Artix™ ESB

AmberPoint Integration Guide

Version 5.1, December 2007

Making Software Work Together™

Updated: January 29, 2008

Contents

List of Figures

Preface

What is covered in this book

Artix supports integration with the AmberPoint Service-Oriented Architecture (SOA) management system. This guide explains how to integrate Artix solutions with AmberPoint. It applies to Artix applications written using C++, JAX-RPC (Java APIs for XML-Based Remote Procedure Call), and JAX-WS (Java APIs for XML-Based Web Services).

Who should read this book

This guide is aimed at system administrators using AmberPoint to manage SOA environments, and developers writing SOA applications. Administrators do not require detailed knowledge of the technology that is used to create distributed enterprise applications.

This book assumes that you already have a working knowledge of AmberPoint. For more information, see http://www.amberpoint.com.

Organization of this book

This book contains the following chapter:

- Chapter 1 describes the architecture of the Artix integration with AmberPoint.
- Chapter 2 explains how to configure integration with the Artix AmberPoint Agent, and shows examples from the Artix AmberPoint integration demo.

The Artix Documentation Library

For information on the organization of the Artix library, the document conventions used, and where to find additional resources, see Using the Artix Library.

CHAPTER 1

Artix AmberPoint Integration

Artix provides support for integration with the AmberPoint SOA management system. This chapter describes two approaches to integrating Artix services with AmberPoint.

In this chapter

This chapter includes the following sections:

AmberPoint Proxy Agent

Overview

There are two possible approaches to integrating Artix with the AmberPoint SOA management system:

- Using the AmberPoint Proxy Agent.
- Using IONA's Artix AmberPoint Agent.

AmberPoint Proxy Agent architecture

AmberPoint provides the AmberPoint Proxy Agent, which acts as a proxy for Web service endpoints by making the service endpoint WSDL available to the service consumer (client). Figure 1 shows a simple AmberPoint Proxy Agent architecture:

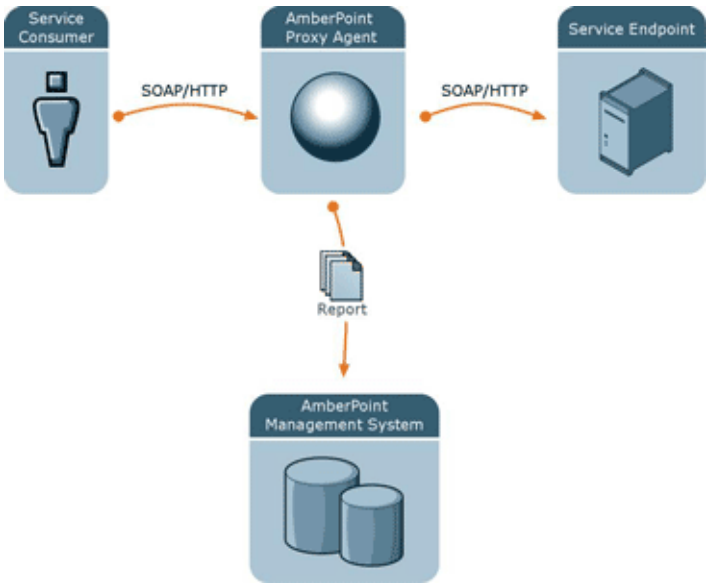

Figure 1: *AmberPoint Proxy Agent Integration*

In this architecture, the following restrictions apply:

- All messages between the service consumer and service endpoint must be routed through the AmberPoint Proxy Agent.
- All messages must use SOAP over HTTP.
- The service consumer is unaware of the back-end service endpoint, and views its relationship as being with the proxy only.

If you can work within these limits, the AmberPoint monitoring and management features can be used out-of-the box with Artix. However, if you require a more flexible integration (for example, with increased performance and scalability), you should use the Artix AmberPoint Agent.

AmberPoint Proxy Agent in a service network

Figure 2 shows the AmberPoint Proxy Agent deployed in a service network with multiple service consumers and service endpoints.

