Tivoli. software

WebSphere. software

DataPower Architectural Design Patterns

Integrating and Securing Services Across Domains

Introduction to DataPower Services

Integration Services

Security Services

Mike Ebbers
Bill Barrus
Servais Bonazebi
Peter Daly
Charlton Lee

Redbooks

ibm.com/redbooks

International Technical Support Organization

DataPower Architectural Design Patterns: Integrating and Securing Services Across Domains

October 2008

SG24-7620-00

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

First Edition (October 2008)

This edition applies to Version 3.6 of the IBM WebSphere DataPower SOA Appliance firmware.

Contents

Notices . . . vii
Trademarks . . . viii

Preface . . . ix
The team that wrote this book . . . ix
Comments welcome . . . xi

Chapter 1. Introduction . . . 1
1.1 DataPower overview . . . 2
1.1.1 Intended audience . . . 2
1.1.2 Role of DataPower . . . 2
1.1.3 SOA . . . 3
1.1.4 Challenges in service-oriented integration . . . 3
1.1.5 Meeting SOA challenges . . . 4
1.2 IBM WebSphere DataPower SOA Appliance product line . . . 5
1.2.1 IBM WebSphere DataPower XML Accelerator XA35 . . . 5
1.2.2 IBM WebSphere DataPower XML Security Gateway XS40 . . . 6
1.2.3 IBM WebSphere DataPower Integration Appliance XI50 . . . 8
1.3 DataPower deployment examples . . . 8
1.4 Use cases in DataPower Redpaper publications . . . 9
1.5 Summary . . . 10

Chapter 2. DataPower services . . . 11
2.1 Overview of SOA services . . . 12
2.2 Introduction to the five core services of DataPower . . . 12
2.2.1 Configuration architecture . . . 13
2.2.2 Configuring and using DataPower . . . 15
2.3 Multi-protocol gateway service . . . 15
2.3.1 When to use . . . 17
2.3.2 Use case . . . 17
2.4 Web Service Proxy service . . . 18
2.4.1 When to use . . . 18
2.4.2 Use case . . . 19
2.5 XML firewall service . . . 19
2.5.1 When to use . . . 20
2.5.2 Use case . . . 20
2.6 Web application firewall service . . . 20
2.6.1 When to use . . . 21
2.6.2 Simple use case . . . 22
2.6.3 Host virtualization use case . . . 22
2.7 XSL Accelerator (Proxy) service . . . 22
2.7.1 When to use . . . 22
2.7.2 Use case . . . 23
2.8 An architectural approach to service selection . . . 23

Chapter 3. Web services and policy management . . . 25
3.1 Web services . . . 26
3.1.1 DataPower Web Service Proxy service . . . 27
3.1.2 Web services proxy summary . . . 29

3.2 Policy management. 30
3.2.1 SOA governance. 30
3.2.2 Web Service Policy Framework (WS-Policy Framework) 30
3.2.3 Configuring Web Service Policy 32
3.2.4 Policy management summary. 40
3.2.5 Resources 41
3.3 Service registries. 42
3.3.1 WebSphere Service Registry and Repository. 42
3.3.2 WSRR governance model. 43
3.3.3 The need to integrate WSRR with UDDI. 46
3.3.4 Querying WebSphere Service Registry and Repository 48
3.3.5 Dynamic endpoint lookup 51
3.3.6 UDDI service registries 53
3.3.7 Service registries summary. 54
3.3.8 Resources 54
3.4 Web services scenarios: A WSRR scenario 55
3.4.1 Request-response Web Service scenario. 55
3.4.2 Service level monitoring: XML variation scenario 62

Chapter 4. Security 69
4.1 XML threat protection 70
4.1.1 Security requirements in an IT environment 70
4.1.2 XML threats. 71
4.1.3 Network-level security 73
4.1.4 Message level security within XML context. 77
4.2 IBM SOA security offering. 79
4.3 WS-Security 80
4.3.1 Goals and requirements 81
4.3.2 WS-Security and DataPower 82
4.3.3 Encryption/decryption capabilities. 82
4.3.4 Digital signature 83
4.3.5 Other WS-* specifications supported 84
4.3.6 Exceptional added value. 86
4.3.7 Resources for further reading 88
4.4 Authentication, authorization, and audit (AAA) 89
4.4.1 Mechanisms supported by AAA 96
4.5 Integrating IBM DataPower and Tivoli for SOA Security. 103
4.5.1 Using WebSphere DataPower, Tivoli Security tools, and open standards 106
4.5.2 AAA using Tivoli Access Manager (TAM) 106
4.5.3 IBM Tivoli Access Manager for e-business. 107
4.5.4 AAA using LDAP Directory 108
4.5.5 Summary. 109
4.6 DataPower security scenarios. 109
4.6.1 Scenario one: DataPower typical security. 110
4.6.2 Scenario two: WebSphere DataPower as an XML firewall 111
4.6.3 Scenario three: DataPower as a Web application firewall 114
4.6.4 Scenario: Securing a WebSphere Message Broker 118
4.7 Summary. 123

Chapter 5. Integration patterns 125
5.1 Enterprise service bus implementation patterns and SOA 126
5.1.1 DataPower as an ESB 126
5.1.2 Application awareness in the ESB 128

5.1.3 Monitoring and managing 128
5.1.4 Routing 130
5.1.5 Protocol conversion 131
5.1.6 Message transformation 134
5.1.7 Securing messages 136
5.1.8 Connectivity 136
5.1.9 Summary. 144
5.1.10 Resources. 144
5.2 Integration to the heritage applications 145
5.2.1 WebSphere Transformation Extender (WTX) 145
5.2.2 DataPower and WebSphere MQ: Quality of service to the traditional 151
5.2.3 Service-enable CICS and IMS traditional applications with DataPower 160
5.2.4 Building message flows in WebSphere Message Broker 166
5.3 Web 2.0. 167
5.3.1 Web 2.0 overview 167
5.3.2 Sample Web 2.0 integration 168
5.3.3 Demonstration. 177
5.3.4 Summary. 178
5.4 DataPower scenarios 178
5.4.1 ESB scenario: WESB, WSRR, Tivoli for ABC Hotel 178
5.4.2 ESB scenario 2: XYZ Insurance 181
5.4.3 Food products corporation 183
5.4.4 Brokerage firm 187
5.4.5 Resources. 189

Chapter 6. Configuration management. 191
6.1 Introduction 192
6.1.1 File system directories and domains. 192
6.1.2 Devices, environments, and load balancers 192
6.1.3 Boot sequence for DataPower 193
6.2 Configuration options 194
6.2.1 WebGUI interface 194
6.2.2 Command-line interface 195
6.2.3 XML Management Interface 196
6.3 Package importing and exporting 198
6.4 Using a repository 199
6.5 IBM Tivoli Composite Application Manager System Edition 201

Appendix A. The enterprise service bus. 203
Definition of an enterprise service bus 205
Enterprise requirements for an ESB 207

Appendix B. XML security 209
XML threats 210
Specific types of XML attacks 210
Single-message xDOS attacks 210
Multiple-message XDoS attacks 211
Unauthorized access attacks 211
Data integrity/confidentiality attacks 211
Systems compromise attacks 211

Security services: Standards supported by DataPower 212
When to use DataPower for security 212
References. 213

Glossary 215

Related publications 217
IBM Redbooks publications 217
Other publications 217
Online resources 218
How to get Redbooks publications. 221
Help from IBM 221

Notices

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

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive, Armonk, NY 10504-1785 U.S.A.

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

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

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

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

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

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

alphaWorks®
CICS®
DataPower device®
DataPower®
DataStage®
DB2®
developerWorks®
IBM®
IMS™
Lotus®
Parallel Sysplex®
pureXML™
RACF®
Rational®
Redbooks®
Redbooks (logo) ®
System z®
Tivoli®
WebSphere®
Workplace™
Workplace Client Technology™
z/OS®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

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

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

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

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

Preface

IBM® WebSphere® DataPower® SOA Appliances are purpose-built network devices that offer a wide variety of functionality such as the securing and management of SOA applications, enterprise service bus (ESB) integration, and high-speed XSL execution. A hardened appliance, DataPower provides robust security features including tamper protection of the device itself.

This IBM Redbooks® publication is for application architects and other consultants who want to include DataPower appliances in their solutions for reasons of speed, security, or ESB integration. This book covers DataPower services, Web services, security, and integration strategies.

The team that wrote this book

This book was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center. These authors were onsite:

Mike Ebbers is a Certified Consulting IT Specialist and a Project Leader with the ITSO, where he has worked for 13 years. In his 34 years with IBM, Mike has worked on mainframe sales and support in the field, and has also developed and taught classes worldwide for IBM education.

Bill Barrus is a Senior Software Engineer in IBM Software Group's Business Partner Technical Strategy and Enablement organization. He began his IBM career doing mechanical design trade-off studies for the U.S. Navy F-14 avionics upgrade program, moved into mechanical computer aided engineering software, and most recently contributed to emerging business opportunity challenges in support of CATIA Engineering Analysis, Lotus® Workplace™ Client Technology - Micro Edition, and IBM WebSphere Voice Server. He is currently involved in providing consultation and support to IBM partners on various IBM products, including DataPower SOA Appliances.

Servais Bonazebi is an IT Mission Director working for Devoteam Solutions, an IBM Business Partner, located in France. Inside the Business Unit EBS, Servais leads the SOA, DataPower, and IT Governance offering. Prior to joining the Devoteam group, Servais worked for IBM France and Altran group on planning, designing, and implementing large-scale distributed systems. Servais has been in the IT industry for more than 17 years, with seven years as Enterprise Architect. He has a master's degree in computer engineering from CESI Aquitaine and a master's degree in business from ESC Lille, France.

Charlton Lee, an IBM Certified SOA Solutions Designer, is a technology leader with 20+ years of experience in architecture, design, development, and project management, engaging with large ASEAN and U.S. customers. Charlton has broad IBM product knowledge of the WebSphere and Rational® SOA product line, with an emphasis on DataPower. As a member of ASEAN ISSW, he is a consulting and education product lead for DataPower across the ASEAN region. He is a renowned IBM promoter who has recently led the ASEAN Software Group SOA practice. He has prior experience working for Oracle®, Capgemini, and KPMG in the U.S., an experience which that several consulting domains including IT and business strategy.

A special thanks to the following for their contributions:

Peter Daly works for IBM Software Services for WebSphere in the UK. A former physics teacher, he returned to academia to create software for Protein Crytallography, which is widely used in leading laboratories throughout the world. Peter also worked at an international laboratory in France and managed supercomputing facilities in the UK. In 1998 he joined IBM as a UNIX® Systems Administrator, then moved into Tivoli® Systems Management before taking on his current role (in 2004) as a WebSphere and DataPower consultant. Peter holds degrees in physics and mathematics/computer science. He is a Chartered Engineer and a member of the British Computer Society.

Srinvasan Muralidhara participated in the planning and review sessions of this book, as well as rewriting some sections of the text. He is an Advisory Engineer with 15 years of industrial experience, with nine years with IBM. He currently works on DataPower-related projects at the IBM WebSphere Technology Institute. He is widely experienced in SOA-related technologies in all tiers of the software development stack. He has studied SOA performance with DataPower appliances and has investigated integrating DataPower with other mid-tier and back-end traditional components such as WebSphere Application Server, MQ, CICS®, and IMS™ in the SOA context of reusing existing systems and enterprise modernization.

Figure 1 is a group picture of the onsite authors in Raleigh.

Figure 1 The authors: Servais, Mike, Charlton, and Bill

Thanks to the following IBMers for their reviews of this document:

Matt McLarty, Holger Reinhardt, Adolfo Rodriguez, Manuel Rohmann, and Gari Singh

Be an author

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

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

Find out more about the residency program. Browse the residency index and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our books to be as helpful as possible. Send us your comments about this book or other IBM Redbooks publications in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an e-mail to:

  redbooks@us.ibm.com

- Mail your comments to:

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

1

Introduction

IBM WebSphere DataPower SOA Appliances represent an important element in the holistic approach of IBM to service-oriented architecture (SOA). These appliances are purpose-built, easy-to-deploy network devices that simplify, secure, and accelerate your XML and Web services deployments while extending your SOA infrastructure. These appliances offer an innovative, pragmatic approach to harness the power of SOA. Through their use, you will be able to enhance the value of your existing application, security, and network investments.

The emergence and proliferation of XML and Web services has seen an explosion in the middleware infrastructure required to support them. An important component in this middleware architecture is the enterprise service bus (ESB), a collection of runtime components that provide services such as routing, transformation/bridging, management, security, and other control functions.

While XML and SOAP enable a rich application-aware communication, their emergence has resulted in challenges in terms of consumability, performance, and security. This book examines architectural patterns for using WebSphere DataPower SOA Appliances. Chapter 2, “DataPower services” on page 11, provides a look at the primary services in DataPower that are the building blocks for the succeeding patterns and includes guidance on the selection of services. We then cover Web services and governance (Chapter 3, “Web services and policy management” on page 25), security (Chapter 4, “Security” on page 69), integration patterns (Chapter 5, “Integration patterns” on page 125), and configuration management (Chapter 6, “Configuration management” on page 191).

1.1 DataPower overview

WebSphere DataPower SOA Appliances provide the ability to understand and act upon application data as it traverses the network. While this application awareness is not, in itself, a new networking concept, XML has accelerated its appeal and difficulty. That is, application awareness comes with many security, complexity, and performance challenges. As a result, a new genre of hardened software, hardware, and XML-centric appliances has arisen to bridge this gap.

These appliances focus on providing consumability, performance, and hardened security. They can extend the ESB into the network and also provide an SOA gateway for business-to-business integration. This introductory chapter gives an overview of IBM WebSphere DataPower SOA Appliances, their features and architecture, as well as a quick summary of the approach taken to use and deploy the appliances in the service-oriented architecture.

1.1.1 Intended audience

This publication is intended for architects who need to understand the scenarios, patterns, and use cases within which WebSphere DataPower SOA Appliances are deployed. The material in this document will help you to successfully deploy these appliances across many domains, resulting in secure and efficient implementations.

1.1.2 Role of DataPower

Unlike middleware running on traditional application environments, there is no one specific role that one can affix to DataPower. DataPower defies generic labeling. The reasons for this are:

- Hardened security: It can act purely as a security gateway.
- Appliance versatility: It is easy to add to the the network at various points to perform different functions.
- XML lingua franca: The promise of XML-speak is uniformity, simplicity, and transparency of handling data. Being XML-centric from core upwards gives DataPower the ability to adapt to different roles.
- Any-to-any transformation: The ability to transform any data format allows it to be an integration device.
- Multi-protocol support: This allows it to be a integration device or bridge for heritage applications.

All the above are bolstered by the ease-of-use factor. Consumability is a core competence of DataPower. DataPower is consumable at different levels:

- Installation: As a network appliance it can be up and running in literally a few minutes.
- Development cycle: On-board Web GUI based approach to creating and managing applications gets rid of the develop-deploy round trip development methodology.
- Development model: The modeling uses the building-block approach where the application is built using a collection of objects one on top of another. The granularity of these objects maps comfortably to architectural components of typical applications.

All the above enable WebSphere DataPower SOA Appliances to be used in a variety of ways, depending on the task at hand.

1.1.3 SOA

Service-oriented architectures enable the creation of composite applications that comprise reusable, loosely coupled service components. The technical foundation of an SOA is in the support for the eXtensble Markup Language (XML) and Web services built atop it. Using the SOAP Protocol, SOA clients can invoke services without explicit support for a wide variety of transport protocols and message formats. A SOAP facade in front of an existing service enables virtualization where clients invoke a virtualized version of the underlying service, thereby eliminating the need to understand intricate details of the service's implementation. In this context, XML enables data to be self-describing with explicit language support for common operations that manipulate the data. For example, the XPath language enables a consistent way to select data items from within an XML document. In SOA, service intermediaries can use XML and other application-layer data to route, secure, control, transform, or otherwise process service requests and responses, decoupled from the actual service implementation that fulfills each particular request.

1.1.4 Challenges in service-oriented integration

Though greatly appealing, the promise of loosely coupled, virtualized services in SOA comes at a price. As the data-centric complexity of XML and SOA operations has increased, traditional software-based middleware has struggled to keep up. In particular, software-based service intermediaries have emerged as natural extensions to traditional server-side service stack environments. Unfortunately, their success and impact has been inhibited by three fundamental challenges. As you will soon see, WebSphere DataPower SOA Appliances overcome these challenges.

Consumability

Often, middleware service stacks have an underlying software engine (generally, J2EE™) upon which a Web service hosting engine is built. In some sense, this camp of products has been built by joining together necessary components in an embedded fashion. For example, a J2EE servlet engine can be extended to receive SOAP over HTTP by providing a new Web service servlet. The Web service itself is deployed on this servlet. The result is a system built out of multiple software layers, each with its own configuration and maintenance requirements. Taken individually, each layer's requirement may prove tedious. Taken together, the collective set of installation and maintenance requirements often proves prohibitive. For example, patch upgrades that affect in layer in the stack of embedded software must be coordinated in a single atomic action. Further complicating this problem has been the traditional software industry's focus of favoring adding more function to a software product over increasing the usability of existing function.

These complexities associated with traditional software middleware built upon general-purpose computers affect other consumability aspects such as usability, learning-curve, trouble-shooting, deployment, and so on. The WebSphere DataPower SOA Appliance form factor (architecture and usability) greatly minimizes difficulties in these difficult areas, greatly enhancing overall consumability.

Security

The advent of SOA has created a common communication framework to understand and operate on application data like has never been seen. With self-describing XML, intermediaries are able to extract portions of the data stream and effect application-aware policies. Unfortunately, this has also enabled a new opportunity for malicious attacks. That is, as XML regularly flows from client to enterprise through IP firewalls without much impediment, the obvious place to attack is in the application data stream itself, the XML. While we are just beginning to understand the repercussions of these types of attacks, they are emerging. XML Denial-of-Service (XDoS) attacks seek to inject malformed or malicious XML into middleware servers with the goal of causing the server to churn away valuable cycles processing the malicious XML. Enterprise-ready application servers are susceptible to many of these types of attacks, leaving open a security hole that must be closed.

Performance

Another key challenge that has emerged with the adoption of XML is in the computational cost of XML processing. Computing on XML in traditional software-based middleware is orders of magnitude more costly (from the computational sense) than native data structures. XML must be parsed and fluffed into the native data structures of the local computer's architecture. Further, XML transformations exacerbate processing needs, as they require multiple passes through the XML structure and are highly sensitive to the transformation processing engine. Securing XML and SOA at the application (XML) level provides computational barriers that can require as much as 60 times the processing capability as plain XML (based on typical workloads).

Additionally, it is often prohibitive from a performance point of view to enable key requirements such as monitoring, auditing, and security. Customers end up sacrificing those functions to keep equipment cost from growing unwieldy.

Built from the ground up to perform SOA and integration workloads, DataPower is architected to handle data processing efficiently. It combines hardware offloading, pipelined processing, and streaming to create a high-performance SOA engine.

