` to use a new port number.

  - Update the entry `jetty.port = 40000` and `jetty.max.port = 40000` in the `webserver.properties` file at the following location `<install_dir> \MercuryDiagnosticsProbe\JavaAgent\DiagnosticsAgent\etc`

Mapping SOA Services Model to Registry

The SOA PE's assets that are published to a registry can be leveraged by any application that can integrate with the registry. This section provides a reference of how the assets are mapped in registry.

In particular, this section includes mappings for:

- Web services
- Technical policies

SOA PE works with only Systinet registry. You must synchronize the Systinet Registry taxonomies with the HP SOA Systinet before configuring the registry settings in SOA PE.

Define **Managed Endpoint** represents the service managed by a Web Service Management System. It represents a proxy of the Functional Endpoint.

Functional Endpoint represents the service exposing functionality managed by a WSMS. The service is deployed on an application server.

Overview

When you publish a business service, the web services contained in that business service along with the policies associated to it are published to the registry.

Role of Registry in the SOA

SOA implementations use registry as a system of record. The positioning of the role of registry technology in an SOA has evolved over the past few years since its inception. Originally, the registry was conceived as a central discovery point for design-time and run-time reuse.

However, most recent thinking in this direction is that if the registry is used as the only way to offer and discover Web services in the SOA, it provides an excellent control point to achieve Governance during various stages of development, deployment, and runtime management. Business stakeholders and enterprise architects define various policies that must be adhered to in the enterprise SOA. These policies are captured and attached to various entities in UDDI.

The registry is used to achieve the following:

- **Reuse** — capture meta-data about Web services as well as other technology assets so that effective search capabilities from various environments may be written against the registry.

- **Policy Definition** – capture various policy definitions that provide the ability for a business person or enterprise architect to mandate policies on various entities.

- **Capture SOA Environment Model** – capture various entities participating in an SOA; their relationships; and some meaningful subset of state information about these entities. This information is used to create a standardized/normalized information store that is used to support various IT governance processes.

- **Integration** – the agreement of various participants in the SOA to settle upon ontologies that are linked together in the registry creates very good potential for different participants to consume and populate information out of registry for their own narrow domains. The registry then provides a central store to create the ultimate spider that links together all this information.

Technical Policy Mapping

Technical policies defined in SOA PE are published to registry according to : Web Services Policy Attachment, Version 1.2.

According to this specification, reusable policy expressions are registered in registry as distinct TModels. SOA PE publishes all policies except routing and load balancing policy as distinct TModels. An example TModel is as follows:

```
<tModel tModelKey="uddi:469fef70-ac75-11dc-a342-3b4e75e1a340"
deleted="false" xmlns="urn:uddi-org:api_v3">

  <name>SchemaPolicy</name>

  <description>SchemaPolicy</description>

  <overviewDoc>

    <description>WS-Policy Expression</description>

<overviewURL>http://nt11812.asiapacific.hpqcorp.net:5002/bse_refresh/
PolicyExpression.jsp?cwcPopup=true&amp;policy=SchemaPolicy</overviewU
RL>

  </overviewDoc>

  <categoryBag>

    <keyedReference
tModelKey="uddi:schemas.xmlsoap.org:policytypes:2003_03"
keyName="policy" keyValue
```

```
<keyedReference
tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03"
keyName="Policy Expression for SchemaPolicy"
keyValue="http://nt11812.asiapacific.hpqcorp.net:5002/bse_refresh/Pol
icyExpression.jsp?cwcPopup=true&amp;policy=KiranSchemaPolicy"/>

    <keyedReference
tModelKey="uddi:systinet.com:soa:model:taxonomies:policyTypes"
keyName="Runtime"
keyValue="uddi:systinet.com:soa:model:taxonomies:policyTypes:runtime"
/>

    <keyedReference
tModelKey="uddi:systinet.com:soa:model:taxonomies:associatedApplicati
on" keyName="Other"
keyValue="uddi:systinet.com:soa:model:taxonomies:associatedApplicatio
n:other"/>

  </categoryBag>

</tModel>
```

Apart from the taxonomies mentioned in the WS Policy Attachment Specification, additional taxonomies are published to indicate the policy type as runtime and associated application as not 'Policy Manager'

Web Service Mappings

When a web service is published to registry, the information published to registry depends on whether the web service was originally created in SOA PE or Systinet.