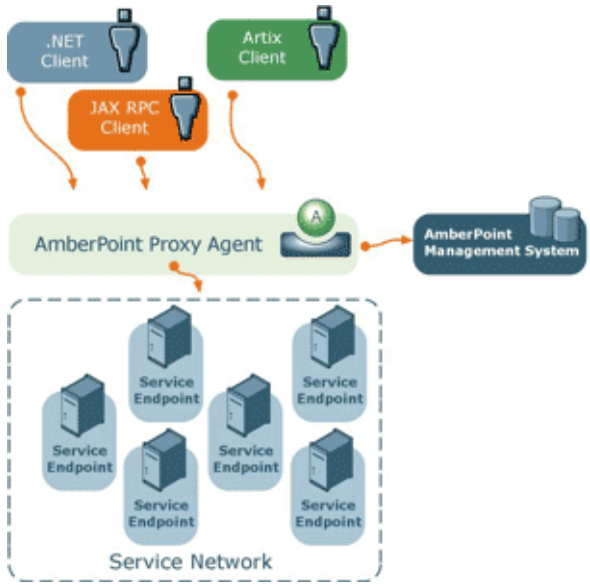

Figure 2: *AmberPoint Proxy Agent Service Network*

Because all messages are routed through the AmberPoint Proxy Agent, the additional network hops may impact on performance. In addition, the proxy involves the risk of a single point of failure.

If these are important issues for your system, you should use the Artix AmberPoint Agent instead.

Further information

For information on using the AmberPoint Proxy Agent, see the AmberPoint product documentation.

IONA Artix AmberPoint Agent

Overview

The Artix AmberPoint Agent enables Artix endpoints to be discovered and monitored by AmberPoint. This is the recommended approach to integrating Artix services with AmberPoint, and can be used with Artix services implemented in C++, JAX-RPC, JAX-WS, and scripting languages.

The Artix AmberPoint Agent can be deployed with Artix endpoints that use SOAP over HTTP to enable reporting of performance metrics back to AmberPoint. The Artix AmberPoint Agent offers significant benefits over the AmberPoint Proxy Agent. For example, these include increased performance and scalability, dynamic discovery, and the use of callbacks. This section describes the Artix AmberPoint Agent in detail.

Artix AmberPoint Agent architecture

Figure 3 shows how Artix can be integrated with AmberPoint using the Artix AmberPoint Agent.

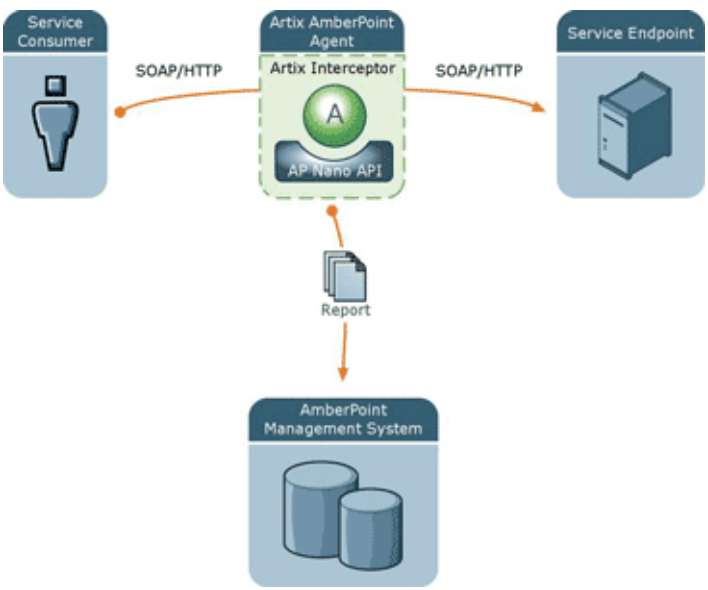

Figure 3: *Artix AmberPoint Agent Integration*

The main components in this architecture are:

- "Artix AmberPoint Agent"
- "Artix interceptor"
- "Artix service endpoints"
- "Service consumers"
- "AmberPoint SOA Management System"
- "AmberPoint Nano Agent API"

Note: Integration with the Artix AmberPoint Agent currently applies to SOAP over HTTP, and services that have one endpoint only.

Artix AmberPoint Agent

An Artix AmberPoint Agent consists of components developed by IONA and AmberPoint (the Artix interceptor, and the AmberPoint Nano Agent API). You can deploy multiple agents into your SOA network to capture data for the AmberPoint management system. Artix AmberPoint Agents gather performance data for all Artix endpoint types, as well as normal Web service endpoints.