1.1.5 Meeting SOA challenges

The IBM WebSphere DataPower SOA Appliances family contains rack-mountable network devices that overcome many of the challenges facing SOA and XML today. At a high-level, the IBM WebSphere DataPower SOA Appliances offer:

- 1U (1.75-inch thick) rack-mountable, purpose-built network appliances.
- XML/SOAP firewalling, field-level XML security, data validation, XML Web services access control, and service virtualization.
- Lightweight and protocol-independent message brokering, integrated message-level security and fine-grained access control, and the ability to bridge mission-critical transaction networks to SOAs and ESBs.
- High performance, multi-step, wirespeed message processing, including XML, XSLT, XPath, and XSD.
- Centralized Web services policy and service-level management.
- WS-* standard support, such as WS-Security, SAML 1.0/1.1/2.0, portions of Liberty Alliance protocol, WS-Federation, WS-Trust, XKMS, Radius, XML Digital Signature, XML-Encryption, WSDM, WS-SecureConversation, WS-Policy, WS-SecurityPolicy, WS-ReliableMessaging, SOAP, WSDL, UDDI, and others.
- Transport layer flexibility, which supports HTTP/HTTPS, MQ, SSL, FTP, and others.
- Scalable, wirespeed, any-to-any message transformation, such as arbitrary binary, flat text, and XML messages, which include COBOL Copybook, CORBA, CICS, ISO 8583, ASN.1, EDI, and others.
- Consumable simplicity: An easy-to-install and easy-to-maintain network appliance that can satisfy both application and network operational groups, supporting current and emerging standards, as well as XML Web services standards out-of-the-box.

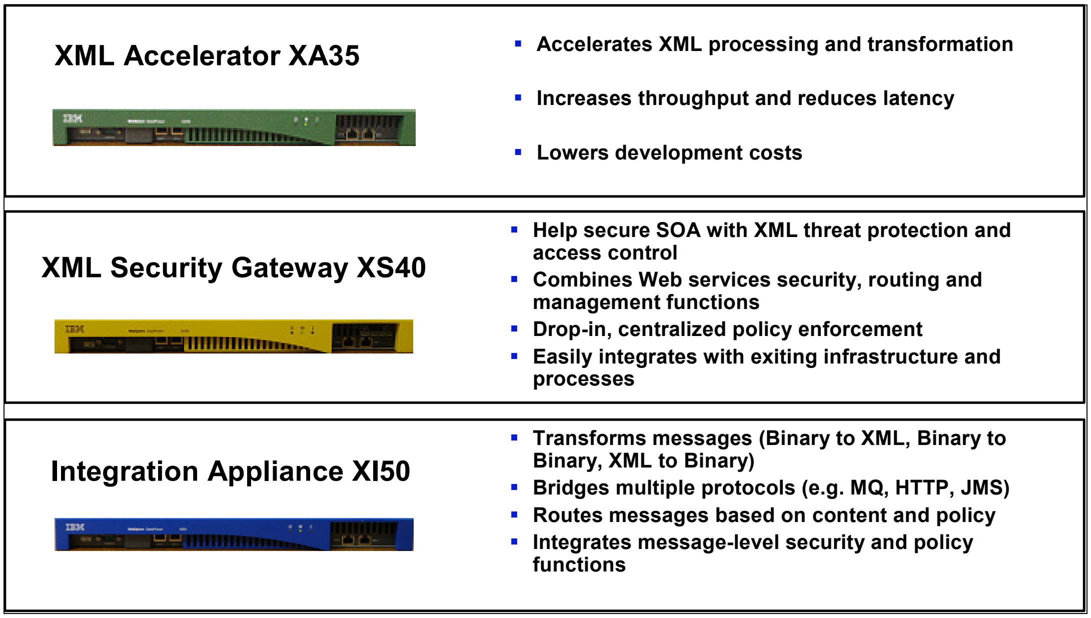
- Enhanced security: Key support includes, but is not limited to, XML/SOAP firewall and threat protection, field-level XML security, data validation, XML Web services access control, service virtualization, and SSL acceleration.
- Acceleration: Drop-in solution that can streamline XML and Web service deployments, helping lower total cost of ownership and accelerate return on your assets, as you continue to move to SOA. WebSphere DataPower SOA Appliances are purpose-built hardware devices capable of offloading overtaxed servers by processing XML, Web services, and other message formats at wirespeed.

1.2 IBM WebSphere DataPower SOA Appliance product line

The product line consists of three appliances. The higher model numbers are a functional super-set of lower-numbered appliances. When examining the model numbers, it may help to keep in mind that A stands for acceleration, S stands for security, and I stands for integration.

All appliances share a basic common engine as well as management/monitoring interfaces. More information about those interfaces is included in Chapter 2, "DataPower services" on page 11. The appliances also support a SOAP (WS-) management interface that enables programmatic access to the appliance's configuration. An Eclipse plug-in, included with the appliances, leverages this interface and provides IDE support. In addition, all appliances come with vast support for monitoring, event logging, and alerting. Supporting a wide variety of event levels and criteria (IP, TCP, HTTP, XML, SOAP), output can be directed to a number of targets including file systems, SNMP, SOAP endpoints, Common Base Event, the console, or a system cache.

The three types of WebSphere DataPower SOA Appliances are shown in Figure 1-1.



XML Accelerator XA35

- **Accelerates XML processing and transformation**
- **Increases throughput and reduces latency**
- **Lowers development costs**

XML Security Gateway XS40

- **Help secure SOA with XML threat protection and access control**
- **Combines Web services security, routing and management functions**
- **Drop-in, centralized policy enforcement**
- **Easily integrates with exiting infrastructure and processes**

Integration Appliance XI50

- **Transforms messages (Binary to XML, Binary to Binary, XML to Binary)**
- **Bridges multiple protocols (e.g. MQ, HTTP, JMS)**
- **Routes messages based on content and policy**
- **Integrates message-level security and policy functions**

Figure 1-1 IBM WebSphere DataPower SOA Appliance product line

1.2.1 IBM WebSphere DataPower XML Accelerator XA35

The most basic appliance, the XA35[1], delivers a number of routing and transformation features. It supports routing based on IP, TCP, HTTP, XML, and SOAP criteria. Routing tables can be defined (or queried) to determine the appropriate routing action. Load balancing algorithms such as least-used and round-robin are also available. Additionally, the XA35 can throttle or adjust request rate based on routing criteria to employ traffic shaping. From a transformation perspective, the XA35 provides basic XSLT processing. Built on the underlying premise of all appliances, special-purposed hardware converts one XML schema to another at wire speed. XSLT 1.0 and XPath 1.0 support (with some 2.0 support) is available. The appliance is able to use and apply internal as well as external schema.

This product can help speed up common types of XML processing by offloading this processing from servers and networks. It can perform XML parsing, XML Schema validation, XPath routing, Extensible Stylesheet Language Transformations (XSLT), XML compression, and other essential XML processing with wirespeed XML performance.

- Unmatched performance: DataPower's purpose-built message processing engine can deliver wirespeed performance for both XML-to-XML and XML-to-HTML transformations with increased throughput and decreased latency.
- Ease of use: The XA35 provides drop-in acceleration with virtually no changes to the network or application software. No proprietary schemas, coding, or APIs are required to install or manage the device. In addition, it supports popular XML Integrated Development Environments (IDEs) to help reduce the number of hours spent in the development and debugging of XML applications.
- Helps reduce infrastructure costs: Unlike simple content switches that only redirect business documents, the DataPower XML Accelerator XA35 fully parses, processes, and transforms XML with wirespeed performance and scalability to help reduce the need for stacks of servers. The XA35 also supports accelerated SSL processing to help further lessen the load on server software.
- Helps cut development costs: The XA35 can enable multiple applications to leverage a single, uniformed XML processing layer for all XML processing needs. By standardizing on high-performance hardware appliances, enterprises can deploy sophisticated applications while helping to eliminate unnecessary hours of application debugging and tuning for marginal performance gains.
- Intelligent XML processing: In addition to wirespeed processing, appliances support XML routing, XML pipeline processing, XML compression, XML/XSL caching, as well as other intelligent processing capabilities to help manage XML traffic.
- Advanced management: The XML Accelerator model XA35 provides real-time visibility into critical XML statistics such as throughput, transaction counts, errors, and other processing statistics. Data network level analysis is provided, and includes server health information and traffic statistics, as well as management and configuration data.

1.2.2 IBM WebSphere DataPower XML Security Gateway XS40

The DataPower model XS40[2]offers security capabilities, such as XML Encryption and XML digital signature processing, in addition to its accelerator capabilities. Supporting WSS 1.0, WS-Trust, and WS-SecureConversation, the XS40 is the foundation upon which Web Services Security can be deployed in an enterprise. Support exists for authorization (access) checking to offboard entities such as the Tivoli Access Manager, Tivoli Federated Identity Manager, RSA, Netegrity, Oblix, and more. There is partial SAML 2.0 support as well as the ability to extract and use security credentials from LDAP, RADIUS, and XKMS. But perhaps the most significant security feature of the XS40 is its firewalling capabilities. In addition to filtering input traffic (as is done in the XA35 routing capabilities), the XS40's firewall enables

[1] A for accelerator

[2] S for security

XML Denial of Service protection. Used in conjunction with schema validation and well-formedness checking, the XS40 is an integral part of an enterprise's network security.

This appliance provides a security-enforcement point for XML and Web services transactions, including encryption, firewall filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed logging, and includes:

- An XML/SOAP firewall: The DataPower XML Security Gateway XS40 filters traffic at wirespeed, based on information from layers 2 through 7 of the protocol stack, from field-level message content and SOAP envelopes to IP address, port/host name, payload size, or other metadata. Filters can be predefined with an easy point-and-click XPath filtering GUI, and automatically uploaded to change security policies based on the time of day or other triggers.
- XML/SOAP data validation: With its unique ability to perform XML Schema validation as well as message validation, at wirespeed, the XS40 ensures that incoming and outgoing XML documents are legitimate and properly structured. This protects against threats such as XML Denial of Service (XDoS) attacks, buffer overflows, or vulnerabilities created by deliberately or inadvertently malformed XML documents.
- Field-level message security: The XS40 selectively shares information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters.
- XML Web services access control: The XS40 supports a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, RADIUS, and simple client/URL maps. The XS40 can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.
- Service virtualization: XML Web services require companies to link partners to resources without leaking information about their location or configuration. With the combined power of URL rewriting, high-performance XSL transforms, and XML/SOAP routing, the XS40 can transparently map a rich set of services to protected back-end resources with high performance.
- Centralized policy management: The XS40's wirespeed performance enables enterprises to centralize security functions in a single drop-in device that can enhance security and help reduce ongoing maintenance costs. Simple firewall functionality can be configured via a GUI and running in minutes, and using the power of XSLT, the XS40 can also create sophisticated security and routing rules. Because the XS40 works with leading Policy Managers, it is an ideal policy execution engine for securing next-generation applications. Manageable locally or remotely, the XS40 supports SNMP, script-based configuration, and remote logging to integrate seamlessly with leading management software. Its emerging support for WS-Policy and WS-SecurityPolicy further augments these capabilities.
- Web services management/service level management: With support for Web Services Distributed Management (WSDM); Universal Description, Discovery, and Integration (UDDI); Web Services Description Language (WSDL); Dynamic Discovery; and broad support for service level management configurations, the XS40 natively offers a robust Web services management framework for the efficient management of distributed Web service endpoints and proxies in heterogeneous SOA environments. Service level management (SLM) alerts and logging, as well as pull and enforce policies, help enable broad integration support for third-party management systems and unified dashboards, in addition to robust support and enforcement for governance frameworks and policies.

1.2.3 IBM WebSphere DataPower Integration Appliance XI50

The XI50[3] adds integration capabilities to its accelerator and security abilities. It enables a key concept of any-to-any transformation where data can be received in any format over any protocol and be converted to any other format over any other protocol using DataPower core high-performance transformation technology. That is, in DataPower everything is a transformation. Supported protocols include HTTP, HTTPS, and MQ. Formats include SOAP/XML, EDI, CICS Cobol Copybook, Corba, ISO 8583, CSV, ASN.1, ebXML, and more.

This appliance provides transport-independent transformations between binary, flat text files and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography. This appliance can transform binary, flat-text, and other non-XML messages to help offer an innovative solution for security-rich XML enablement, ESBs, and mainframe connectivity.

- Any-to-any transformation engine: XI50 can parse and transform arbitrary binary, flat text, and XML messages, including EDI, COBOL Copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, DataPower's patented DataGlue technology uses a fully declarative, metadata-based approach.
- Transport Bridging: With support for a wide array of transport protocols, the XI50 is capable of bridging request/response flows to/from protocols such as HTTP, HTTPS, MQ, SSL, IMS Connect, FTP, and more.
- Integrated message-level security: XI50 includes mature message-level security and access control functionality. Messages can be filtered, validated, encrypted, and signed, helping to provide more secure enablement of high-value applications. Supported technologies include WS-Security, WS-Trust, SAML, and LDAP.
- Lightweight message brokering: Sophisticated multi-step message routing, filtering, and processing. Multiple synchronous and asynchronous transport protocols. Detailed logging and audit trail, including non-repudiation support.

For full product information about IBM WebSphere DataPower SOA Appliances, see:

http://www.ibm.com/software/integration/datapower/index.html

1.3 DataPower deployment examples

WebSphere DataPower SOA Appliances provide a robust, secure platform for middleware integration that can be deployed in a wide array of deployment scenarios to perform a wide variety of middleware use cases. This section highlights the most common scenarios and points to some use cases in our series of Redpaper publications, which are listed in 1.4, “Use cases in DataPower Redpaper publications” on page 9.

[3] I for integration

Figure 1-2 depicts common scenarios for deploying these appliances in the intranet, the demilitarized zone (DMZ), and a federated extranet (for example, a business partner). Of particular importance is DataPower's capability to pass even the most stringent requirements for enterprise DMZ deployment. Stated simply, the DataPower architecture is a secure environment with absolutely no Java™ on the appliance. Network ports are secured by default with no remote access beyond its command-line interface over the secure SSH protocol, Web Graphical User Interface (WebGUI) over HTTPS, and XML management APIs over HTTPS.

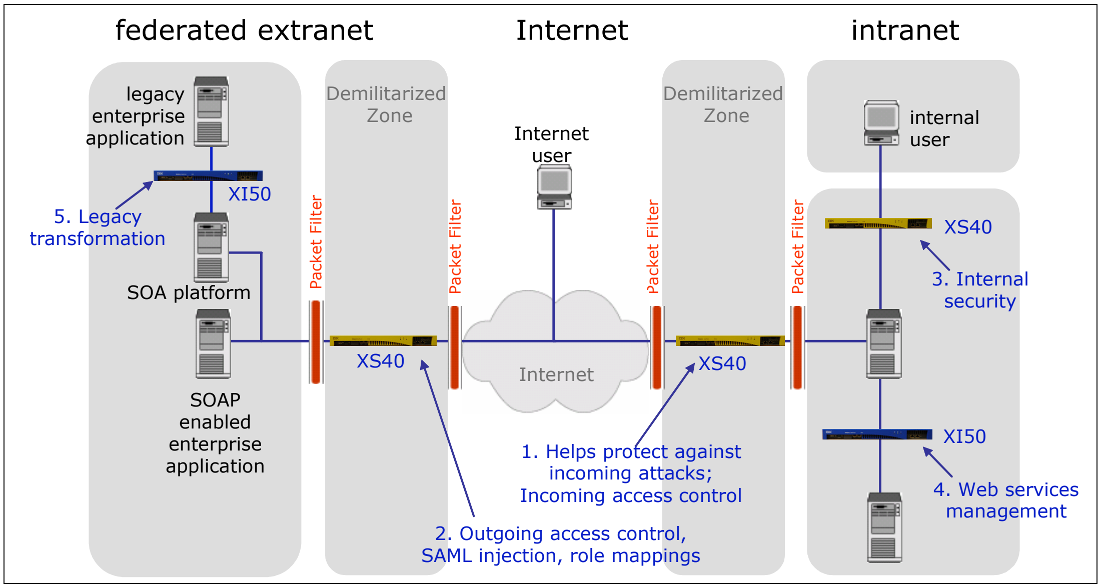


Figure 1-2 DataPower deployment scenarios

1.4 Use cases in DataPower Redpaper publications

A series of Redpaper publications is available with implementation details that help identify the circumstances under which WebSphere DataPower SOA Appliances should be deployed. In this series you will also find ways to integrate these appliances with other products, such as LDAP, WSRR, TAM, ITCAM SOA, ITCAM SE, and others. The content of these publications form a collection of well-documented use cases with step-by-step instructions on how configuration can be done using the DataPower WebGUI.

Here is the list of the four Redpaper publications and their URLs:

- *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327

  http://www.redbooks.ibm.com/abstracts/redp4327.html?Open

- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364

  http://www.redbooks.ibm.com/abstracts/redp4364.html?Open

- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*, REDP-4365

  http://www.redbooks.ibm.com/abstracts/redp4365.html?Open

- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366

  http://www.redbooks.ibm.com/abstracts/redp4366.html?Open

1.5 Summary

This introductory chapter provided an overview of the environment and challenges of service-oriented integration. We introduced the WebSphere DataPower SOA Appliance product line and described typical deployment scenarios. We pointed to detailed use cases available in IBM Redpaper publications.

Performance and scalability aspects are not covered in this book. DataPower is built from the ground up for performance in such difficult areas as security and XML processing. It is commonly accepted that DataPower outperforms most middleware components in their core areas. Indeed, since DataPower can take on so many roles, it is difficult to come up with standard benchmarks.

In the rest of this book we cover key features and uses of the appliances.

2

DataPower services

This chapter discusses the following primary services that are provided by the DataPower devices:

- Multi-protocol gateway
- Web Service Proxy
- XML firewall
- Web application firewall
- XSL Accelerator (Proxy)

After describing each service's features, functions, and usage scenarios, this chapter provides an architectural approach to service selection.

2.1 Overview of SOA services

Before we describe the features of the three WebSphere DataPower SOA Appliances, let us review the concepts of service-oriented architecture (SOA), since that is typically the context into which DataPower devices are installed. Within this SOA context, *services* are defined as network-accessible application interfaces, and are used as the building blocks of SOA. Web services—services built using XML-based standards over open protocols—offer flexibility and interoperability. Additionally, they offer the correct level of abstraction for serving as a control point for exercising various parameters of a robust SOA environment, such as security. The enterprise service bus (ESB) is the component or set of components that provides connectivity between service consumers and service providers in an SOA.

Within the DataPower context, the term *service* has a different meaning. A DataPower service is an object that offers a specific type of intermediary processing between two integrated endpoints, such as an SOA service consumer/provider pair. This chapter deals with the different kinds of services supported by DataPower and serves as the foundation for succeeding chapters.