For web services created in SOA PE initially, the following is published :

The WSDL for the proxy web service. The WSDL is published according to: *Using WSDL in a UDDI Registry, Version 2.0.2 – Technical Note*. The following UDDI entries are created:

— UDDI business service for the WSDL service

— UDDI binding template for the WSDL port

— UDDI tModel for each binding and port type

- The WSDL for the proxy Web service. The Web service WSDL is published to the UDDI registry if it does not exist in the registry already. It is published according to the technical note cited above.

- Relationships. The relationships are published as keyed references in the UDDI business service category bag. The following types of relationships are published:

  — **Reusable Policies** – A UDDI business service contains a keyed reference to indicate the reusable policies attached to the web services. Local Policy Reference taxonomy as mentioned in WS Policy Attachment specification is used to indicate this relationship

  — **Non Reusable Policies** – A UDDI business service contains a keyed reference to indicate the web service specific policies attached to it. Remote Policy Reference taxonomy as mentioned in WS Policy Attachment specification is used to indicate this relationship.

NOTE: For discovered services from Systinet, the UDDI business service representing the web service is expected to be already registered in registry. Only the policy association relationship and the binding template information is updated in registry

For discovered services from Systinet, the managed endpoint is published as per GIF. The figure shown below explains the mapping,
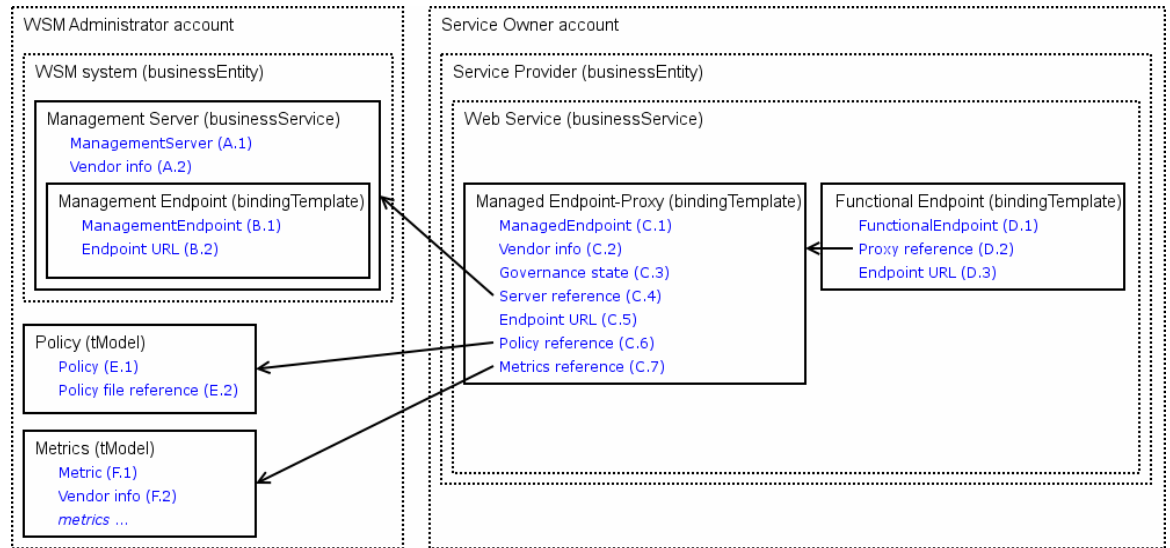
While publishing the managed endpoint following steps are followed by SOA PE,