Deployment modes

Artix AmberPoint Agents can be deployed in different ways in your system, for example:

- *Embedded in Artix consumers intercepting traffic*. This is suitable if Artix is deployed on the client side only, and the service endpoints do not support AmberPoint. This requires configuration for the consumer only.
- *Embedded in Artix service endpoints intercepting traffic*. This is suitable if Artix is used to implement the service endpoint. This works even when the consumers are third party products. This requires configuration for the service endpoint only. This is the most common and recommended approach, as shown in Figure 4.
- *Deployed as standalone Artix intermediaries (proxies) on your service network*. This option is suitable if you do not want touch your existing system and you do not want to update your endpoints or consumers. This approach is also necessary if Artix is not deployed at either the consumer or service endpoints.

Figure 4: *Artix AmberPoint Agent Embedded in Service Endpoint*

Artix interceptor

An Artix interceptor is deployed on the dispatch path of all messages exchanged between Artix service endpoints and consumers. It may be deployed in the same process as the consumer and/or the endpoint, or as an intermediary between the consumer and service.

The Artix interceptor captures all data in the dispatch path. This applies to the Artix C++ and Java core runtimes. The Artix interceptor then reports performance metrics using the AmberPoint nano agent API.

Artix service endpoints

An Artix service endpoint is a service built using Artix, and described using WSDL. The endpoint can be implemented using C++, JAX-RPC, JAX-WS, or even a scripting language, such as JavaScript. However, its main characteristic is that it can be described in WSDL, and classified as a service, which can therefore be consumed. The Artix AmberPoint Agent provides a WSDL contract describing the endpoint that is being monitored.

Service consumers

Service consumers are clients that consume service endpoints by exchanging messages based on the service interface. Consumers can be built using Artix, or any product that supports the technology used by the endpoint. For example, a pure CORBA client could be a consumer for a CORBA endpoint. A .NET client could be a consumer for an Artix SOAP endpoint.

AmberPoint SOA Management System

In this document, AmberPoint is the general term used to describe the system in which all performance metrics are stored and viewed. For the purposes of this document, all interactions are made using the AmberPoint Nano Agent API, and the AmberPoint graphical tools are used to view the Artix data. This simplifies the architecture of AmberPoint for the sake of this discussion.

AmberPoint Nano Agent API

The AmberPoint Nano Agent API is a Java public API provided by AmberPoint that enables customers to monitor their endpoints. This is the API that Artix uses to notify AmberPoint of the existence of the service endpoint. Artix also uses the AmberPoint nano agent API at runtime to report performance metrics about a previously registered endpoint.

The AmberPoint Nano Agent API enables the Artix interceptor to do the following:

- Allow dynamic discovery of new Artix endpoints without manual registration of the endpoints by the user. This registration process assumes that the Artix interceptor has the required configuration for the nano agent to contact AmberPoint. When the Artix AmberPoint Agent becomes active, it uses the Nano Agent API to register a new endpoint.
- Allow periodic reporting of messages using the Artix interceptor. These reports contain performance data about the endpoint and the messages being exchanged.

Artix AmberPoint Agent in a service network

Figure 5 shows the Artix AmberPoint Agent deployed in a service network with multiple service consumers and service endpoints.

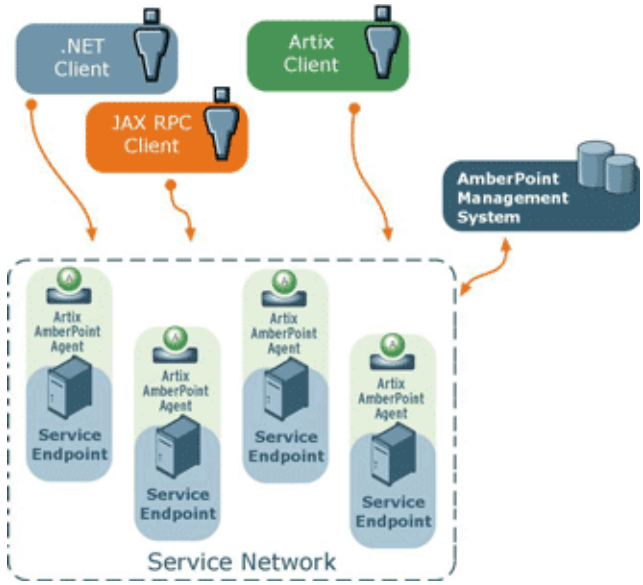

Figure 5: *Artix AmberPoint Agent Service Network*

This loosely-coupled architecture has the following benefits:

- Because the Artix AmberPoint Agent is collocated and embedded in the service endpoint, there are no additional network hops, so performance is maximized.
- Unlike with the AmberPoint Proxy Agent, there is no risk of a single point of failure, so reliability and scalability are also improved.
- An Artix AmberPoint Agent can be embedded into an Artix router.This enables it to dynamically discover and monitor the Artix service endpoints and consumers that the router creates and manages.
- Because the client is aware of the back-end service endpoint, the use of callbacks is supported.