2.2 Introduction to the five core services of DataPower

These capabilities are housed in DataPower's five core services that encapsulate operations performed on inbound traffic. Figure 2-1 lists the primary services with their accompanying icons. Note that the term services here refers to DataPower capabilities rather than SOA services.

- Multi-Protocol Gateway
- Web Service Proxy
- XML Firewall
- Web Application Firewall
- XSL Accelerator

Figure 2-1 DataPower primary services

All of these services use the same basic programming constructs, and thus their capabilities often overlap. For instance, both an XSL proxy and an XML firewall can perform XSL transformations. Table 2-1 describes which services are offered on each appliance. Each successive service has the capabilities of the previous one. The exception is the WS-Proxy, which supports features such as SLM that are not available in multiprotocol gateway.

Table 2-1 Services offered on each appliance

Core services offered	DataPower appliances	Typical scenarios
Multi-protocol gateway	XS40, XI50	Bridge request and response protocol differences. Multiple transports in and out.
WS-Proxy	XS40, XI50	Process WSDL described services.
XML firewall	XS40, XI50	Send and receive XML traffic over HTTP to and from XML-based applications.
Web application firewall	XS40, XI50	Protect heritage XML, SOAP, and B2B messages, non WSDL based Web services and non Web service applications.
XSL accelerator	XA35, rarely used on XS40 and XI50	Optimize XML/XSLT transformations.

2.2.1 Configuration architecture

Before going into the details of each primary service, it may be helpful to understand the configuration architecture of the DataPower device®. Each of the three available devices consists of layers of related objects. Service objects, such as an XSL proxy, XML firewall, or Web Service Proxy, occupy the top layer. These objects provide the coordinated runtime service needed to implement solutions to SOA requirements. A single device can run many services simultaneously.

At the next layer, processing policy objects and the many objects referenced by a processing policy provide message handling and routing services. These objects perform such tasks as message transformation, message filtering, authentication and authorization, cryptographic operations, error handling, and dynamic message routing. Any single service has only one processing policy. The processing policy, however, might have any number of rules. Each rule might contain any number of actions. Many actions might employ an XML stylesheet language transformation (XSLT). See Figure 2-2 for an illustration of this relationship.

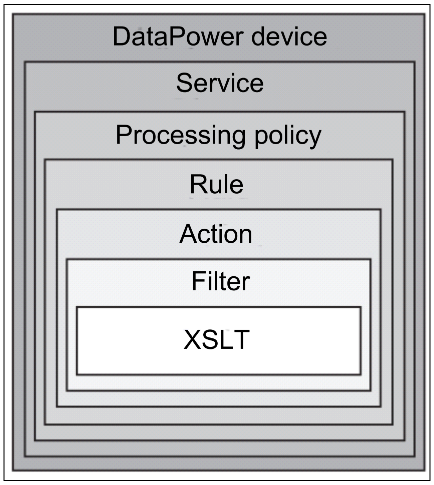


Figure 2-2 Basic configuration architecture

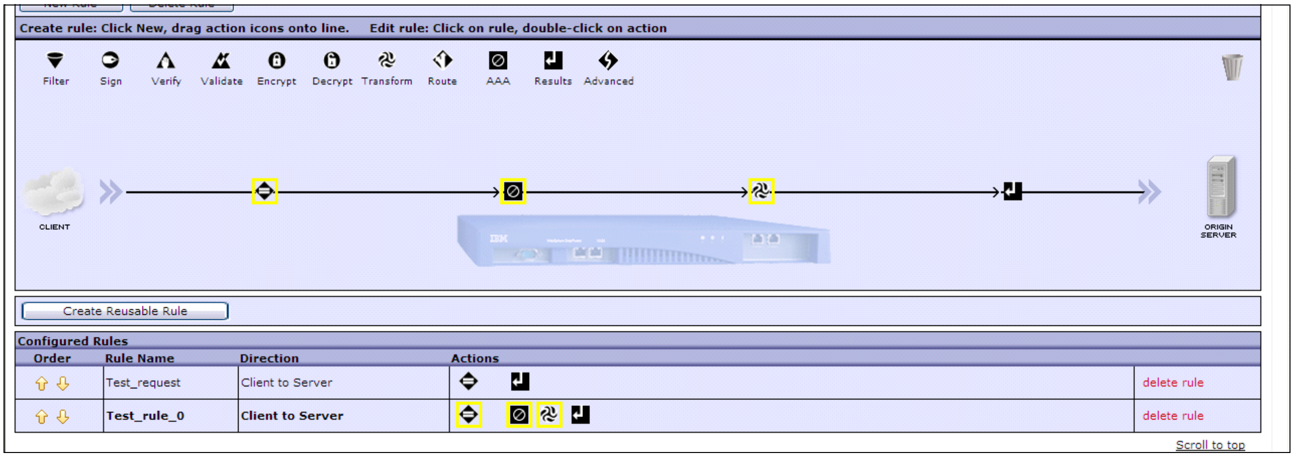
A wide range of objects provide ancillary or supporting capabilities to top-level service or processing policy objects. For example, a service might require the use of SSL communications, and thus the services of SSL-related objects. A processing policy might include an action that routes messages, and this routing action in turn depends upon a routing map object. A service might require the use of monitoring for threat protection, which in turn requires the configuration of one or more monitor objects. All services on the device might use one or more log target objects to deliver real-time updates to SNMP consoles running elsewhere in the enterprise.

At the most fundamental level, network and administration objects provide the configuration settings needed to connect the device to the physical network, enable the WebGUI or CLI interfaces, set user access policies, or create connectivity to other specialized network devices.[1]

[1] *WebSphere DataPower XML Integration Appliance XI50 WebGUI Guide*, Version 3.6.1, Second Edition (December 7, 2007) (http://www.ibm.com/support/docview.wss?rs=2362&uid=swg24014405)

2.2.2 Configuring and using DataPower

DataPower appliances provide a powerful intermediation architecture with a heavy emphasis on ease-of-use and consumability. Service intermediation is declaratively defined using a notion of a flow of basic mediation actions. An action can correspond to basic functions of intermediary processing, such as routing, decryption, or logging. Using these building blocks, the administrator builds these collective flows by augmenting actions as processing steps. To this end, DataPower provides a powerful Web Graphical User Interface (Web GUI), as shown in Figure 2-3. It shows the palette of common mediations (actions) that can be dropped in the message processing policy.



Create rule: Click New, drag action icons onto line.   Edit rule: Click on rule, double-click on action

Configured Rules

Order	Rule Name	Direction	Actions	
⇧ ⇩	Test_request	Client to Server		delete rule
⇧ ⇩	**Test_rule_0**	**Client to Server**		delete rule

Figure 2-3 DataPower Web GUI

In addition to the Web GUI, DataPower provides a command-line interface (CLI) that is accessible via SSH and Telnet. The appliances provide console (serial) and SSH access to this CLI. All management actions that can be done via the Web GUI can also be performed using the CLI.

2.3 Multi-protocol gateway service

The multi-protocol gateway (MPGW) service can perform everything an XML firewall can. It supports multiple front ports. MPGW is a key service that brings things together (such as any-to-any transformation and protocol translation) to make DataPower an ESB. MPGW can serve many purposes, including the following:

- Any-to-any transformation engine: MPGW can parse and transform arbitrary binary, flat text, and XML messages, including EDI, COBOL Copybook, ISO 8583, CSV, ASN.1, and ebXML. Unlike approaches based on custom programming, DataPower's patented DataGlue technology uses a fully declarative, metadata-based approach.
- Transport bridging: With support for a wide array of transport protocols, the MPGW is capable of bridging request/response flows to/from protocols such as HTTP, HTTPS, MQ, SSL, IMS Connect, FTP, and more, as listed above.
- Integrated message-level security: MPGW includes mature message-level security and access control functionality. Messages can be filtered, validated, encrypted, and signed, helping to provide more secure enablement of high-value applications. Supported technologies include WS-Security, WS-Trust, SAML, and LDAP.
- Lightweight message brokering: MPGW includes sophisticated multi-step message routing, filtering, and processing. It also includes multiple synchronous and asynchronous

transport protocols. Detailed logging and audit trail, including non-repudiation support, completes this capability.

The MPGW service is housed only in the XI50 and is typically located in the extranet or the intranet. It provides linkage to the System z® platform, as indicated in Figure 2-4. It also can be located in the intranet, where it can provide Web service management. As mentioned previously, the XI50 contains a super-set of all the functions of XS40 and can thus be used in place of the XS40 at other locations, such as for security in the DMZ.

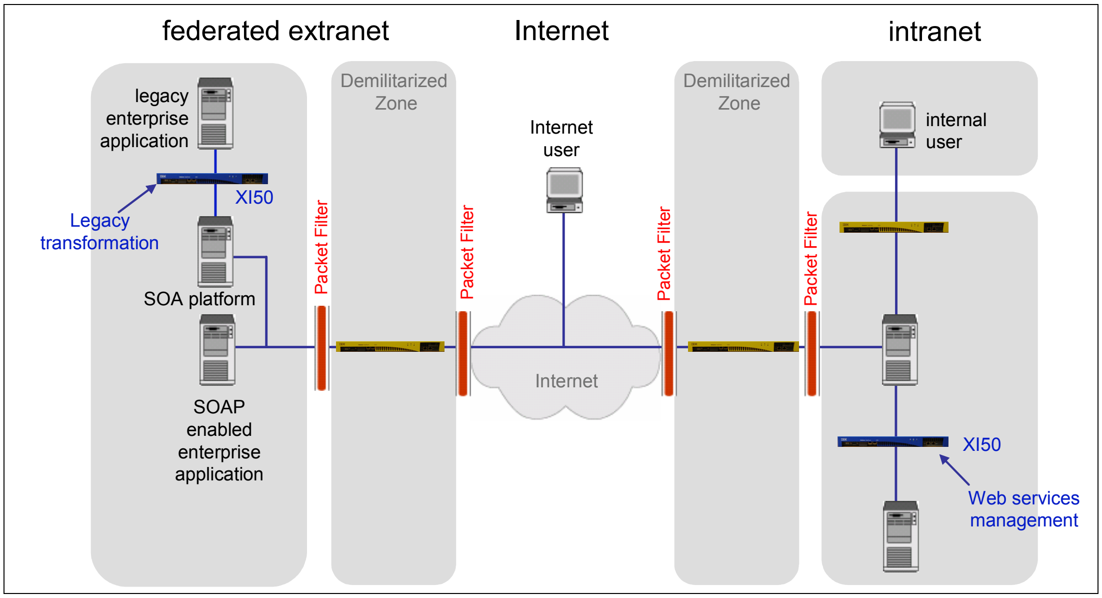


Figure 2-4 Deployment scenarios

MPGW can accept a variety of client-side protocol handlers. The gateway can subsequently provide a variety of server-side routing protocols. The client-side protocols do not need to be the same server-side protocols.

The gateway employs protocol handlers to offer protocol-specific connection points to clients who need to request service from back-end servers. The back-end service can be statically or dynamically identified. The MPGW service supports the following client-side and server-side protocols:

FTP The gateway can act as either an FTP client, polling a remote FTP server for requests, or as an FTP Server, accepting incoming FTP connections.

HTTP Including GET and POST. POST can contain XML, SOAP, DIME, and SOAP with Attachments.

HTTPS Including GET and POST. POST can contain XML, SOAP, DIME, and SOAP with Attachments.

IMS The gateway can accept incoming IMS protocol requests and can initiate IMS connections on the back side, if licensed only on the XI50.

NFS The gateway can poll an NFS-mounted directory for the presence of new files and place responses on an NFS-mounted directory.

MQ Messages The gateway can use GET and PUT queues to communicate using MQ messages, if licensed only on the XI50.

TIBCO EMS Supports TIBCO Enterprise Messaging Service, if licensed.

WebSphere JMS Supports the default WebSphere JMS server.

2.3.1 When to use

The MPGW service is used to send and receive XML traffic over multiple transports while performing protocol mediation, routing, document transformation, policy enforcement, and heritage integration. MPGW provides transport-independent transformations between binary, flat text files and XML message formats. Visual tools are used to describe data formats, create mappings between different formats, and define message choreography. MPGW can transform binary, flat-text, and other non-XML messages to help offer an innovative solution for security-rich XML enablement, ESBs, and mainframe connectivity. Use the MPGW service when you need to perform one or more of the following:

- Implement Web services policy enforcement points, including security policies and reliable messaging policies.
- Implement Web Services Addressing protocol enforcement.
- Accept and send SOAP, raw XML, or unprocessed (binary) documents.
- Filter, validate, transform, encrypt, or decrypt XML documents.
- Route XML documents to the appropriate back-end service.
- Sign documents or verify signatures.
- Process large documents in the streaming mode.
- Implement document-level security or service-level security.
- Communicate with clients, servers, and peers with SSL encryption.
- Monitor and control data traffic based on request sources and requested resources to the WSDL operation level.
- Allow, reject, strip, and process attachments (MIME, DIME, MTOM).

2.3.2 Use case

In this brief application modernization example, protocols are translated and documents are transformed from one format to another using the MPGW.

A document passes the security gateway over HTTPS and terminates at an XS40 in the DMZ. Authentication and authorization are performed against an access control list prior to the documents passing to the XI50 multi-protocol gateway.

The document is transformed from XML to the appropriate back-end destination format (for example, from XML to COBOL Copybook). Then the message is routed to the appropriate destination. In this example, the message is routed to an MQ Queue, where it is ready for delivery to a back-end CICS. See Figure 2-5.

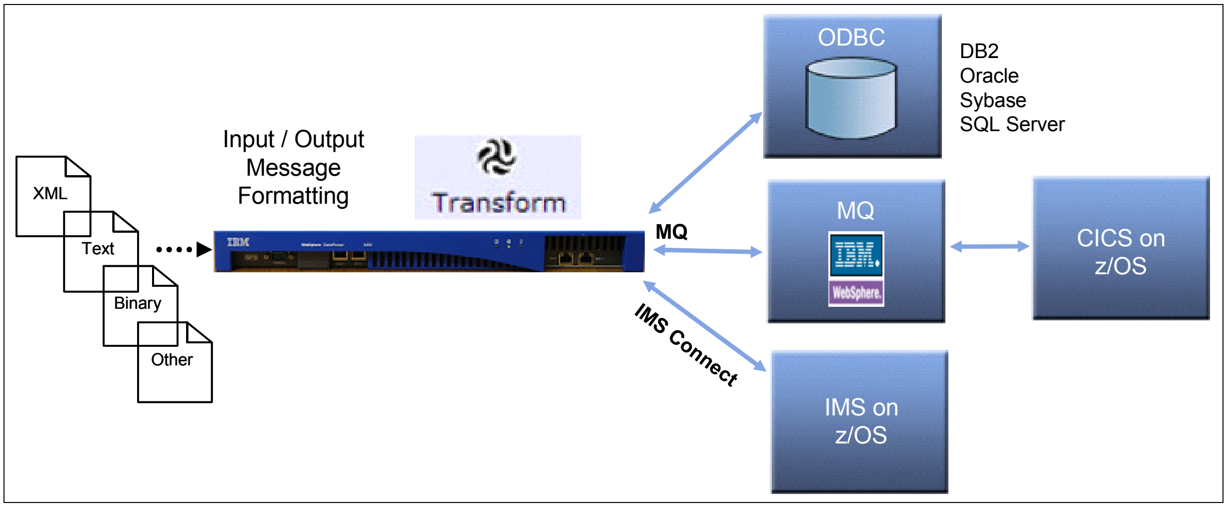


Figure 2-5 Application modernization scenario

2.4 Web Service Proxy service

The Web Service Proxy reads and parses one or more WSDL files to establish an automatic configuration. The WSDL files used for configuration could be obtained through subscriptions to a UDDI or WSRR registry. Once the service establishes a configuration matched to the underlying WSDL files, an administrator can then customize the service to meet additional requirements.

WSP automatically performs SOAP schema validation from WSDL types definition and decryption of encrypted request messages. It can import a WSDL to audit and monitor Web service operations. When you configure a Web Service Proxy service, you provide one or more of the following:

- Policy rule definition at WSDL, operation, port, service, and WS-Proxy levels
- Service level monitoring (SLM) at WSDL, operation, port, service, and WS-Proxy levels
- SLM data shared with HA/load-balanced peers
- Protocol support with Front Side Protocol Handlers, particularly for HTTP(S).
- UDDI and WSRR for WSDL publish and subscribe

2.4.1 When to use

The WS-Proxy service can establish a service that is described by a WSDL and is used to receive SOAP-based Web service traffic over multiple transports and forward the traffic to Web service applications over HTTP or MQ.

The Web Service Proxy service provides an XML threat-reduction and security-enforcement layer for XML messages and Web services transactions, including encryption, filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed logging.

Use a Web Service Proxy when you need to perform one or more of the following tasks:

- Implement Web services Policy enforcement points, including security policies and reliable messaging policies.
- Implement Web Services Addressing protocol enforcement.
- Accept and send SOAP, raw XML, or unprocessed (binary) documents.
- Filter, validate, transform, encrypt, or decrypt XML documents.
- Route XML documents to the appropriate back-end service.
- Sign documents or verify signatures.
- Process large documents in the streaming mode.
- Implement document-level security or service-level security.
- Communicate with clients, servers, and peers with SSL encryption.
- Monitor and control data traffic based on request sources and requested resources to the WSDL operation level.
- Allow, reject, strip, and process attachments (MIME, DIME, MTOM).
- WS-Proxy is the primary component to enforce governance aspects of SOA such as SLM.

2.4.2 Use case

Access to a third-party Web service, described by a WSDL, is front-ended by the Web Service Proxy. Access to a specific operation, GetLocationInformationByAddress, is controlled by an Authentication, Authorization, and Auditing Policy (AAA) that extracts identity information from the WS-Security UsernameToken. Requests exceeding specified service level parameters are queued and bleed off at the specified rate. Access to all other operations is unrestricted.

In this case the WS-Proxy is created by importing the defined WSDL. This includes operations and endpoints. Requests from the client are automatically validated via WSDL:types. Each operation from the WSDL is used to specify the processing policy. A SOAP request with a WS-Security header is validated and a response is returned.

2.5 XML firewall service

The XML firewall (XMLFW) service implements basic or comprehensive site-specific XML security practices. XML firewalls are used to provide security, threat mediation, and content processing services for XML, SOAP, and B2B applications. An XML firewall can perform everything an XSL proxy can, in addition to:

- Accept, send, transform SOAP, XML, or binary documents.
- Extensive Authentication, Authorization, and Auditing.
- Validation with schema, fetch, post, filter, validate, encrypt/decrypt, sign/verify, and dynamic routing.
- Complete HTTP Header RWD Control.
- XML threat protection, XDoS, MMXDoS, X-Virus, and SQL-Injection.
- MIME, DIME Attachment support, transformation, and conversion.
- Count, duration, and service-level monitors.

- Error support and log activity subscription. Destination support includes SMTP, SNMP, Syslog, Syslog-NG, NFS, File, and SOAP/HTTP.