- The functional service's bindingTemplate (BT1) is copied to a new bindingTemplate (BT2) contained by the same business service

- bindingTemplate (BT2) is updated with a reference to bindingTemplate (BT1) using "uddi:systinet.com:management:proxy-reference" taxonomy

- BT1's access point is updated with the proxy endpoint

- Both binding templates are updated with additional categorizations

An example is shown below:

```
<businessService
  serviceKey="uddi:example.com:myService:bs"
  businessKey="...">
  <name>My service</name>
  <bindingTemplates>
    <bindingTemplate
      bindingKey="uddi:example.com:myService:bt"
      serviceKey="uddi:example.com:myService:bs">
      <accessPoint URLType="http">http://example.com/myServiceProxy</accessPoint>
      <tModelInstanceDetails>
       <tModelInstanceInfo>
        ...
        </tModelInstanceInfo>
        ….

      </tModelInstanceDetails>
      <categoryBag>
        <keyedReference
```

```
            tModelKey="uddi:systinet.com:management:system"
            keyName="Management System"
            keyValue="HP SOA PE"/>
          <keyedReference
            tModelKey="uddi:systinet.com:management:type"
            keyName="Management entity type"
            keyValue="managedEndpoint"/>
<keyedReference
          tModelKey="uddi:systinet.com:management:state"
          keyName="Governance state"
          keyValue="managed"/>

     <keyedReference
        tModelKey="uddi:systinet.com:management:url"
        keyName="URL from AccessPoint"
        keyValue="http://example.com/myServiceProxy"/>
    </categoryBag>
  </bindingTemplate>

  <bindingTemplate
    bindingKey="uddi:example.com:myService:functionalEndpoint"
    serviceKey="uddi:example.com:myService:bs">
    <accessPoint URLType="http">http://example.com/myService</accessPoint>
    <tModelInstanceDetails>
      ...
    </tModelInstanceDetails>
    <categoryBag>
      <keyedReference
        tModelKey="uddi:systinet.com:management:system"
        keyName="Management System"
        keyValue="HP SOA PE"/>
      <keyedReference
        tModelKey="uddi:systinet.com:management:type"
        keyName="Management entity type"
        keyValue="functionalEndpoint"/>
      <keyedReference
        tModelKey="uddi:systinet.com:management:proxy-reference"
        keyName="Proxy reference"
        keyValue="uddi:example.com:myService:bt"/>
    </categoryBag>
  </bindingTemplate>
</bindingTemplates>
<categoryBag>
  <keyedReference
      tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03"
      keyName="Policy Expression for myServiceProxyRoutePolicy1"

keyValue="https://soamanagerhost:5003/bse_refresh/PolicyExpression.jsp?cwcPopup=true&policy=myService
ProxyRoutePolicy1" />

    <keyedReference
      tModelKey="uddi:schemas.xmlsoap.org:localpolicyreference:2003_03"
      keyName="SchemaValidation"
      keyValue="uddi:a8666990-ac99-11dc-9cdd-a32db2519cdc" />

    <keyedReference
      tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03"
      keyName="Policy Expression for myServiceProxyLoadBalancingPolicy"

keyValue="https://soamanagerhost:5003/bse_refresh/PolicyExpression.jsp?cwcPopup=true&policy=myService
ProxyLoadBalancingPolicy" />
```

```
    </categoryBag>
</businessService>
```

For services created in SOA PE the proxy binding template is published according to: *Using WSDL in a UDDI Registry, Version 2.0.2 – Technical Note*