Supported AmberPoint features

The Artix AmberPoint Agent enables the use of the following AmberPoint features:

- Dynamic discovery of Artix clients and services using SOAP over HTTP.
- Monitoring of Artix client and service invocations, and reporting them back to AmberPoint.
- Mapping Qualities of Service (QoS) to customer Service Level Agreements (SLAs).
- Monitoring of Artix invocation flow dependencies, which enables AmberPoint to draw Web service dependency diagrams.
- Centralized logging and performance statistics.

Further information

For detailed information on using AmberPoint features, see the AmberPoint product documentation.

CHAPTER 2

Configuring the Artix AmberPoint Agent

This chapter explains how to set up integration with IONA's Artix AmberPoint Agent, and shows examples from the Artix AmberPoint integration demos.

In this chapter

This chapter includes the following sections:

Installing AmberPoint

Overview

Artix supports integration with version 5.1 of the AmberPoint SOA management system. This section explains how to install AmberPoint to enable integration with the Artix AmberPoint Agent.

Installation steps

When installing the AmberPoint runtime, perform the following steps:

1. In the AmberPoint installation wizard, choose a suitable HTTP port number for the J2EE application server in which the AmberPoint server will be deployed (for example, `9090`).
2. AmberPoint comes bundled with Tomcat application server, so for the demo purposes, choose to install Tomcat.
3. Select **Deploy AmberPoint into the container**.
4. Select **Install a Java VM specifically for this application**.
5. Select **Deploy a new sphere with the SOA Management System**. This deploys the persistence runtime into the J2EE application server, and configures it to use the embedded Tomcat HSQL relational database management system.
6. You can also install AmberPoint sample Web services, but these are not required.
7. Provide a user name and password with administrative privileges (for example, `admin`/`admin`).
8. When installation is complete, copy the AmberPoint Nano Agent Server into the deployment directory of the application server. For example, for Tomcat, use the following command:

```
copy AP_InstallDir/add_ons/socket_converter/apsocketconverter.war
    AP_InstallDir/server/webapps
```

If you are not using Tomcat, use the vendor's visual tools to deploy `apsocketconverter.war` into the application server.

Configuring AmberPoint for Artix Integration

Overview

This section explains how to configure the AmberPoint SOA management system for integration with Artix. This section applies to Artix applications written in C++, JAX-RPC, and JAX-WS.

Starting the AmberPoint Server

When you have completed the AmberPoint installation steps, run the AmberPoint server using Window's **Start** menu.

Alternatively, execute the following script:

Windows	*AP_InstallDir*\server\bin\startup.bat
UNIX	*AP_InstallDir*/server/bin/startup.sh

You can see how your application server starts up and deploys the AmberPoint server in the log files in the *AP_InstallDir*/server/logs directory.

Configuring the AmberPoint Nano Agent Sever

When the application server has started and deployed all the AmberPoint .war files, perform the following steps:

1. Open a web browser and specify the following URL:
   http://hostname:port/apasc/
2. Login using the admin user name and password that you provided when installing AmberPoint.
3. When logged in, click **Network**|**Infrastructure** in the tabbed menu. This displays a list of registered **Deployments** with this application server's container.

4. Ensure that one of the deployed items is named `apsocketconverter` and has a green button beside it This indicates that the AmberPoint Nano Agent Server has been successfully deployed and is ready to be configured.
5. In the left pane, click the **Register** button.

   From the drop-down menu, select **Message Source**|**Simple Message Source**: This displays the **Register Message Source** form.
6. In the **Register Message Source** form, enter the following:

Name	`Artix Message Source`
Type of Message Source	`File`
Start At	`At present`
Location	*`AmberPointInstallDir`*`\server\amberpoint\apsocketconverter\logdir`

   The source **Name** can be any string value. The **Location** specifies the location of the log file for incoming messages. The default **Criteria for this policy** applies this message source to all active services that this AmberPoint system is aware of.
7. Without modifying the **Criteria for this policy,** click **Preview Services** to see which services this message Source applies to. If you have no services currently registered, only one service named **MonitorEnabler** is displayed.
8. Click the **Go** button at the top left of the screen, and wait until the **Policy Status** is `Applied`.
9. Return to a command window to build an Artix AmberPoint demo (see "Configuring Artix C++ Services for AmberPoint Integration" on page 28 or "Configuring Artix Java Services for AmberPoint Integration" on page 33).