2.5.1 When to use

The XML firewall service is used to send and receive XML traffic over HTTP to and from XML-based applications. Although the XML firewall can handle non-XML traffic, route using non-HTTP protocols within the firewall policy (for example, it can only accept and send back HTTP in the policy that processes input and response), and route using other protocols such as MQ, we recommend that the XML firewall service by used primarily for simple testing. The XMLFW is HTTP only. For other capabilities, consider the multi-protocol gateway service[1].

Use the XMLFW service when you need to take one or more of the following actions:

- Accept and send SOAP, raw XML, or unprocessed (binary) documents.
- Decrypt, encrypt, filter, transform, and validate XML documents.
- Route XML documents to the appropriate back-end service.
- Sign documents and verify signatures.
- Process large documents in the streaming mode.
- Implement document-level security or service-level security.
- Communicate with clients, servers, and peers with SSL encryption.
- Allow, reject, strip, and process attachments (MIME, DIME, MTOM).

2.5.2 Use case

In the case of a credit application, an XML firewall is created to provide integrity, confidentiality, and non repudiation via digital signing and encryption of messages. The client generates a SOAP message and the XML firewall processes the message, validates the SOAP documents using the schema ID attribute, and signs and encrypts the message using a certificate-key pair.

2.6 Web application firewall service

The Web application firewall service provides security, proxy, threat mediation, and content-processing services for a Web-based application (such as enrollment, benefits management, ticket sales, or a trading system). The Web application firewall is designed to handle traffic that is primarily URL-encoded HTTP form POST operations (or HTTP GET with or without query strings), rather than Web services SOAP-based XML payloads (although XML traffic can be handled as well). The Web application firewall offers:

- Destination service proxy
- SSL termination
- Authentication and authorization service
- Rate limiting
- Session start and timeout enforcement
- URL-encoded name-value input processing
- HTTP protocol filtering
- Threat protection, such as SQL injection
- Cookie handling, including sign and encrypt
- Error handling
- XML and non-XML content processing

[1] Based on a DataPower Forum entry by John Rasmussen on when to use XML fw, XSL proxy, MPGW, WSProxy, and Web App fw. Posted February 15, 2008, 3:25p.m.

2.6.1 When to use

Web application firewall (WAF) is used to provide security, monitoring, threat mediation, and content-processing services for Web-based, non-XML applications. Use this to send and receive HTTP traffic to and from Web applications. There is limited support for XML traffic.

You can use WAF to enforce security within your enterprise by deploying them on your company intranet or the demilitarized zone (DMZ). The WAF service can execute a security policy on messages that arrive in the DMZ before sending them to a back-end Web application. These tasks may also be performed on J2EE application servers, such as the WebSphere Application Server, but configuration using the DataPower management interface is much easier and does not require custom code. In addition, WAF can provide a single enforcement point for many applications within your enterprise.

The WAF service can be configured to virtualize a back-end Web application, handle rate limit requests, and enforce an AAA policy.

The WAF service can listen for requests on multiple TCP ports to virtualize or proxy back-end Web applications. The WAF service can also require Web clients to send requests over the Secure Sockets Layer (SSL). In situations when you need to limit the number of requests being sent to the back-end Web service, you can create a rate limiting policy to control the number of requests. You can also require clients to provide credentials when trying to access the Web application, which can be enforced using a DataPower AAA policy.

Use the WAF service when you need to provide one or more of the following:

- Destination service proxy
- SSL termination
- Authentication and authorization service
- Request rate limitation
- Session start and timeout enforcement
- URL-encoded name-value input processing
- HTTP Protocol filtering
- Threat protection, such as SQL injection
- Cookie handling, including sign and encrypt
- Error handling
- XML and non-XML content processing

2.6.2 Simple use case

In a simple use case you might use a Web browser as an external client to connect to the Web application firewall service in DataPower. Once you have been authenticated, your request is forwarded to the back-end Web application. The Web application firewall service uses an AAA policy to validate users. In a production environment, you would also need to secure the connection from the Web application firewall service to the back-end Web application, using either a security token or SSL. See Figure 2-6.

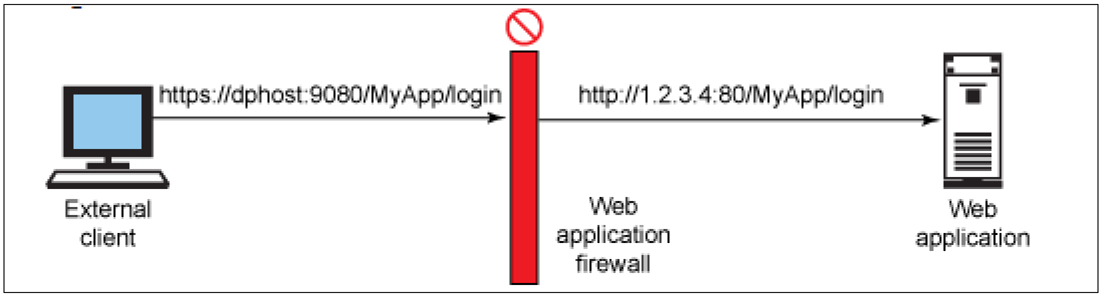

https://dphost:9080/MyApp/login
http://1.2.3.4:80/MyApp/login
External client
Web application firewall
Web application

Figure 2-6 Web application firewall

2.6.3 Host virtualization use case

DataPower appliances are typically deployed in the DMZ, which lets them perform pre-processing on incoming messages before they enter a company intranet. Web clients should not know the back-end endpoint of your Web application. First, if the back-end endpoint changes, your Web clients would need to be notified of those changes. Second, malicious users could send multiple requests to try to overwhelm the back-end Web application. To hide the Web application end-point address from Web clients, you can direct clients to a front port on which the WAF service listens for requests. This step is required when you are creating the WAF service.[1]

2.7 XSL Accelerator (Proxy) service

The very nature of XML and Web services presents a number of architectural and runtime difficulties. While XML provides many advantages over other wire-encoding schemes (its ubiquity and simplicity, for example), it is fundamentally more cumbersome to deal with. Parsing XML is tedious and rapidly becomes a performance bottleneck.

The XSL Accelerator service, more commonly referred to as XSL Proxy service, can perform XML parsing, XML schema validation, XML Path Language (XPath) routing, Extensible Stylesheet Language Transformations (XSLT), XML compression, and other essential XML processing with wirespeed XML performance.

2.7.1 When to use

XSL Proxy service is used for optimization and acceleration of XML and XSLT transformations.

[1] “Integrating Web applications with the DataPower Web application firewall service,” IBM Developer Works, Ozair Sheikh, 12 Dec 2007:
http://www.ibm.com/developerworks/websphere/library/techarticles/0712_sheikh/0712_sheikh.html

XSL Proxy is used to optimize transformation of XML/SOAP with XSLT by providing an order of magnitude performance increase by off loading it from servers and networks. The XSL proxy can provide SSL support via SSL hardware for communication with clients, servers, and peers and complete HTTP header RWD control. Message validation can be accomplished via the schema validation feature. The proxy can process large documents using streaming mode, provides count and duration monitors, and provides error-tracking support and log activity subscription features.

Use the XSL Proxy service when you need to do one or more of the following:

- Accept and send SOAP, raw XML, or unprocessed input.
- Validate and transform XML documents.
- Process large documents in the streaming mode.
- Communicate with clients, servers, and peers with SSL encryption.
- Monitor and log activity by delivering log information to external managers.
- Allow, strip, or process attachments (DIME or MIME).

2.7.2 Use case

A typical use case involves building a proxy for an HTTP server and defining a policy to transform based on XSLT processing instruction. The client issues a request and the response is transformed and returned to the client as HTML. See Figure 2-7.

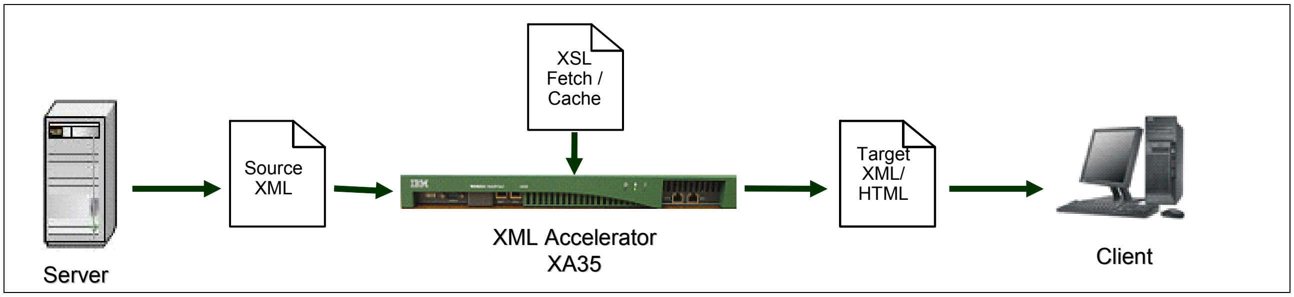


Figure 2-7 XSL proxy

2.8 An architectural approach to service selection

The role of DataPower in the SOA topology depends on what DataPower is doing in the network. If it provides a Web service facade to the back end, it will probably rely on WS-Proxy. If it plays a role in bridging and routing with traditional applications or a layered ESB environment on the back end, it is probably using mostly the MPGW service. If it is in the DMZ acting as a security gateway, it could use either MPGW or XML firewall.

A quick way to select the best service can be found by answering some questions:

- Does the service use a WSDL? If so, then use WS-Proxy.
- Does the service uses multiple transports? If so, then use MPGW.
- Does the service use only XML over HTTP? If so, then use XML firewall.
- Is there only non-XML and HTTP traffic? If so, then use Web application firewall.

While examining each service, it is best to choose the service with the most sophisticated functionality. For example, if you choose XML firewall today, you will lose out on the function provided by the Web Service Proxy of dynamically updating WSDL files. The MPGW is a super-set of the XML firewall and should be considered its replacement for reasons such as efficiency, ongoing development, and compatibility with other services.

Use Web application firewall to provide security, threat mediation, and content processing services for Web-based application

Use the MPGW, instead of XML firewall, if you intend to use protocols other than HTTP. In most cases, services with HTTP front end and back end should use MPGW instead of XML firewall, because of its extensibility. Multiple Front Side Handlers are supported. Keep in mind that service level management is only available in MPGW and WSP.

A multi-protocol gateway offers many of the same services and capabilities as a Web Service Proxy. Unlike a Web Service Proxy, a MPGW cannot use a WSDL to determine a configuration.

Use MPGW when you want to do one or more of the following:

- Implement reliable messaging policies.
- Implement Web Services Addressing protocol enforcement.
- Accept and send SOAP, raw XML, or unprocessed (binary) documents.
- Transform XML to binary format documents and binary format documents to XML.
- Filter, validate, transform, encrypt, or decrypt XML documents.
- Route XML documents to the appropriate back-end service.

Service composition

SOA extends the object-oriented concepts of encapsulation and re-use into the distributed IT world of integrated applications and even integrated enterprises. For example, we can think of a well-crafted service framework implemented in an ESB such as DataPower as providing reusable components to serve as building blocks for complex applications. Concepts such as chaining services and reusing services in different applications can also be used to define how services can be determined. As an example, one might want to split a complex application into two firewalls—a WAF frontending and serving HTML followed by an XSL Proxy doing only XML transformation. In this way, the later service can be reused as a means for intranet ESB-ESB communication.

The remaining chapters of this book address SOA topics such as Web services, security, and integration.

3

Web services and policy management

The encapsulation of services by interfaces and their invocation through location-transparent, interoperable protocols is the basic means by which SOA enables increased flexibility and reusability. In order to really understand how these benefits can be achieved, we delve a little further into the detail of good service design by considering coupling and decoupling of aspects of service interactions.

SOA Governance is a complex topic, much of which is beyond the scope of this book. We introduce it in this chapter, but our discussion is limited to those aspects of it that can be implemented using Web services in DataPower.

3.1 Web services

Web services is a loosely coupled, language-neutral, platform-independent way of linking applications within organizations, across enterprises, and across the Internet. A key benefit of the emerging Web services architecture is the ability to deliver integrated, interoperable solutions, which makes it critical to ensure the integrity, confidentiality, and overall security of these services.

The implementation of SOA using Web services technologies is the current state of the art in systems integration. But for some time, the vision of much of the IT industry has been to achieve rapid, flexible integration of IT systems across all elements of the business cycle. The drivers behind this vision include:

- Increasing the speed at which businesses can implement new products and processes or change existing ones
- Reducing implementation and ownership costs
- Enabling flexible pricing models by outsourcing elements of the business or moving from fixed to variable pricing, based on transaction volumes
- Simplifying the integration work that is required by mergers and acquisitions
- Achieving better IT utilization and return on investment
- Simplifying the enterprise architecture and computing model

Achievement of these goals affects the entire scope of a business's processes and IT systems. Perhaps until recently the industry has lacked a consistent and comprehensive approach to technology and architecture on this scale. Although several systems that cover some elements of this scope have been implemented, there has not been a single, broadly accepted approach.

The combination of service-oriented architecture, an approach that draws together proven techniques from several proceeding architecture and design styles, with new open standards and integration technologies associated with Web services, has the potential to provide such a consistent approach.

We can apply these relationships to various aspects of service that can be identified in a SOA. For some aspects, SOA or other design principles specify the desired style of relationship. For other aspects, several relationships might be appropriate, depending on specific scenarios. For each aspect, different techniques can be applied to implement the desired relationship. It should be noted, however, that this area is the subject of ongoing debate and evolution, and the table should not be taken as definitive. Similarly, the available technologies are evolving rapidly. For example, the emerging WS-Policy specification will affect this area of design deeply in coming years.

The basic Web services are often described in terms of SOAP, WSDL, and services registries such as WebSphere Services Registry and Repository (WSRR) and UDDI-based registries. The link between Web services and SOA is threefold:

- Web services provide an open standard and machine-readable model (WSDL) for creating explicit, implementation-independent descriptions of service interfaces. WSDL is an XML-based interface definition language that separates function from implementation, and enables design by contract as recommended by SOA. WSDL descriptions contain a PortType (the functional and data description of the operations that are available in a Web service), a binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP/HTTP), and a port (providing a specific address through which a Web service can be invoked using a specific protocol binding).

- Web services provide communication mechanisms that are location-transparent and interoperable.
- Web services are evolving through BPEL4WS, document-style SOAP, WSDL, and emerging technologies such as WS-ResourceFramework to support the technical implementation of well-designed services that encapsulate and model reusable function in a flexible manner.

For further information see:

- IBM Web services

  http://www.ibm.com/webservices
- IBM on demand operating environment

  http://www-3.ibm.com/software/info/openenvironment/
- IBM developerWorks®: SOA and Web services zone

  http://www.ibm.com/developerworks/webservices

3.1.1 DataPower Web Service Proxy service

DataPower is a powerful addition to any integration architecture when the Web Service Proxy is a component of the solution. What makes DataPower a significant Web services engine is that it already possesses fundamental capabilities in the following prerequisites for good Web services implementations:

- Ability to handle open standards messaging protocols (SOAP)
- Ability to handle the industry-preferred transport protocols (including HTTP, HTTPS, JMS)
- Executes (asserts) security management at both the transport level (HTTPS) and at a protocol level (WS-Security)
- Provides dynamic service discovery and binding through service registries such as WSRR and UDDI, which means that service integration can occur on demand

With DataPower, Web services can utilize existing infrastructure as appropriate. Deployment costs and efforts are marginalized. The increasing availability of Web services support in Enterprise Application Integration middleware enables the integration of different middleware infrastructures via a proxy gateway pattern, which is DataPower’s strength. DataPower is built with emerging enterprise service bus technologies, which interact with existing integration infrastructure, avoiding the forced replacement of traditional infrastructure to accommodate the emerging technologies. Although SOA and Web services cannot remove the cost and effort of deploying integration infrastructure, with DataPower, they offer several characteristics to minimize it.

Service-oriented architecture as implemented with DataPower’s Web services Gateway defines concepts and general techniques for designing, encapsulating, and invoking reusable business functions through loosely bound service interactions. Most of the techniques have been proven individually in previous technologies or design styles. SOA unites them in an approach intended to bring encapsulation and re-use to the enterprise level.

Web services in DataPower provide an emerging set of open-standard technologies that can be combined with proven existing technologies to implement the concepts and techniques of SOA. DataPower provides industry support for Web services standards, interoperability among different implementations of Web services, and the infrastructure technology that is required to support an SOA. This gives customers increasingly mature and sophisticated technologies that are suitable for SOA implementation.

DataPower provides a way to address some of the issues associated with today's building of Web services in conjunction with SOA:

- DataPower provides support for Web services and SOA using open non-proprietary standards.
- DataPower supports multiple bindings through a WSDL, and is architected to proxy a HTTP transport and function as a gateway to a more secured transport.
- DataPower is designed from the ground up to drive a solution based on open standards. However, when the design calls for it, DataPower has onboard integration client run times for EMS, FTP, IMS, and SQL.

Aspects of coupling and decoupling of service interactions

A basic tenet of SOA is that the use of explicit service interfaces and interoperable, location-transparent communication protocols means that services are loosely coupled with each other. DataPower's Web Service Proxy can decouple several areas of the solution through the WSDL that is imported, and the population of the policy model object using the GUI. The following decoupling examples transverse from the high-level business to the lower level technology constructs. DataPower Web Service Proxy service can:

- Decouple business data models that are usually coupled between service requesters and providers. The application code of each must understand the information that is required to describe (for example) a customer, account, or order.
- Decouple communication protocols by how they are declared in the service interface. In practice, this requires that application's code to a protocol-independent service API, such as a suitable implementation of JAX-RPC. If this is the case, then the protocol binding in the service interface definition can be changed (for example, from SOAP/HTTP to SOAP/JMS).
- Decouple data formats: It is very common, for example, to convert traditional formats, such as COBOL copybooks, to XML formats when enabling service interfaces to heritage systems. Alternatively, different XML schema may be used by different systems in an SOA to describe the same data models. In either case, the supported format is, or can be, defined in a service interface, and middleware transformation capabilities can be used in the service infrastructure to perform the required transformations without affecting application code or behavior.
- Decouple the identity of a service provider, which would be negotiated through a third-party broker component such as Tivoli Identity Manager (TIM). TIM might use geographical location, client identify, membership scheme information, transaction value, or several other criteria to match the service requester with a suitable service provider.

Configuring Web Service Proxy

A single Web Service Proxy object can handle traffic that is bound for all endpoints that are described in multiple WSDL files. A remote client can use the address and port that are assigned to a single Web Service Proxy to obtain services from all endpoints that are described in these WSDL files. Therefore, the Web Service Proxy serves as an interface to the WSDL domain, a virtualization where abstraction is achieved using the WS-Policy interface.