An example is shown below,

```
<businessService serviceKey="uddi:37bd5bd0-ac9a-11dc-9cdd-a32db2519cdc"
businessKey="uddi:systinet.com:demo:hr" xmlns="urn:uddi-org:api_v3">

  <name>myServiceProxy</name>

  <bindingTemplates>

    <bindingTemplate bindingKey="uddi:37e83c60-ac9a-11dc-9cdd-a32db2519cdc"
serviceKey="uddi:37bd5bd0-ac9a-11dc-9cdd-a32db2519cdc">

      <accessPoint
useType="http">http://soamanagerhost:9032/myServiceProxy/myServiceSoapSoapBinding</accessPoint>

      <tModelInstanceDetails>

        <tModelInstanceInfo tModelKey="uddi:37783c80-ac9a-11dc-9cdd-a32db2519cdc">

          <instanceDetails>

            <instanceParms>myServiceSoapPort</instanceParms>

          </instanceDetails>

        </tModelInstanceInfo>

        <tModelInstanceInfo tModelKey="uddi:83814670-a944-11dc-bc79-f05f9301bc77"/>

      </tModelInstanceDetails>

    </bindingTemplate>

  </bindingTemplates>

  <categoryBag>

    <keyedReference tModelKey="uddi:uddi.org:wsdl:types" keyName="WSDL Entity
type" keyValue="service"/>

    <keyedReference tModelKey="uddi:uddi.org:xml:namespace" keyName="XML
namespace" keyValue="http://wsm.hp.com/myService"/>

    <keyedReference tModelKey="uddi:uddi.org:xml:localName" keyName="XML local
name" keyValue="myServiceProxy"/>

    <keyedReference
tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03"
keyName="Policy Expression for myServiceProxyRoutePolicy1"
keyValue="https://nttiju.asiapacific.hpqcorp.net:5003/bse_refresh/PolicyExpression.jsp?cwcPopup=true&amp;policy=myServiceProxyRoutePolicy1"/>
```

```
        <keyedReference tModelKey="uddi:schemas.xmlsoap.org:localpolicyreference:2003_03"
        keyName="SchemaValidation" keyValue="uddi:a8666990-ac99-11dc-9cdd-
        a32db2519cdc"/>

        <keyedReference
        tModelKey="uddi:schemas.xmlsoap.org:remotepolicyreference:2003_03"
        keyName="Policy Expression for myServiceProxyLoadBalancingPolicy"
        keyValue="https://soamanagerhost:5003/bse_refresh/PolicyExpression.jsp?cwcPopup=tru
        e&amp;policy=myServiceProxyLoadBalancingPolicy"/>

      </categoryBag>

</businessService
```

A

Appendix A Product Compatibility Matrix

Product Compatibility Matrix

The following table lists the HP Software products compatible for integration with SOA PE. The table also lists the versions compatible for the integration with SOA PE.

Product	Version
HP Business Process Insight	2.10
HP SOA Systinet	3.00
HP Diagnostics	7.5

Index

A

attachments, 3-5

audit
    architecture, 4-1
    database, 4-1
    messages, 4-1
    service, 4-1

audit integration, 4-1

B

BaseXmlHandler interface, 3-5

billing, 4-1

Broker Configurator, 3-3

brokered services
    add custom handler, 3-7
    definition, 3-3
    jar, 3-3
    manually implement, 3-3

BSE, 2-3

business services, 2-3

C

custom policy handler, 3-5
    add, 3-7

D

database
    message table, 4-3
    message trace table, 4-1
    schema, 4-1

database integrations, 2-3

document overview, 1-1

F

forecasting, 4-1

G

general integration, 2-2

H

handler. *See* policy handler

I

interfaces
    BaseXmlHandler, 3-5
    startup tasks, 3-1

J

Java API integrations, 2-3

M

management integration
    database, 2-3
    general, 2-2, 3-1
    Java API, 2-3, 2-4
    web services, 2-2

message table, 4-3

message trace table, 4-1

N

non-repudiation, 4-1

P

policy handler
    add custom, 3-7
    custom, 2-3, 3-5
    definition, 3-5

R

reports, 2-3, 3-1, 4-3, 4-5

S

security customization, 2-4

server.xml, 3-2

service.wsdl, 3-3

service.xml, 3-3

SLA
    reports, 3-1
    reports example, 4-3, 4-5