Configuring the AmberPoint port

If the default AmberPoint Nano Agent Server port (`33333`) does not suit your setup, change the following attributes to the new port number:

- `messageLogWriter logLocation` in your Artix `apobserver.configuration` file.
- `messageLogReader logLocation` in:

```
AP_InstallDir/server/webapps/apsocketconverter.war@/WEB-INF/
   application/resources/readerConfig.xml
```

Whenever you update values in the Artix `apobserver.configuration` file, you must restart the services already being monitored by the Artix AmberPoint Agent for the changes to take effect.

If you update the Nano Agent Server port, you may need to restart the application server for changes to take effect (except for those servers that support hot deployment).

For example, these settings appear as follows in the Artix `apobserver.configuration` file:

```
...
<ap:messageLogWriter
   logWriterImplClass="com.amberpoint.msglog.socketimpl.SocketLogWriter"
      logName="{hostname}" <!-- default = localhost -->
      logLocation="{port}" <!-- default = 33333 -->
      syncEverySoManyEntries="50">
</ap:messageLogWriter>
  ...
<ap:hostMapper algorithm="asSent" urlProperty="ap:requestURL"/>
  ...
<ap:hostMapper algorithm="asSent" urlProperty="ap:wsdlUrl"/>
  ...
```

Configuring Artix C++ Services for AmberPoint Integration

Overview

This section explains how to configure Artix C++ and JAX-RPC services to support the Artix AmberPoint Agent. It describes Artix AmberPoint demo configuration settings in detail. However, if your AmberPoint installation and demo run on the same host, you do not need to make any configuration changes to run the demo. If you wish to run the demo now, skip this section, and see the `readme.txt` in the following directory:

```
ArtixInstallDir/Version/cxx_java/samples/integration/amberpoint
```

This `amberpoint` demo is based on the `.../samples/routing/content_based` demo, with some modifications to enable Artix and AmberPoint integration.

Configuring the AmberPoint Nano Agent plug-in

You must enable the AmberPoint Nano Agent plug-in for the Artix runtime. For example, the configuration scope in which the demo servers run includes an Artix plug-in named `ap_nano_agent`. This is loaded into the Artix runtime, and enables discovery and monitoring by AmberPoint of services and consumers running inside Artix processes.

```
demos {
    content_based {
      orb_plugins = ["xmlfile_log_stream", "soap", "at_http", "ap_nano_agent"];
    ...
   }
 ...
}
```

In this demo, there are three server instances, each exposing the same interface but running under different service and endpoint name pairs. These are as follows:

```
{TargetService1, TargetPort1}
{TargetService2, TargetPort2}
{TargetService3, TargetPort3}
```

Configuring the Artix router

To enable router support, you must also add the AmberPoint Nano Agent plug-in to the router's configuration. For example, the demo configuration scope in which the Artix router runs includes additional configuration for the Artix `routing` plug-in. Its `orb_plugins` list includes the `ap_nano_agent` plug-in, which enables the router's endpoints and consumers to be discovered and monitored by AmberPoint.

```
demos {
  content_based {
    ...
    router {
      orb_plugins = [ "xmlfile_log_stream", "ap_nano_agent", "routing" ];
      plugins:routing:use_pass_through="false";
     ...
    }
  }
}
```

The `ap_nano_agent` plug-in must precede the `routing` plug-in. This is because the Artix AmberPoint Agent must register itself in the interceptor chain before the routing plug-in instantiates and activates the services that it manages.

Setting `plugins:routing:use_pass_through` to `false` disables passing data through the router without parsing. The `ap_nano_agent` plug-in requires that the underlying payload is parsed in the Artix type format.