Creating a Web Service Proxy

To create a Web Service Proxy, you only need to define the basic configuration. The basic configuration consists of specifying the source of the configuration. The configuration source can be any combination of the following types. Key configuration considerations for WS-Proxy creation are:

- Determining the WSDL source
- Configuring Service Level Monitor (SLM)
- Configuring Web Services Policy (WS-Policy)
- Configuring the mediation for Web Services Addressing (WS-Addressing)
- Configuring for reliable messaging
- Implementing proxy-specific threat protection measures
- Modifying default basic proxy settings
- Configuring header and parameter settings

3.1.2 Web services proxy summary

The Web Service Proxy (WSP) service reads and parses one or more WSDL files to establish an automatic configuration. The WSDL files used for configuration could be obtained through subscriptions to a UDDI or WSRR registry. Once the service establishes a configuration matched to the underlying WSDL files, an administrator can then customize the service to meet additional requirements.

WSP automatically performs SOAP schema validation from WSDL types definition and decryption of encrypted request messages. It can import a WSDL to audit and monitor Web service operations. When you configure a Web Service Proxy service, you provide one or more of the following:

- Policy rule definition at WSDL, operation, port, service, and WS-Proxy levels
- Service level monitoring (SLM) at WSDL, operation, port, service, and WS-Proxy levels
- SLM data shared with HA/load-balanced peers
- Extensive protocol support with Front Side Protocol Handlers: HTTP(S), FTP, MQ, raw XML, WebSphere JMS, and NFS and FTP front-side pollers
- UDDI and WSRR for WSDL publish and subscribe

The WS-Proxy service can establish a service that is described by a WSDL and is used to receive SOAP-based Web service traffic over multiple transports and forward the traffic to Web service applications over HTTP or MQ.

The Web Service Proxy service provides an XML threat-reduction and security-enforcement layer for XML messages and Web services transactions, including encryption, filtering, digital signatures, schema validation, WS-Security, XML access control, XPath, and detailed logging.

Use a Web Service Proxy when you need to perform one or more of the following tasks:

- Implement Web services policy enforcement points, including security policies and reliable messaging policies.
- Implement Web Services Addressing protocol enforcement.
- Accept and send SOAP, raw XML, or unprocessed (binary) documents.
- Filter, validate, transform, encrypt, or decrypt XML documents.
- Route XML documents to the appropriate back-end service.
- Sign documents or verify signatures.

- Process large documents in the streaming mode.
- Implement document-level security or service-level security.
- Communicate with clients, servers, and peers with SSL encryption.
- Monitor and control data traffic based on request sources and requested resources to the WSDL operation level.
- Allow, reject, strip, and process attachments (MIME, DIME, MTOM).

This proxy service is also discussed in 2.4, “Web Service Proxy service” on page 18.

3.2 Policy management

For some, a Web services framework seems like an expensive, resource-intensive interface that could possibly be accomplished by other means, albeit without open standard support. But this position can be shortsighted because invariably it fails to take into account the policy management that can be achieved around the Web service as defined by the WS-Policy framework. The value statements around Web services policy management cannot be overstated, most importantly in the WS-Policy domain of WS-SecurityPolicy. As a DMZ ready security appliance optimized for authorization, authentication, and signing messages on the edge of a solution space (that is, ESB), DataPower is an amazing fit for applying Web services policies. DataPower’s policy model is very robust and the Web Service Proxy GUI is very well designed to support building policy management statements without going into the code. For SOA-based solutions, this is DataPower’s strength.

3.2.1 SOA governance

SOA increases the level of cooperation and coordination required between business and information technology (IT), as well as among IT departments and teams. This cooperation and coordination is provided by SOA governance, which covers the tasks and processes for specifying and managing how services and SOA applications are supported.

Governance becomes more important in SOA than IT in general. In SOA, service consumers and service providers run in different processes, are developed and managed by different departments, and require a lot of coordination to work together successfully. For SOA to succeed, multiple applications need to share common services, which means that they need to coordinate making those services common and reusable. These are governance issues, and they are much more complex than in the days of monolithic applications or even in the days of reusable code and components.

As companies use SOA to better align IT with the business, companies can ideally use SOA governance to improve overall IT governance. Employing SOA governance is key if companies are to realize the benefits of SOA. For SOA to be successful, SOA business and technical governance is not optional, it is required. SOA governance is beyond the scope of this book, except for those aspects of it that can be implemented using Web services in DataPower.

3.2.2 Web Service Policy Framework (WS-Policy Framework)

WSDLs are standards-based interfaces to services, but how do we interact with those interfaces? What are the controls that must be invoked to insure interoperability between disparate systems, platforms, solutions?

WS-Policy

To address the interoperability needs for Web services, DataPower fully leverages the Web Service Policy Framework. This is a set of specifications that were created to describe the capabilities and constraints of domain-specific policies that establish the framework for interoperability. Through WS-Policy's defined metadata, DataPower enables interoperability between Web service consumers and Web service providers. Through the proven WS-Policy specifications, DataPower is able to automate the service governance models to create a concrete instance of Web service governance. DataPower uses WS-Policy to provide a general purpose model and syntax that describes and communicates the policies of a Web service.

DataPower applies a policy as a collection of one or more policy assertions. Assertions express the capabilities and constraints of a particular DataPower-managed Web service. Some assertions specify traditional requirements and capabilities that will ultimately manifest on the wire (for example, authentication scheme, transport protocol selection). Some assertions specify requirements and capabilities that have no wire manifestation, yet are critical to proper service selection and usage (for example, privacy policy, QoS characteristics). DataPower leverages a single policy grammar to allow both kinds of assertions to be reasoned about in a consistent manner in the Web service.

Tip: To establish governance around Web services, DataPower relies on the Web Service Policy Framework specification to define a flexible and concise syntax and semantic for service providers and service requestors that describes their requirements, preferences, and capabilities in flexible and concise domain-specific forms of assertions.

Policy attachments

Since WS-Policy does not specify how policies are discovered or attached to a Web service, DataPower uses the WS-PolicyAttachment language to define several methods for associating the WS-Policy expressions with Web services. WS-PolicyAttachment can describe how policies are to be attached to the elements of a WSDL. DataPower is WS-PolicyAttachment compliant.

Policy assertions

A policy assertion identifies a behavior that is a requirement (or capability) of a policy subject. DataPower uses assertions to indicate domain-specific (for example, security, transactions) semantics and are expected to be defined in separate, domain-specific specifications. Policy expressions allow for both simple declarative assertions as well as more sophisticated conditional assertions.

A policy can be built up using assertions and nested combinations of operators. This policy syntax is used to describe acceptable combinations of assertions to form a complete set of instructions to the policy-processing instruction for a given Web service invocation. Each set of assertions is termed an alternative.

The flexible syntax of the Web Service Policy Framework, however, can lead to potentially complex policies. So a normal form of policy language was defined to simplify the manipulation and to clarify the understanding of policies. Policies in the normal form are how the basic operation of the policy processing is to be carried out. Through normalization, complex policies can be reduced prior to executing more complex statements such as merge and intersection operations.

Tip: An assertion is the basic unit of policy. It can be thought of as an instruction to a policy-processing infrastructure. The meaning of individual assertions is specific to each domain. For example, the assertion could declare that the message be encrypted. The actual definition of this assertion would be in the WS-Security Policy domain specification.

Bringing it together

DataPower has a rich GUI to support building a Web services based policy. To build a policy, there is a hierarchy to consider:

1. Policy: A set of domain-specific policy statements
2. Policy Statement: A group of policy assertions
3. Policy Assertion: An individual preference, requirement, capability, or other property

Framework extensibility: The WS-Policy framework was built to be extensible, and DataPower can easily leverage policy language additions as the business requirements dictate. For more advanced users, there may be a need to create new policy expressions from existing (or new) individual policy assertions from different policy domains (that is, WSSecurity, WSReliableMessaging). Policy authors should refer to the guidelines found at:

`http://www.w3.org/TR/2007/WD-ws-policy-guidelines-20070928/`

3.2.3 Configuring Web Service Policy

What is WS-Policy and how is it used with the DataPower Web services proxy service? DataPower implements WS-Policy Framework and WS-Policy Attachment specifications using the Web Service Proxy service.

WS-Policy configuration

For DataPower, the Web Service Proxy already supported the configuration of endpoints from a WSDL file, from a set of WSDL files, from a WSRR or UDDI subscription. Support for governance-enhanced WS-Policy augments endpoint configuration and supports the following behaviors:

- Parsing WSDL files that contain WS-Policy elements, which demonstrates WS-Policy internal attachment behavior and recognizes standardized policy domains.
- Configuring the Web Service Proxy to recognize and enforce standardized policy domains. Policy domains are field-extensive. The Web Service Proxy supports the following standardized policy domains:
  - WS-SecurityPolicy
  - WS-ReliableMessagingPolicy
- Associating, or attaching, WS-Policy expressions, which are not embedded in the WSDL file, to a WS-Policy policy subject (for example, a WSDL endpoint). You can use the service view of the Web Service Proxy from the WebGUI to attach WS-Policy expressions to a set of WS-Policy policy subjects.

WS-Policy models

DataPower uses an abstraction of the policy model from the physical WSDL for setting management policies of the service. So, a physical WSDL can change, but the abstract model remains the same (Figure 3-1).

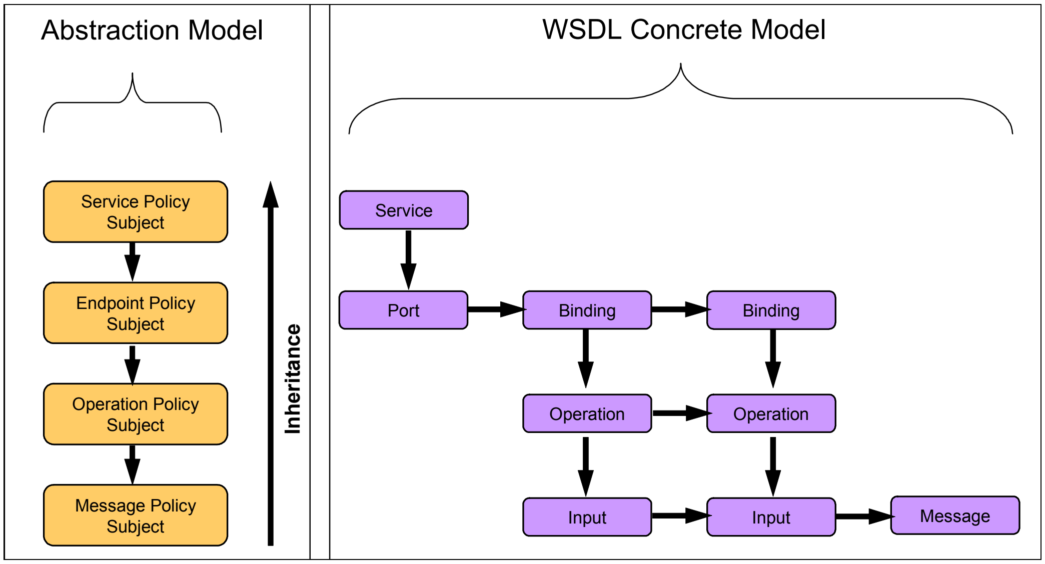


Figure 3-1 Policy model

The DataPower policy abstract model leverages the hierarchical structure of the WSDL through inheritance. When attaching policy at different levels of the WSDL hierarchy, the policy applies to this level of the hierarchy and is inherited by lower levels of the hierarchy. At a lower level of the hierarchy, one can refine the policy at this level by attaching policy. The refined policy applies to this level of the hierarchy and is inherited by lower levels. Each level of the policy subject can inherit the part's policy attributes, but the inherited policy can be refined and that refinement of the policy will cascade to its children. This system behavior allows for concise control of policy statements as the granularity of the service increases, which maintains a coherent linkage of policies throughout the service object. This rich feature set of DataPower is a powerful way of applying policy statements throughout the service model and offers fine-grained control to the implementor.

Policy and assertion security

We strongly recommend that DataPower should sign policies and assertions to prevent tampering. DataPower should never accept policies unless they are signed and have an associated security token to specify that the signer has the right to *speak for* the scope containing the policy. That is, a relying party should not rely on a policy unless the policy is signed and presented with sufficient credentials to pass the relying parties' acceptance criteria.

Tip: DataPower should secure policies and assertions as part of a SOAP message using WS-Security or can secure using other object-specific security mechanisms.

Web Services Security Policy Language (WS-SecurityPolicy)

WS-Policy defines a set of security policy assertions that apply to Web Services Security:

- SOAP Message Security
- WS-Trust
- WS-SecureConversation

The policy defines a base set of assertions that describe how messages are to be secured. Flexibility with respect to token types, cryptographic algorithms, and mechanisms used, including using transport-level security, is part of the design and allows for evolution over time. The intent of the policy language is to provide enough information for compatibility and interoperability to be determined by Web services participants, along with all information necessary to actually enable a participant to engage in a secure exchange of messages.

DataPower uses WS-SecurityPolicy as a building block that is used in conjunction with other Web service and application-specific protocols to accommodate a wide variety of security models.

Security policy model

The security policy language defines policy assertions for the security properties for Web services. These assertions are primarily designed to represent the security characteristics defined in the WSS (SOAP Message Security, WS-Trust, and WS-SecureConversation specifications), but they can also be used for describing security requirements at a more general or transport-independent level, though DataPower remains focused on the WS-* implementations.

The language defines sets of patterns or sets of assertions that represent common ways to describe how messages are secured on a communication path. The intent allows flexibility in terms of the tokens, cryptography, and mechanisms used, including leveraging transport security, but to be specific enough to ensure interoperability based on assertion matching.

It is a goal of the security policy model to leverage the WS-Policy framework’s intersection algorithm for selecting policy alternatives and the attachment mechanism for associating policy assertions with Web service artifacts. Consequently, wherever possible, the security policy assertions do not use parameters or attributes. This enables first-level, QName-based assertion matching without security domain-specific knowledge to be done at the framework level. The first level matching is intended to provide a narrowed set of policy alternatives that are shared by the two parties attempting to establish a secure communication path.

In adherence to the model, DataPower assertions allow additional attributes, based on schemas, to be added on to the assertion element as an extensibility mechanism, but the WS-Policy framework will not match based on these attributes. Attributes specified on the assertion element that are not defined in this specification or in WS-Policy are to be treated as informational properties.

Security Assertion model

DataPower supports rich semantics for combinations of security constraints divided into simple patterns:

- What parts of a message are being secured (protection assertions)
- General aspects or pre-conditions of the security (conditional assertions)
- The security mechanism (security binding assertions) that is used to provide the security
- The token types and usage patterns (supporting token assertions) used to provide additional claims
- Token referencing and trust options (WSS and trust assertions).

To indicate the scope of protection, assertions identify message parts that are to be protected in a specific way, such as integrity or confidentiality protection, and are referred to as protection assertions.

The general aspects of security include the relationships between or characteristics of the environment in which security is being applied, such as the tokens being used, which are for integrity or confidentiality protection and that are supporting the applicable algorithms to use, and so on.

The security binding assertion is a logical grouping that defines how the general aspects are used to protect the indicated parts (for example, that an asymmetric token is used with a digital signature to provide integrity protection, and that parts are encrypted with a symmetric key that is then encrypted using the public key of the recipient). At its simplest form, the security binding restricts what can be placed in the wsse:Security header and the associated processing rules.

WS-ReliableMessagingPolicy

It is often a requirement for two Web services that wish to communicate to do so reliably in the presence of software component, system, or network failures. The primary goal of WS-ReliableMessagingPolicy is to create a modular mechanism for reliable message delivery. It defines a messaging protocol to identify, track, and manage the reliable delivery of messages between exactly two parties, a source and a destination. It also defines a SOAP binding that is required for interoperability. Additional bindings may be defined.

This mechanism is extensible, allowing additional functionality, such as security, to be tightly integrated. This specification integrates with and complements the WS-Security, WS-Policy, and other Web services specifications. Combined, these allow for a broad range of reliable, secure messaging options.

WS-ReliableMessagingPolicy is the second domain-specific assertion (WS-SecurityPolicy being the other) identified under the Web Services Policy Framework that DataPower supports.

Many errors may interrupt a conversation. Messages may be lost, duplicated, or reordered. Furthermore, the host systems may experience failures and lose volatile state. WS-ReliableMessaging provides an interoperable protocol that a reliable messaging source and reliable messaging destination use to provide the source and destination with a guarantee that a message that is sent will be delivered. With DataPower's implementation of WS-ReliableMessaging (the source endpoint, destination endpoint, and DataPower acting as a mediator) delivery assurance is guaranteed (assuming no loss of infrastructure).

DataPower leverages the standard four basic delivery assurances documented in the specifications that endpoints can provide:

- AtMostOnce: Messages will be delivered at most once without duplication or an error will be raised on at least one endpoint. It is possible that some messages in a sequence may not be delivered.
- AtLeastOnce: Every message sent will be delivered or an error will be raised on at least one endpoint. Some messages may be delivered more than once.
- ExactlyOnce: Every message sent will be delivered without duplication or an error will be raised on at least one endpoint. This delivery assurance is the logical *and* of the two prior delivery assurances.
- InOrder (Sequence): Messages will be delivered in the order in which they were sent. This delivery assurance may be combined with any of the above delivery assurances. It requires that the sequence observed by the ultimate receiver be non-decreasing. It says nothing about duplications or omissions.

Securing WS-ReliableMessaging

DataPower can be configured to select a specific *AAA policy* to perform authentication of incoming reliable messaging messages. As this AAA policy can be the same one that is used in later processing by the request or response rule, the AAA policy can be cached, so it does not have to be evaluated again. The same AAA policy can secure the reliable messaging control messages, such as CreateSequence and TerminateSequence, and it can secure incoming reliable messaging data messages, which have a sequence header. This prevents unauthorized clients from issuing CreateSequence requests, or from disrupting existing reliable messaging sequences with CloseSequence or TerminateSequence messages, or from falsely acknowledging messages with SequenceAcknowledgement messages.

DataPower can establish a wire-level *SSL* session to protect sequence life-cycle messages. All reliable messaging control messages and sequence messages are bound to the original SSL/TLS session that is created by the reliable messaging source to transmit the CreateSequence control message. Sequence messages that are received by the reliable messaging destination with the correct identifier but on a different SSL/TLS session are rejected.

Other policy enhancements for reliable messaging

In this section we discuss other policy enhancements for reliable messaging.

Destination accept incoming CreateSequence

To establish better control over what clients can request reliable messaging, a choice can be made to force the source (client) to present an offer for such service, and if the destination responds favorably (accept), only then will DataPower create a reliable messaging destination. A lessor control allows any client to initiate a reliable messaging conversation.

InOrder (sequence) messages

Sequencing messages in a known order (usually processing order) provides more assurance as to the quality of the message. The destination can make more assumptions around the quality of the data if it can be assured of processing in the same sequence as the source. Therefore, flagging *Destination InOrder Delivery Assurance* insures that:

- There is ExactlyOnce delivery assurance.
- No messages will be passed from the receive queue for further processing unless their sequence number as assigned by the client is one greater than the last one that was processed by the destination. The source may issue an acknowledgement to the destination for messages processed until it is satisfied that all messages have been processed.