Configuring the consumer hostname

`plugins:ap_nano_agent:hostname_address:publish_hostname` specifies the form in which the Artix AmberPoint Agent resolves the host address that an Artix service consumer (proxy) runs on. This variable takes the following values:

`unqualified`	The host name in short form, without the domain name (*`hostname`*).
`ipaddress`	The host name in the form of an IP address (for example, `123.4.56.789`). This is the default.
`canonical`	The host name takes a fully qualified form (*`hostname.domainname`*).
`true`	same as `unqualified`
`false`	same as `ipaddress`

`plugins:ap_nano_agent:hostname_address:local_hostname` is an arbitrary string used as the client hostname instead of trying to resolve it using the underlying IP runtime. This is undefined by default.

To report the correct service consumer address invoking to an Artix service monitored by this agent, specify the following setting in the client and server configuration scope:

```
plugins:bus:register_client_context="true";
```

Configuring the service hostname

The server-side host name resolution is driven by the specific transport. Because the HTTP transport is the only one currently supported the following variables must be configured:

- `policies:soap:server_address_mode_policy:publish_hostname`
- `policies:at_http:server_address_mode_policy:publish_hostname`

Possible values are the same as those for `plugins:ap_nano_agent:hostname_address:publish_hostname`.

These variables specify the format that a service endpoint address is published to service consumers. AmberPoint discovers Artix services by consuming a published WSDL contract. It correlates the address in the WSDL with the inflow of log messages that describe operations invoked on an endpoint. This means that you must synchronize these configuration values with the configuration values of the AmberPoint Client Nano Agent.

Configuring the AmberPoint hostname

The default Artix hostname resolution setting is `ipaddress`, which is the same as that for the configuration of AmberPoint Client Nano Agent. However, if you change the Artix hostname resolution, you must also update the AmberPoint Client Nano Agent configuration file. For example:

ArtixInstallDir`/`*Version*`/cxx_java/etc/amberpoint/5.1/nanoagent/conf/apobserver.configuration`

To update the hostname resolutions setting, open the file in a text editor and find the two occurrences of the `hostMapper algorithm` attribute.

You must update the value of `hostMapper algorithm` attribute if you change the value of `policies:soap:server_address_mode_policy:publish_hostname` and `policies:at_http:server_address_mode_policy:publish_hostname` configuration variables.

The equivalent AmberPoint values are as follows:

Artix publish_hostname variable	AmberPoint hostMapper algorithm
`ipaddress`	`useIpAddr` or `asSent`
`canonical`	`useFQN` or `asSent`
`unqualified`	`asSent`

To avoid updating the AmberPoint Nano Agent Client configuration each time you change the Artix configuration, simply use `hostMapper algorithm="asSent"`.

If you are running your Artix services and the AmberPoint Nano Agent Server on different machines, you must also update the `messageLogWriter logName` attribute to point the host name or IP address where the Nano Agent Server is running.

Configuring the AmberPoint port

If the default AmberPoint Nano Agent Server port (`33333`) does not suit your setup, you can update your AmberPoint configuration file to the new port number. For more details, see "Configuring the AmberPoint port" on page 27.

Viewing Artix services in AmberPoint

When you run the demo, and start the Artix router and servers, and make client invocations to the router, these calls are in turn forwarded on to the servers.

AmberPoint dependency diagrams

While the demo is running, in the AmberPoint GUI, select the **Network|Services|Dependencies** screen. AmberPoint tracks the call flow, as it happens, between Artix services with the Artix AmberPoint Agent in their runtime. The dependency flow diagram is a directed graph, and can be of any complexity. For example, a client makes three calls to the source service implemented by the router. Each call is routed to the intended destination service, defined by the routing rules. Each `TargetService` receives a single call out of the three made. And each dependency tracking is shown in relation to the service selected in the **Selector** list, which is referred as a primary service.

You can manually create dependencies between services using the AmberPoint tools if so desired. See the AmberPoint user documentation for details on what you can do with dependency diagrams (for example, using the **Network**|**Services**|**Dependencies** screen).

AmberPoint performance diagrams

You can use the AmberPoint **Performance**|**Activity** screen to view performance statistics. See the AmberPoint user documentation for details on what you can do with performance statistics.

AmberPoint logging policies

You can collect call logs by adding an AmberPoint logging policy using the **Exceptions**|**Services** screen. To add an AmberPoint logging policy, click the **Add Logging Policy** button at the top of the screen. This displays the **Add Policy** form,. Use this form to specify a meaningful name, and tune its parameters to your needs. If you wish to log messages for all available services, edit the policy rules at the bottom of this form.

When the log policy is created, you must wait until it is applied, like when you created a **Message Source** (see "Configuring the AmberPoint Nano Agent Sever" on page 25). After the log policy has been applied and turns green, send some more traffic using the demo. You can then watch the **Message Log** using the **Exceptions**|**Services**|**Message Log** tab.

Further information

There are many other AmberPoint features that you can use with Artix. For example, when AmberPoint has captured the Artix traffic, you can use its runtime to define customers and their SLAs, and map these SLAs to the services in the network. You can also create reactions (alerts) if an SLA violation has occurred and so on. See the AmberPoint user documentation for more details.

Artix AmberPoint demo

For more details on the Artix AmberPoint integration demo, see:

```
ArtixInstallDir\Version\cxx_java\samples\integration\amberpoint\README.txt
```

Artix C++ configuration

- Configuring and Deploying Artix Solutions, C++ Runtime
- Artix Configuration Reference, C++ Runtime

Configuring Artix Java Services for AmberPoint Integration

Overview

This section explains how to configure Artix JAX-WS services to support the Artix AmberPoint Agent. It describes Artix AmberPoint demo configuration settings. However, if your AmberPoint installation and demo run on the same host, you do not need to make any configuration changes to run the demo. If you wish to run the demo now, skip this section, and see the `readme.txt` in the following directory:

```
ArtixInstallDir/Version/java/samples/advanced/management/amberpoint
```

This `amberpoint` demo is based on `.../java/samples/basic/wsdl_first`, with some modifications to enable Artix and AmberPoint integration.

Server configuration

The Artix Java configuration mechanism uses the XML-based Spring Framework. The following code shows the server-side configuration taken from the `server.xml` file in the Artix `amberpoint` demo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- -->
<!-- Copyright (c) 1993-2006 IONA Technologies PLC. -->
<!--            All Rights Reserved. -->
<!-- -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

    <!-- wiring server life cycle listener for gathering the server's endpoint information -->
    <bean id="com.iona.cxf.management.amberpoint.ServerLifeCycleListenerImpl"
    class="com.iona.cxf.management.amberpoint.ServerLifeCycleListenerImpl">
        <property name="bus" ref="cxf" />
    </bean>
```

```
    <!-- wiring the Nano Agent Logger factory for writing logger to apsocketconverter -->
    <bean id="com.iona.cxf.management.amberpoint.nanoagent.NanoAgentLoggerFactory"
    class="com.iona.cxf.management.amberpoint.nanoagent.NanoAgentLoggerFactory">
        <property name="bus" ref="cxf" />
    </bean>

    <!-- wiring the Amberpoint integration feature-->
     <jaxws:endpoint name="{http://apache.org/hello_world_soap_http}SoapPort"
                  createdFromAPI="true">
        <jaxws:features>
          <bean class="com.iona.cxf.management.amberpoint.interceptor.InvocationMessageFeature"/>
        </jaxws:features>
     </jaxws:endpoint>
</beans>
```

This example shows the configuration setting for the server lifecycle listener, which gathers the server's endpoint information. It also shows how to log the server to AmberPoint. And finally, the `hello_world` service endpoint is configured for Artix AmberPoint integration, using the `jaxws:endpoint` attribute.

For details on how to make your configuration available to the Artix Java runtime, see "Configuring the AmberPoint hostname" on page 36.

Client configuration

The following code shows the client-side configuration taken from the `client.xml` file in the Artix `amberpoint` demo.

```
?xml version="1.0" encoding="UTF-8"?>
<!-- -->
<!-- Copyright (c) 1993-2007 IONA Technologies PLC. -->
<!--            All Rights Reserved. -->
<!-- -->
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jaxws="http://cxf.apache.org/jaxws"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://cxf.apache.org/jaxws http://cxf.apache.org/schemas/jaxws.xsd">

    <!-- wiring the Nano Agent Logger factory for writing logger to apsocketconverter -->
    <bean id="com.iona.cxf.management.amberpoint.nanoagent.NanoAgentLoggerFactory"
    class="com.iona.cxf.management.amberpoint.nanoagent.NanoAgentLoggerFactory">
        <property name="bus" ref="cxf" />
    </bean>

    <!-- wiring the Amberpoint integration feature-->
    <jaxws:client name="{http://apache.org/hello_world_soap_http}SoapPort"
                  createdFromAPI="true">
        <jaxws:features>
          <bean class="com.iona.cxf.management.amberpoint.interceptor.InvocationMessageFeature"/>
        </jaxws:features>
     </jaxws:client>