  As the specification assigns message assurance to the end points, it is up to the source endpoint to issue sequence requests for missing messages until the destination endpoint issues a SequenceAcknowledgements confirming receipt. DataPower facilitates the process by acting as a middleware helper by providing queue facilities for messages that are beyond a gap in the received sequence numbers. Defaulting at 10 and maxing at 256, we are reminded that DataPower is a very loosely coupled stateless appliance that carries limited memory for long-running processes, in this case sequenced messaging. So it is prudent to ensure that the endpoints function correctly when sequenced reliable messaging is enabled.

Destination accept two-way offers/source create sequence on request/source create sequence on response

DataPower can assume more of a mediator role with enablement of any of these conditions. DataPower can provide more granular control in the middleware layer for frequency and maximum count of acknowledgements retransmissions, maximum queue length, inactivity intervals before time-out, and so on. Again with an eye on DataPower's memory constraints, it can provide strong reliable messaging support for the endpoints and absorbs much of the burden for exception processing.

Web Security conformance policy (WS-I Conformance)

At the policy or WSDL level, it is possible to set the WS-I Conformance policy assertion within DataPower. The current main profile options are:

- WS-I BP 1.0: validates messages against the requirements that are defined in WS-I Basic Profile, Version 1.0
- WS-I BP 1.1 (Default): validates messages against the requirements that are defined in WS-I Basic Profile, Version 1.1
- WS-I AP 1.0 (Default): validates messages against the requirements that are defined in WS-I Attachments Profile, Version 1.0
- WS-I BSP 1.0 (Default): validates messages against the requirements that are defined in WS-I Basic Security Profile, Version 1.0. 5

Web Services Interoperability (WS-I) Basic Profile

The WS-I Basic Profile focuses on the core foundation technologies upon which Web services are based: HTTP, SOAP, WSDL, UDDI, XML, and XML Schema. The goal of Web services is to enable communication between different software and hardware systems. These systems typically differ in both their hardware and software configurations. These differences have been overcome through the definition of standard protocols, such as those employed in building Web services. Occasionally, incompatibility issues arise even when using these standard protocols, which can lead to interoperability problems. WS-I Basic Profile is a set of guidelines that Web services should adhere to in order to achieve optimum interoperability.

A unique feature of Web services is that it is a relatively high-level integration protocol with near-ubiquitous support in the IT industry. Adherence to the specifications of this protocol alone is an important reason for its success and is behind why many individual projects have used the Web services standards to perform integrations between different platforms.

For DataPower services that use processing policies, you can determine whether the configuration of the service sends requests that are conformant to the back-end server and send responses that are conformant to the original client. For example, validation detects the appropriate use of the Envelope method for a sign action. If the configuration is non-conformant, the summary includes details about the sign action with a value other than the WS-Security method.

Setting a conformance policy object

Setting a conformance policy object validates incoming requests against back-end server responses against the WS-I Basic Profile and WS-I Basic Security Profile standards.

The WS-I Basic Profile specification defines conformance of a Web service instance. The profile consists of the following set of non-proprietary Web services specifications:

- SOAP 1.1
- WSDL 1.1
- UDDI 2.0
- XML 1.0 (Second Edition)
- XML Schema Part 1: Structures
- XML Schema Part 2: Data types
- RFC2246: The Transport Layer Security Protocol Version 1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile

- RFC2616: HyperText Transfer Protocol 1.1
- RFC2818: HTTP over TLS
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer Protocol Version 3.0

The profile provides constraints and clarifications to those base specifications with the intent to promote interoperability. Some of the constraints imposed by the profile are intended to restrict, or require, optional behavior and functionality, so as to reduce the potential for interoperability problems. Some of the constraints or requirements are provided to clarify language in the base specification that may be the source of frequent misinterpretation that have been a frequent source of interoperability problems.

The highlights of the key constraints imposed by the profile are:

- Precludes the use of SOAP encoding
- Requires the use of HTTP binding for SOAP
- Requires the use of the HTTP 500 status response for SOAP fault messages
- Requires the use of the HTTP POST method
- Requires the use of WSDL1.1 to describe the interface of a Web service
- Requires the use of rpc/literal or document/literal forms of the WSDL SOAP binding
- Precludes the use of solicit-response and notification-style operations
- Requires the use of WSDL SOAP binding extension with HTTP as the required transport
- Requires the use of WSDL1.1 descriptions for UDDI Model elements representing a Web service

Usage:

- WS-Conformance is a fundamental Web services extension that enables many other key Web services extensions. We recommend that WS-I BP 1.0 or 1.1 be turned on.
- Interoperability can be hampered if the partners are validating against a different WS-I Profile that a particular service is using. An agreement on profiles should be reached between the partners, while always striving to validate against the more recent WS-I BP 1.1 profile.

Configuring Web Services Addressing (WS-Addressing)

WS-Addressing provides transport-neutral mechanisms to address Web services and messages. Specifically, this specification defines XML elements to identify Web service endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

Web Services Addressing is a mechanism for controlling routing based on standardized SOAP header content. With WS-Addressing enabled, routing for client requests and associated server responses can be uncoupled, as can the routing of fault and standard response messages. The current WS-Addressing implementation is interoperable with both the WebSphere Extended Technology Kit implementation and with the Web services enhancements for (.NET) (WSE 2.0). It provides:

- Support for traditional back-end routing with WS-Addressing To headers
- Bridging between WS-Addressing front ends and synchronous (traditional) back ends

- Bridging between traditional front ends and WS-Addressing back ends
- Routing in any direction between WS-Addressing-enabled front ends and back ends to include the case where the server response does not pass through the appliance

The DataPower service provides the mediation, if any, between the clients and servers. Support for a specific level of WS-Addressing does not preclude simultaneous support for traditional addressing formats. In other words, DataPower can provide a mediation service between WS-Addressing and other addressing methods.

- No WS-Addressing (default): This disables WS-Addressing support. Both clients and servers use traditional addressing. This requires no additional configuration.
- Synchronous gateway to WS-Addressing: This specifies that the DataPower service mediates addressing between clients that use traditional addressing and servers that use WS-Addressing.
- WS-Addressing gateway to synchronous: This specifies that the DataPower service mediates addressing between clients that use WS-Addressing and servers that use traditional addressing.
- Pure WS-Addressing: This specifies that the DataPower service mediates addressing between clients and servers that use WS-Addressing.

Configuring service level monitor

DataPower has the ability to throttle or shape traffic and log events based on runtime characteristics of traffic flowing through the appliance. The mechanism used by DataPower to achieve this are called service level monitor objects and assign a SLM to a Web Service Proxy. The SLM object has considerable flexibility as to where events are measured at different levels of granularity. There are pre-supplied blank templates that provide a customized framework to build a SLM unique to what is being measured, and they map accordingly to these below:

- A global SLM that monitors all Web Service Proxy transactions

  Use the Web Service Proxy template to create a global SLM to monitor all transactions and errors in the scope of the Web Service Proxy.
- A WSDL-specific SLM that monitors all services that are described in a specific WSDL file

  Use this template to create a WSDL-specific SLM to monitor all transactions and errors in the scope of a WSDL file. Note that if the Web Service Proxy is based upon a single WSDL file this template and the Services Proxy template are equivalent.
- A service-specific SLM that monitors a single Web service

  Use this template to create a WSDL-service-specific SLM to monitor all transactions and errors in the scope of a WSDL service.
- A port-specific SLM that monitors a single Web service port
- An operation-specific SLM that monitors a single Web service operation

  Use this template to create a WSDL-port-specific SLM to monitor all transactions and errors in the scope of a WSDL port.
- A custom SLM Statement that provides more precise control of monitored transactions

  Use this template to create a WSDL-operation-specific SLM to monitor all transactions and errors in the scope of a WSDL operation.

Each template supports the specification of a transaction-based rule and an error-based rule. The transaction-based rule is based on a raw count of transactions. The error-based rule is based on transaction errors. Also, there is dual rules capability, which implements level-specific restrictions based on transaction volume and error frequency.

Traffic throttling and shaping

The SLM objects are an ideal way to control traffic entering the edge of the network. There are two types of traffic control that DataPower utilizes for edge management:

- Traffic throttling: This is a simplified model of controlling throughput of messages by discarding packets that go over a certain threshold. DataPower's Limit field sets the threshold, and an interval is set for duration of throttling.
- Traffic shaping: An SLM that can improve delivery while maintaining SLAs on performance by protecting bandwidth.

Usage: There are unique use cases for traffic throttling versus traffic shaping:

- Traffic throttling can serve as a basic SLM to thwart denial-of-service attacks. Traffic counts can be fine tuned for more granularity to better control what gets throttled.
- Traffic shaping is an SLM that can improve delivery while maintaining SLAs on performance by protecting bandwidth. Note that this is *not* a guarantee of service. That must be accommodated for in another fashion if it is required.

Custom SLM statements

Usage of custom SLM statements provides considerably more control on the SLM object. Custom SLM statements can be grouped together with credential and resource classes and scheduler objects. Calendars or moving intervals can be applied. Industry standard algorithms can be established with the threshold type. These are:

- Count all (default): The threshold level is applied to the resources specified by a resource class.
- Count errors: The threshold is based on errors.
- Back-end latency: The threshold is based on server latency.
- Internal latency: The threshold is based on internal latency (processing time).
- Total latency: The threshold is based on the sum of measured latencies.

Usage:

- Use back-end latency when servers have trouble keeping up with demand.
- Count errors can become part of a larger error model. A notify event can generate a log entry that can be monitored by an SNMP runtime object that is part of the overall monitoring solution.
- Total Latency can be used in a use case where a performance SLA must be protected.

3.2.4 Policy management summary

Policy management is a subset of the overarching IBM SOA governance model. While seemingly specific to Web services, policy management can be applied outside of Web services. But it is Web services that provides such a solid and well-documented interface for policy management. It is the SLMs enabled by policy management that make such a strong case for Web services. DataPower is an appliance that is built specifically to operate within some key domains (that is, security and ReliableMessaging) defined by the WS-Policy Framework.

3.2.5 Resources

For further information see:

- *Patterns: Service-Oriented Architecture and Web Services*, SC24-6303

  http://www.redbooks.ibm.com/abstracts/sg246303.html?Open
- *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346

  http://www.redbooks.ibm.com/abstracts/sg246346.html?Open
- “First look at the WS-I Basic Profile1.0,” 2002

  http://www.ibm.com/developerworks/webservices/library/ws-basicprof.htm
- WC3, Web Services Addressing (WS-Addressing), 2004

  http://www.w3.org/Submission/ws-addressing/
- Web Services Security Policy Language (WS-SecurityPolicy)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf
- developerWorks, “developerWorks Forum,” 2007

  http://www.ibm.com/developerworks/forums/thread.jspa?messageID=14056243
- developerWorks, “developerWorks Forum,” 2007

  http://www.ibm.com/developerworks/library/specification/ws-rm/
- IBM, BEA, Microsoft®, TIBCO, “Web Services Reliable Messaging Protocol Specification,” 2005

  http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf
- Web Services Reliable Messaging

  http://www.ibm.com/developerworks/webservices/library/specification/ws-rm/
- Web Services Policy Framework

  http://www.ibm.com/developerworks/webservices/library/specification/ws-polfram/

  http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf
- Web Services Security Policy Language

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf
- Understand WS-Policy processing

  http://www.ibm.com/developerworks/webservices/library/ws-policy.html
- Web Services Reliable Messaging

  http://www.ibm.com/developerworks/webservices/library/specification/ws-rm
- Web Services Reliable Messaging Protocol (WS-ReliableMessaging)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf
- Web Services Policy Attachment

  http://www.ibm.com/developerworks/webservices/library/specification/ws-polatt/
- Web Services Policy Attachment (WSPolicyAttachment)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polatt/ws-polat-2006-03-01.pdf

3.3 Service registries

In order to provide the governance for a service's life cycle, you need a registry that manages metadata about the service artifacts and socializes the service, and a repository that owns the artifacts of the service. The type of artifacts that you need to govern varies, since services can be implemented as traditional Web services, EJB™, JMS, MQ, or even COBOL services. You also need the ability to manage policies and other artifacts attached to the services. In the case of a traditional Web service, you would store the WSDL and XSD documents in a registry/repository. You can then place these documents under governance and socialize to different communities of interest. Once the artifacts are published into the registry, it becomes your authoritative source for the service artifacts. You can trust it for accurate information about the service and its artifacts, such as their life-cycle state.

It is important that the registry be optimized for the demanding runtime environment. This allows the ESB to use the registry at run time for things like dynamic endpoint selection, policy retrieval and enforcement, and contract management. The best solution is one where both the registry capabilities and the repository capabilities are in one component, because you cannot govern what you do not own.

3.3.1 WebSphere Service Registry and Repository

WebSphere Service Registry and Repository (WSRR) enables one to store, access, and manage information about services and service interaction endpoint descriptions (referred to as service metadata) in an SOA. This information is used to select, invoke, govern, and reuse services as part of a successful SOA.

WSRR has support for runtime access from DataPower, including mechanisms to contact the registry at run time or interact with the registry during configuration. The following runtime access scenarios are good examples of how DataPower can interrogate the WSRR database to make critical runtime decisions based on policies and other relevant metadata archived in WSRR.

- Validation: access to schemas for validation of XML messages at configuration time. Principally relevant to the XML firewall role for DataPower appliances.
- Service virtualization: access to the registry at run time in order to acquire the actual endpoint based on a logical profile for the service.
- Policy determination: runtime or configuration time access to policies such as WS-Policy from the registry. This could be the XML firewall roles with different types of policy information being relevant to each.
- Availability/performance: runtime access to data enabling choices between implementations of services based on their availability and performance characteristics.
- Namespace translation: access to standard transformations between known namespaces. A good usage scenario would be promotional namespace changes based on the stage in the promotion life cycle. The WSRR state engine can be leveraged to provide the correct namespace based on whether one is in integration test phase versus production.

Usage: WSRR subscriptions are by far the most robust method supported by DataPower to enforce governance policies. There are several access methods that are supported by WSRR and its policy object abstraction. In addition, caching the subscription objects is the only reasonable runtime option for working with dynamic runtime governance policy objects. Otherwise, production scenarios are most likely utilizing static WSDL objects defined in the file system. Remote lookup of UDDI repositories will probably introduce unacceptable latency issues without pre-caching.

3.3.2 WSRR governance model

The WSRR governance module provides a simple configurable life-cycle model that can be used to manage governed entities (services and other assets) through a SOA life cycle. This is represented as a state machine with the states indicating the position of the service/asset in its life cycle. Transitions are used to validate changes to the governed entities and apply access control before performing the action represented by the transition. Following a successful transition the governed entity then adopts a new state.

During governance processes, users will perform actions on the governed entities (services, assets) in the WSRR subject to access control. These actions are constrained by the life-cycle model, thus ensuring that governance and changes in state are socialized to other users/systems where needed and audited for assurance of the governance processes.

Typical scenarios that are discoverable by DataPower (and managed by WSRR) to support enablement of governance include:

- How do you organize the shared services/assets so that they can be effectively reused at a later date?

  The answer is WSRR governed entity and metadata. This is covered by the governed entities within the WSRR. Using the breadth of the content model to represent the services/assets and adding metadata such as classifications, properties, and relationships helps customers organize their services. In addition, the registry needs to deal with the granularity of governed entities since it is essential to manage the high-value services and assets as a whole rather than micro-managing each individual asset or document. Thus, the governed entity needs to aggregate a number of artifacts that all progress through the same life cycle together.

- Who is allowed to change a service reused by others?

  The answer is WSRR access control. The WSRR needs to provide very clear access control to say which users can do what to each service (governed entity). This depends on the ownership of the entity within the organization, the role of the user, the action or change that they want to perform, and the life-cycle state that the service is in.

- Who is using a service and what will be impacted by changes to that service?

  The answer is WSRR governance actions and impact analysis. The WSRR maintains relationships between entities within the registry including between governed entities. Impact analysis can be used to traverse these dependencies and evaluate the dependent entities that may be affected by a change. Different types of analysis will be needed for different actions, and the socialization of future and immediate changes is necessary in order to ensure consistency in the operational environment.

- Who needs to approve any changes?

  The answer is WSRR access controls manage who can approve (or perform other actions) and auditing provides accountability. The life-cycle model together with access controls provide the means of defining who can do what to each type of entity.

Determining this is not a job for the WSRR but for the process developer. However, once the governance process is deployed, the WSRR needs to record who approved what changes for auditing purposes and to prove that the documented processes are being followed.

See Figure 3-2 for a graphical depiction on the governance capabilities of WSRR.

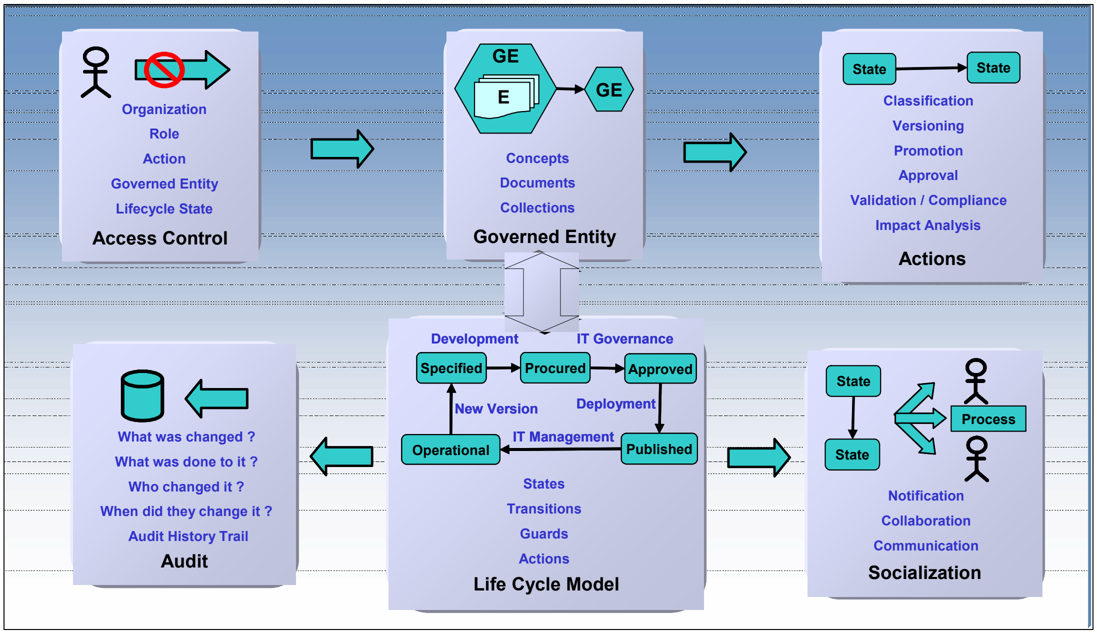


Figure 3-2 WebSphere service registry and repository governance

Governance enablers: Service life cycle overview

The life cycle of a service describes the permissible states and actions in each state, from design to retirement. The default service life cycle, with full state management, is installed with WSRR, and is based upon the IBM SOA Foundation. In addition, WSRR allows developers to design a custom life cycle and can define the conditions in which a service can move from one state to the next (Figure 3-3).

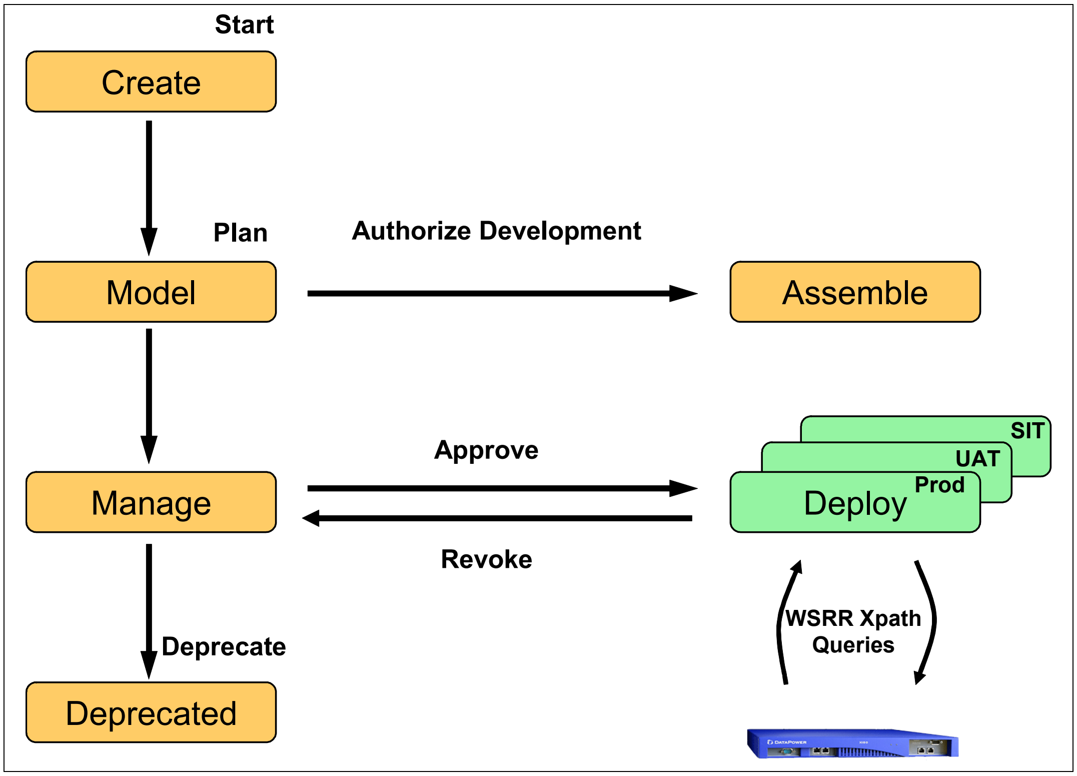


Figure 3-3 WSRR default life cycle

From a DataPower perspective (runtime and management), we are interested in the interactions that WSRR has with the deploy and manage service life cycle.

WSRR in the deploy and run phases

From a DataPower perspective, such as a mediation device that is optimized for WSRR lookup, dynamic endpoint selection is a key use case that involves querying WSRR for the service provider or a set of candidate service providers, and binding to the endpoint that is the best choice. With this scenario, endpoint selection rules are encoded in ESB-managed mediations of DataPower.

Another use case is dynamic retrieval and enforcement of the policies that are in effect for a service interaction in the areas of logging, filtering, data transformation, or routing. WSRR stores the policy definitions as well as information about which service interaction endpoints they apply to. Policy enforcement points can be configured using that information and can be updated when WSRR-managed metadata changes.

DataPower, as a SOA runtime element, can maintain configuration information beyond the minimalist set of service metadata managed by WSRR, and can merge with cached WSRR-managed metadata and relate it to the additional configuration information and policy data.

WSRR in the manage phase

Service monitoring and management products like ITCAM for SOA provide instrumentation of service interactions that allow monitoring of service interactions and service endpoint behavior. Summary information about service behavior can be pushed into WSRR to decorate service endpoint metadata with execution statistics. Decorated service endpoint metadata can then be used by mediation devices such as DataPower to re-configure policy enforcement points that implement the SLAs that users want to enforce in service interactions.

WSRR subscription method

From the point-of-view of the DataPower appliance, WSRR provides functional equivalency to a UDDI registry as a source of Web Service Proxy configuration data. DataPower provides GUI and a command-level interface (CLI) for configuration. Access must first be provided to the WSRR server, then to the services from that server.

A significant configuration parameter for both the server and a service object is the synchronization method of polling versus manual. DataPower supports caching of WSRR services. It is through caching that WSRR can provide dynamic runtime endpoint resolution. Otherwise, without caching there would be too much latency in the XPath lookup. If caching is turned on but the polling is infrequent, the local cache could become stale, so it is important to cache WSRR locally with a refresh synchronization parameter that is frequent enough to match changes to the WSRR data store.

3.3.3 The need to integrate WSRR with UDDI

Why integrate a UDDI registry with WSRR? This section describes some of the advantages to integrating a UDDI registry with WSRR. Though this topic is comparative in nature, DataPower must be aware of the level of integration between the two as it impacts WSDL configuration and policy invocation.

Advanced runtime capabilities

The UDDI registry was originally created solely for design or development time look-up by humans. UDDI is not suitable for dynamic look-ups by today's demanding runtime environments. Since everything in UDDI is a tModel that only references the SOA artifacts, it can take multiple requests to the UDDI registry to retrieve the information that the runtime needs. You also cannot selectively choose what information from the tModel you wish to retrieve, which impacts performance. Integrating your existing UDDI registry with, or migrating to, WSRR gives you the runtime capabilities required in today's SOA.

In addition to the UDDI APIs, WSRR has a robust set of APIs, both Java and Web services, that allow you to create, retrieve, update, and delete content in WSRR. WSRR query support is based on XPath, which gives you the ability to retrieve the content that you need in one remote request to the registry instead of several requests, as required in UDDI. WSRR also allows you to retrieve only the content that you need, reducing the amount of information that you are streaming and caching over the wire.

In addition to the robust API that can be used by any ESB, WSRR has out-of-the-box integration with WebSphere DataPower. DataPower provides primitive mediations such as the EndpointLookup, queue look-ups, and registry content retrieval, allowing DataPower to fully leverage WSRR's capabilities to retrieve runtime policies that can then be enforced.

Trusted authoritative source

DataPower can query and make runtime decisions by leveraging WSRR's integrated registry and repository capabilities, which means that the artifacts that WSRR owns in its repository are discoverable by DataPower. This capability gives you a trusted source for your SOA artifacts, since the artifacts cannot be changed or moved without the registry's knowledge. When UDDI is integrated with WSRR, the overview URL in UDDI points to the artifact inside WSRR. When artifacts in WSRR change, the UDDI registry is notified and updated. DataPower does not have any ownership of duties in these syncing events, as WSRR plug-in modules manage this activity.

As discussed, DataPower by default can use file-based, static WSDL documents. But with WSRR it now can leverage the WSRR service discovery mechanism, which automatically searches a target application environment for services, and then loads the corresponding WSDL documents into WSRR. The service discovery mechanism also finds service component architecture (SCA) modules and loads them into WSRR. The service discovery mechanism updates the metadata in WSRR if the previously discovered services are uninstalled. This enables WSRR to be the authoritative source for all services, rogue and governed, in your SOA environment.

Enforceable life cycle

UDDI does not have an enforceable life cycle for artifacts. One cannot really tell what the state of an artifact is, or ensure that policies are met, before an artifact transitions states. Integrating an existing UDDI registry with or migrating to WSRR gives one an enforceable life cycle for artifacts that DataPower can interact with. In this scenario, WSRR manages the state management of the services, and DataPower can rely on WSRR to manage the state, but provide necessary metadata that WSRR needs to make state-changing decisions.

Integrating with UDDI

WSRR uses a push-and-pull model to integrate with a UDDI V3.0.2 registry. The UDDI synchronization operations are performed by the WSRR-UDDI synchronization module. When documents are created, updated, or deleted in WSRR, a WSRR notification event is generated. The synchronization model carries out repeated polls of WSRR and processes these created, updated, and deleted events by performing the corresponding operations in the UDDI registry using the standard UDDI Publishing API. The synchronization model also carries out repeated polls of the UDDI registry and makes the appropriate changes in WSRR using the Service Registry API.

As an implementation device of Web services, DataPower does not have direct responsibility with the synchronization of WSRR and UDDI, but there are mapping considerations that must be taking into account that impact how the service or its policy is queried against either repository type (Figure 3-4).

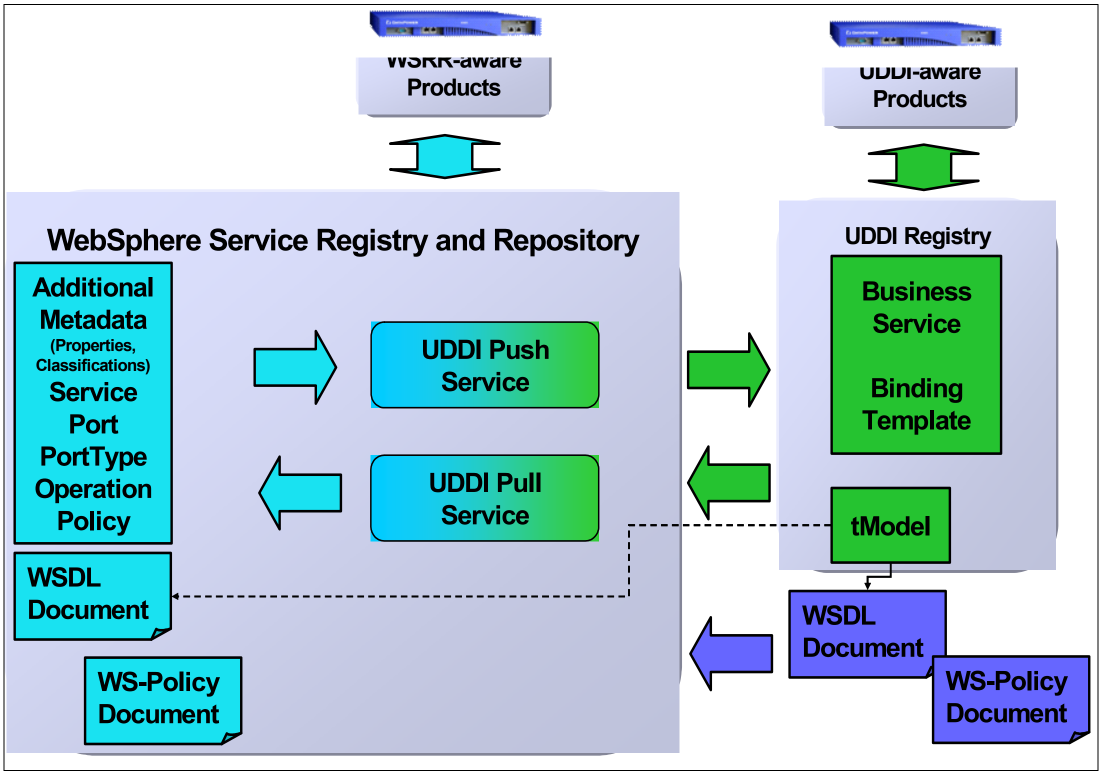


Figure 3-4 WSRR/UDDI integration

3.3.4 Querying WebSphere Service Registry and Repository

In this section we discuss querying WebSphere Service Registry and Repository.

Concept and bsrURI

DataPower has an intelligent interface to WebSphere Service Registry and Repository in that it is aware of metadata that is unique to WSRR, which facilitates searching for services in its registry. DataPower is concept aware, a WSRR abstract object of WSDLs. But concepts may or may not select a unique WSDL, and a WSDL name by itself will not guarantee uniqueness. Therefore, DataPower is also aware of the bsrURI—a specific URI assigned to each WSDL doc. The bsrURI will guarantee uniqueness and can act with the efficiency of an indexed key. But the bsrURI is an internal WSDL key whose uniqueness only has meaning to a single registry. Thus, the client must have an understanding of what the bsrURI is associated with to have the correct context. A typical use case would be issuing a XPath search with parameters that could include any of the following: a specific WSDL name, a concept, a promotional state, a version number, a policy statement, or any other attributes that have contextual meaning. Then after testing for one and only one return, reference the associated bsrURI from that point forward.

Security

As with all server communications that interact with DataPower, SSL is supported. We strongly recommend that each WSRR server connection be associated with a SSL Proxy profile.

Access methods

WSRR can store valuable service metadata that an application needs at run time to make decisions, such as where to route a message or what operation to execute upon receiving a message. In the role of a competent mediation appliance, DataPower can make intelligent queries for information from WSRR (such as WSDL information, owner of a service, or classification of a service).

Figure 3-5 illustrates the use case of DataPower querying WSRR for runtime metadata. The architecture benefits of interjecting the flow with a mediation device is expounded upon in 3.3.5, "Dynamic endpoint lookup" on page 51. This enables key runtime information to be externalized from the DataPower configuration.

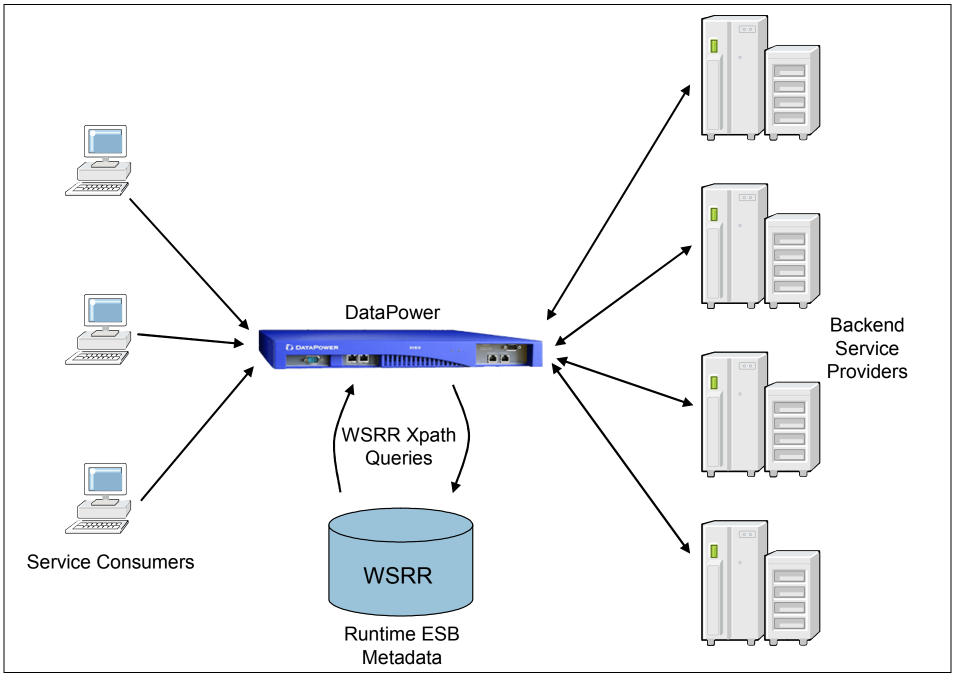


Figure 3-5 Service registry and DataPower topology

Access via SOAP

IBM supports two flavors of APIs for interfacing with WSRR: a Java-based API and a SOAP-based API. Both support publishing (creating and updating) service metadata artifacts and metadata associated with those artifacts, retrieving service metadata artifacts, deleting the artifacts and their metadata, and querying the content of WSRR.

XPath expressions in the Query API perform searches for coarse-grained and fine-grained services and artifacts. Queries can be performed using semantic annotations, properties, and all or parts of physical service metadata artifacts. Fragments of metadata can be requested to be returned (such as endpoints), all metadata to be returned, and both metadata and documents to be returned.

DataPower uses the SOAP-based API to query WSRR as a default behavior (Example 3-1).

Example 3-1 Using a SOAP API to query WSRR

```
amp;<xsl:stylesheet version="1.1"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:dp="http://www.datapower.com/extensions"
    xmlns:dpquery="http://www.datapower.com/param/query"
    extension-element-prefixes="dp"
    exclude-result-prefixes="dp">

<xsl:param name="dpquery:host" select="'WSRR'" />
<xsl:param name="dpquery:port" select="'9080'" />
<xsl:param name="dpquery:xpath-query" />
<xsl:param name="dpquery:comma-separated-properties" />
....
<xsl:template match="/">

<xsl:variable name="soap-query">

    <soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
    <executeQuery
        xmlns="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/ws/sdo">
    <datagraph xmlns="commonj.sdo">
    <WSRR
        xmlns="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo">
    <root>_1</root>
    <xsl:choose>
    <xsl:when
        test="string-length($dpquery:comma-separated-properties) > 0">
    <artefacts
        xmlns="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo"
        xsi:type="PropertyQuery" bsrURI="_1"
        queryExpression="{$dpquery:xpath-query}">

        <xsl:call-template name="process-properties">
            <xsl:with-param name="properties-string"
                select="$dpquery:comma-separated-properties" />
            <xsl:with-param name="count" select="1" />
        </xsl:call-template>
    </artefacts>
    </xsl:when>
    <xsl:otherwise>
    <artefacts
        xmlns="http://www.ibm.com/xmlns/prod/serviceregistry/6/0/sdo"
        xsi:type="GraphQuery" bsrURI="_1"
        queryExpression="{$dpquery:xpath-query}" />
    </xsl:otherwise>
    </xsl:choose>
    </WSRR>
    </datagraph>
    </executeQuery>
    </soap:Body>
    </soap:Envelope>

</xsl:variable>
```

```
<xsl:copy-of select="dp:soap-call(
 concat('http://',$dpquery:host,
    ':',$dpquery:port,
    '/WSRRCoreSDO/services/WSRRCoreSDOPort')
 ,$soap-query,'',0,'WSRRCoreSDO')"/>

</xsl:template>
....
```

Notice the parameters in this stylesheet, including one called *comma-separated-properties*, to choose between a graph query or a property query. Also notice that a soap-call() extension function is used to make the actual query to WSRR.

Access via REST

It is also possible to query WSRR with a REST interface from DataPower.

Further reading: Refer to "Using DataPower SOA Appliances to query WebSphere Service Registry and Repository" at:

http://www.ibm.com/developerworks/websphere/techjournal/0805_peterson/0805_peterson.html

3.3.5 Dynamic endpoint lookup

The Web services implementation of SOA provides for a loose coupling between service requestor providers. Via the WSDL, decoupling the service implementation from the service interface is possible, so that changing the provider's service implementation does not require any changes to the service requestor. This issue with this static architecture is a static endpoint provided by the service description (WSDL) that means coupling the requestor and provider at the transport level (Figure 3-6).