</beans>
```

This example shows how to log the client to AmberPoint. It also shows how the `hello_world` client is configured for Artix AmberPoint integration, using the `jaxws:client` attribute.

For details on how to make your configuration available to the Artix Java runtime, see "Configuring the AmberPoint hostname" on page 36.

Configuring the AmberPoint hostname

If you are running your Artix services and the AmberPoint Nano Agent Server on different machines, you must update the hostname in your AmberPoint Nano Agent Client configuration file. For example:

```
ArtixInstallDir/Version/java/samples/advanced/management/amberpoint/apobserver.configuration
```

You should update the `messageLogWriter logName` attribute to point the hostname or IP address where the AmberPoint Nano Agent Server is running.

Configuring the AmberPoint port

If the default AmberPoint Nano Agent Server port (`33333`) does not suit your setup, you can update your AmberPoint configuration file to the new port number. For more details, see "Configuring the AmberPoint port" on page 27.

Accessing Artix Java configuration

You can make your Artix Java configuration available to the Artix Java runtime in one of the following ways:

- Use one of the following command-line arguments to point to your XML configuration file:

```
-Dcxf.config.file.url=<myCfgURL>
-Dcxf.config.file=<myCfgResource>
```

  This enables you to save your XML configuration file anywhere on your system and avoid adding it to your `CLASSPATH`.
- Specify the XML configuration file on your `CLASSPATH`.
- Programmatically, by creating a bus and passing the configuration file location as either a URL or string, as follows:

```
(new SpringBusFactory()).createBus(URL myCfgURL)
(new SpringBusFactory()).createBus(String myCfgResource)
```

Demo examples

The Artix Java sample applications uses the command-line approach. For example, in the Artix AmberPoint demo, the following command is used to start the server:

```
start java -Djava.util.logging.config.file=%CXF_HOME%\etc\logging.properties
   -Dcxf.config.file=server.xml demo.hw.server.Server
```

The following command is used to start the client:

```
start java -Djava.util.logging.config.file=%CXF_HOME%\etc\logging.properties
   -Dcxf.config.file=client.xml demo.hw.client.Client .\wsdl\hello_world.wsdl
```

Viewing the demo in AmberPoint

You can use the following AmberPoint tools to view the demo application.

AmberPoint dependency diagrams

While the demo is running, in the AmberPoint GUI, select the **Network**|**Services**|**Dependencies** screen. AmberPoint tracks the call flow, as it happens, between Artix services with the Artix AmberPoint Agent in their runtime. The dependency flow diagram is a directed graph, and can be of any complexity. You can manually create dependencies between services using the AmberPoint GUI tools if so desired. See the AmberPoint user documentation for details on what you can do with dependency diagrams (for example, using the **Network**|**Services**|**Dependencies** screen).

AmberPoint performance diagrams

You can use the AmberPoint **Performance**|**Activity** screen to view performance statistics. See the AmberPoint user documentation for details on what you can do with performance statistics.

AmberPoint logging policies

You can collect call logs by adding an AmberPoint logging policy using the **Exceptions**|**Services** screen. To add an AmberPoint logging policy, click the **Add Logging Policy** button at the top of the screen. This displays the **Add Policy** form. Use this form to specify a meaningful name, and tune its parameters to your needs. If you wish to log messages for all available services, edit the policy rules at the bottom of this form.

When the log policy is created, you must wait until it is applied, like when you created a **Message Source** (see "Configuring the AmberPoint Nano Agent Sever" on page 25). After the log policy has been applied and turns green, send some more traffic using the demo. You can then watch the **Message Log** using the **Exceptions**|**Services**|**Message Log** tab.

Further information

There are many other AmberPoint features that you can use with Artix. For example, when AmberPoint has captured the Artix traffic, you can use its runtime to define customers and their SLAs, and map these SLAs to the services in the network. You can also create reactions (alerts) if an SLA violation has occurred and so on. See the AmberPoint user documentation for more details.

Artix AmberPoint demo

For more details on the Artix AmberPoint integration demo, see:

```
ArtixInstallDir/Version/java/samples/advanced/management/amberpoint\README.txt
```

Artix Java configuration

- Configuring and Deploying Artix Solutions, Java Runtime
- Artix Configuration Reference, Java Runtime

Spring Framework

www.springframework.org

Index