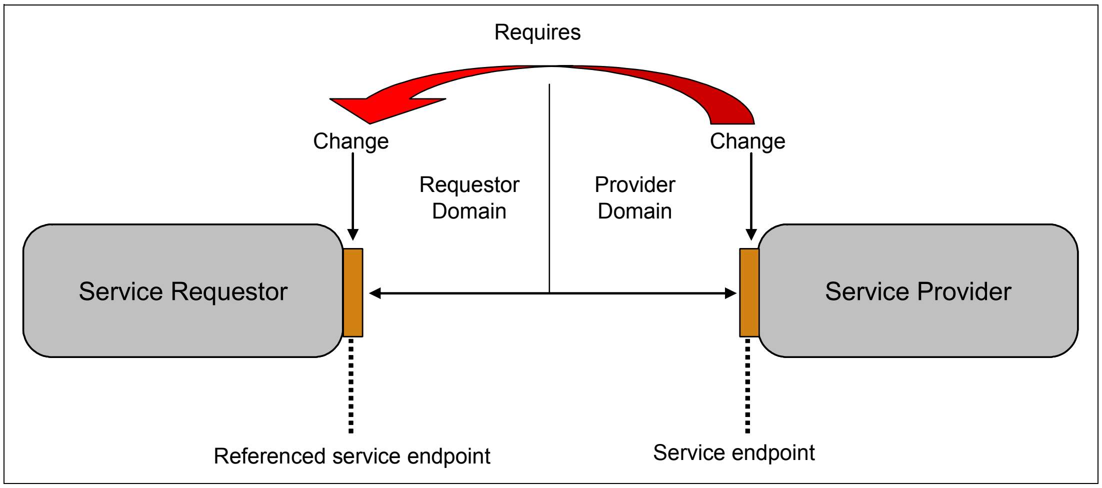


Figure 3-6 Transport-level coupling

Every time the physical endpoint of the service provider changes, every service consumer must reflect the changes in its application logic. DataPower can provide a better solution acting as an intermediary that intercepts a service call and routes it to the appropriate endpoint. Abstracting from transport-specific information to have a transparent intermediary layer between service consumers and providers is one aspect of the enterprise service bus (ESB) pattern (see 5.1.1, "DataPower as an ESB" on page 126). DataPower's Web Service

Proxy implements a SOA-based ESB pattern by acting as such an intermediary layer. Implementing this part of an ESB layer in a static way by referencing WSDL files in the WS-Proxy component and providing a static service endpoint to the consumer provides a single point of control when changes are made to the real service endpoint, thus simplifying management of the service endpoint. With this approach, the service consumer is decoupled from the provider at the transport level, but changes on the proxy side are still necessary when changes are made on the provider's endpoint (Figure 3-7).

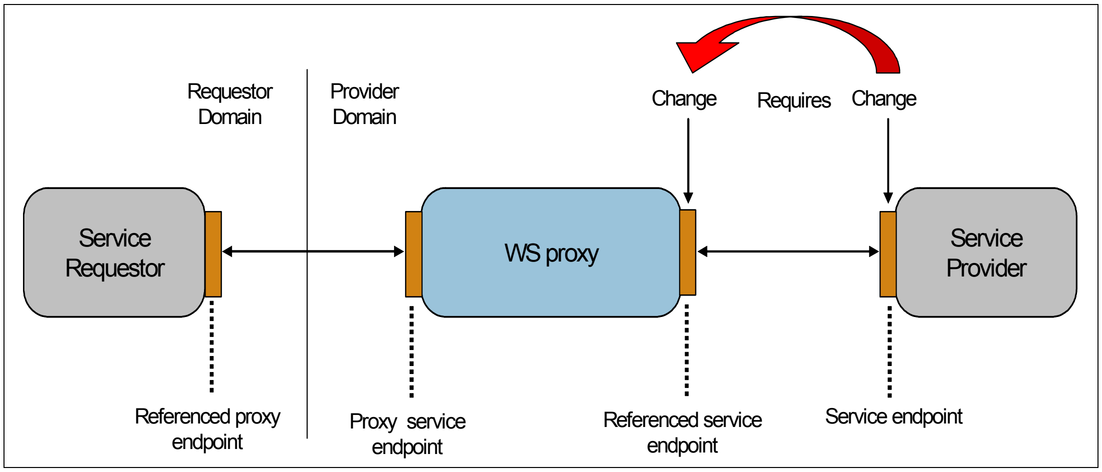


Figure 3-7 Decoupling with WS-Proxy

Using DataPower's WebSphere Service Registry and Repository subscription feature inside the service proxy instead of statically referencing the WSDL file is the way to get a more dynamic service intermediary. DataPower can subscribe to WSDL files or to other artifacts stored in WebSphere Service Registry and Repository, and synchronize with changes either automatically by polling or manually driven from outside. From a service endpoint management point of view, this approach provides a flexible and dynamic ESB layer functionality that reflects changes to the endpoints automatically without having to make changes on the service consumer or on the ESB side (Figure 3-8).

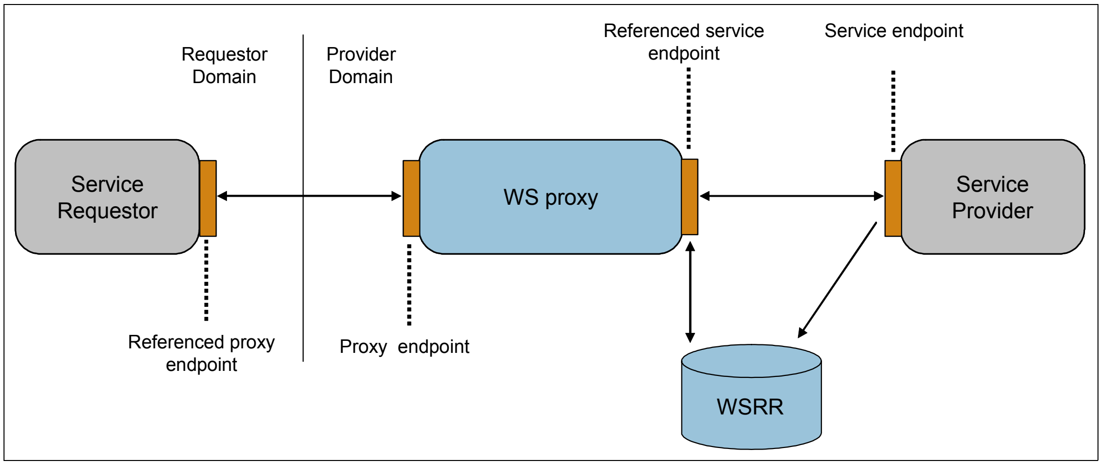


Figure 3-8 Dynamic decoupling with WS-Proxy

Further reading: See "Managing services dynamically using WebSphere DataPower SOA Appliances with WebSphere Service Registry and Repository" at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0802_rohmann/0802_rohmann.html

3.3.6 UDDI service registries

Standard UDDI access methods are supported. WSDLs can be discovered in a UDDI registry with a subscription method. This is achieved by specifying subscription key fields, with two usually required, one for the bindingTemplate and one for the tModel. There is no caching mechanism available for UDDI lookups.

For subscribing to UDDI registries, the following basic Web services subscription pattern defines Web services interactions among service requesters, service providers, and service directories (UDDI) as follows:

1. Service requesters find Web services in a UDDI service directory.
2. They retrieve WSDL descriptions of Web services offered by service providers that previously published those descriptions to the service directory.
3. After the WSDL has been retrieved, the service requester binds to the service provider by invoking the service through SOAP (Figure 3-9).

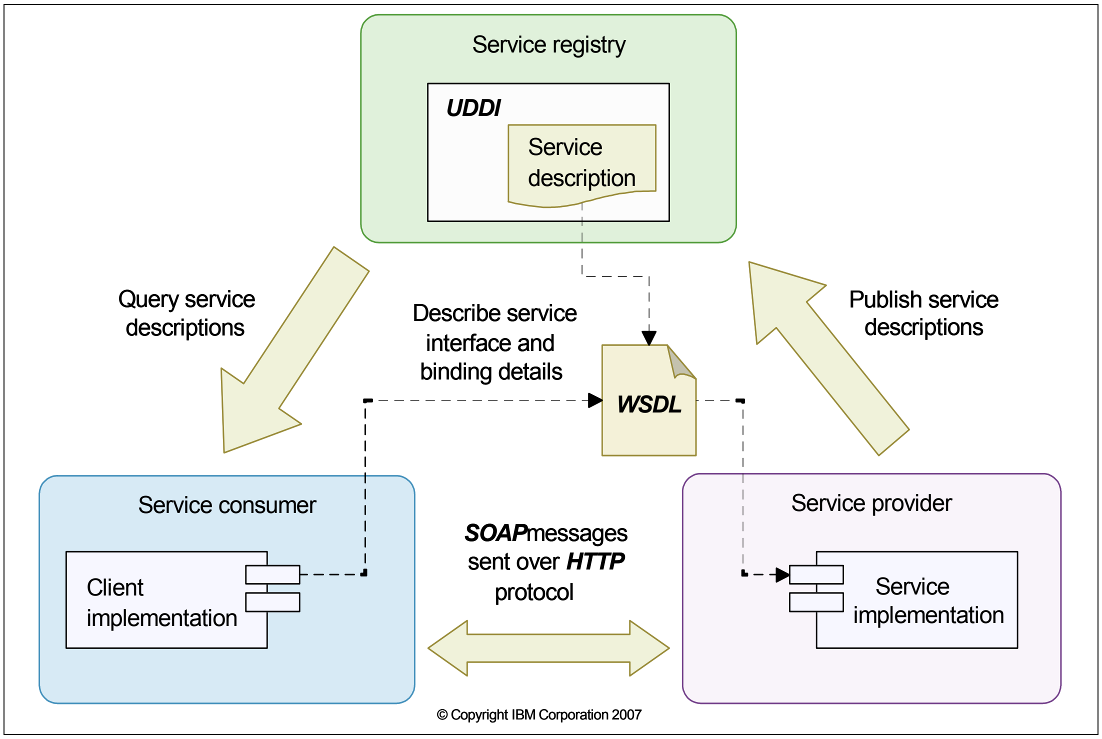

Service registry
UDDI
Service description
Query service descriptions
Describe service interface and binding details
WSDL
Publish service descriptions
Service consumer
Client implementation
SOAPmessages sent over HTTP protocol
Service provider
Service implementation
© Copyright IBM Corporation 2007

Figure 3-9 Generic UDDI service registry

But this pattern is highly coupled. DataPower would be inserted as a mediation device to the services UDDI services repository, shielding and detaching the consumer from the services lookup and proxying the services interface (Figure 3-10).

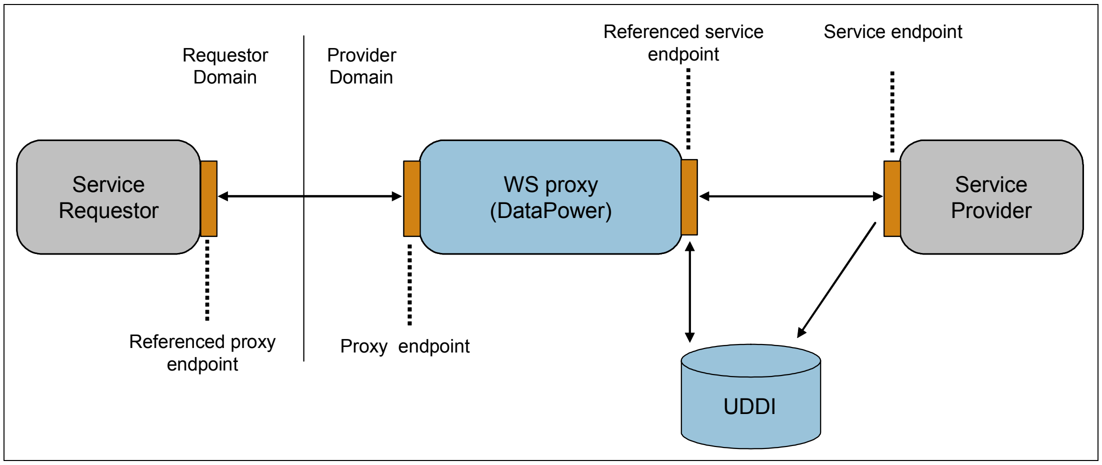


Figure 3-10 DataPower proxy to UDDI

3.3.7 Service registries summary

WSRR is the master metadata repository for service interaction endpoint descriptions in a IBM Web services solution space. As the integration point for service metadata, WSRR establishes a central point for finding and managing service metadata that can support dynamic endpoint lookups.

DataPower can subscribe to UDDI repositories, which are accessed via industry open standards XPath methods.

3.3.8 Resources

For more information refer to the following resources:

- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366

  http://www.redbooks.ibm.com/abstracts/redp4366.html?Open
- *WebSphere Service Registry and Repository Handbook*, SG24-7386
- Introduction to the IBM SOA Programming Model

  http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel/
- Synchronize UDDI registries with WebSphere Service Registry and Repository for better SOA governance

  http://www.ibm.com/developerworks/websphere/library/techarticles/0802_olson/0802_olson.html
- IBM WebSphere Service Registry and Repository Version 6.1 information center

  http://publib.boulder.ibm.com/infocenter/sr/v6r1/index.jsp
- Using DataPower SOA Appliances to query WebSphere Service Registry and Repository

  http://www.ibm.com/developerworks/websphere/techjournal/0805_peterson/0805_peterson.html

- A day in the life of WebSphere Service Registry and Repository in the SOA life cycle

  http://ltsbwass001.sby.ibm.com/cms/developerworks/websphere/library/techarticles/0609_mckee/0609_mckee.html

- Introducing IBM WebSphere Service Registry and Repository, Part 2: Architecture, APIs, and content

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0609_mckee2/0609_mckee2.html

3.4 Web services scenarios: A WSRR scenario

To illustrate the principles of a services subscription with WSRR from DataPower, refer to *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366:

http://www.redbooks.ibm.com/abstracts/redp4366.html?Open

3.4.1 Request-response Web Service scenario

To illustrate the principles of WS-Policy in a request-response message exchange, assume the following characteristics:

- BuyMyProduct.com is a company that offers their services through Web services.
- BuyMyProduct.com currently offers the product quote service (WSDL-Quote) and plans to offer a gold version of the product quote service (WSDL-Quote-Gold). BuyMyProduct.com offers both services through simple service interfaces.
- The participants in the request-response message exchange are the customer (the requesting client) and the Web service endpoint (the responding service by the BuyMyProduct.com provider).
- BuyMyProduct.com developed, tested, and deployed their WSDL-Quote service on a WebSphere Application Server (WAS) server. Customers can send a request to the WSDL-Quote service to query the price of a product. The request message contains the name of the product. The service responds with the current price information (Figure 3-11).

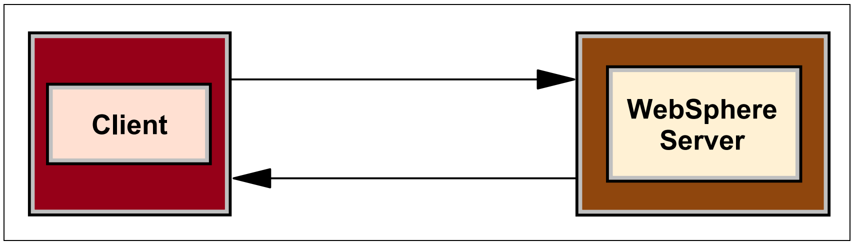


Figure 3-11 Request-response message exchange to WebSphere Application Server

- Several BuyMyProduct.com customers have asked for an enhancement to the price quote service to respond with the current inventory level and the customer-specific committed price for the product. BuyMyProduct.com developed their new WSDL-Quote-Gold service and wants to offer this new service while continuing to support their WSDL-Quote service.
- The WSDL-Quote-Gold service is for privileged customers who register for the value-add service. When a customer sends a request to the WSDL-Quote-Gold service, the service responds with the price quote, the available discounts, and the current inventory level. To

support the discounts, the request and the response need to be signed and encrypted as well as contain a time stamp.

- At this same time, BuyMyProduct.com makes the decision to proxy both services on a DataPower device. The decision was to provide a single point for authentication and message security on the DataPower device while allowing the WAS engine to focus on application logic crunching (Figure 3-12).

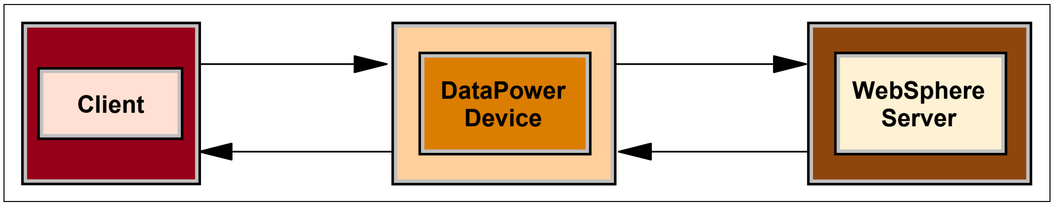


Figure 3-12 Request-response exchange through DataPower to WebSphere Application Server

The policy for the WSDL-Quote service indicates that the request must contain a UsernameToken in a WS-Security header. Because there is no attempt to verify the identity of the user, the policy is appropriate. The UsernameToken is important only for calculating the number of requests from a particular user.

The policy for the WSDL-Quote-Gold service indicates that the request must contain a UsernameToken and message protection. The request and the response must be signed and encrypted. The request must contain a UsernameToken in a WS-Security header. The response does not contain the UsernameToken.

To define the required policy for these services, an administrator to the DataPower device augments the base policy in the source WSDL files to include XML policy expressions in the provided policy templates. These templates contain the required WS-SecurityPolicy assertions to implement policy. These templates represent common security patterns.

UsernameToken policy expression without password

The policy for the WSDL-Quote service indicates that the request must contain a UsernameToken in a WS-Security header. Because there is no attempt to verify the identity of the user, the policy is appropriate. The UsernameToken is important only for calculating the number of requests from a particular user.

The following code excerpt shows the policy expression to capture a behavior that includes a UsernameToken (Example 3-2).

Example 3-2 Capturing a username token

```
<wsp:Policy xmlns:wsp="..." xmlns:sp="...">
<sp:SupportingTokens> <wsp:Policy> <sp:UsernameToken> <wsp:Policy> <sp:NoPassword/>
</wsp:Policy> </sp:UsernameToken> </wsp:Policy> </sp:SupportingTokens> </wsp:Policy>
```

This policy expression results in a message with the pattern shown in Example 3-3.

Example 3-3 Policy expression results

```
<?xml version="1.0" encoding="utf-8" ?> <soap:Envelope xmlns:soap="..."> <soap:Header>
<wsse:Security soap:mustUnderstand="1" xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
oasis-200401-wss-wssecurity-secext-1.0.xsd"> <wsse:UsernameToken>
<wsse:Username>Shiu-Fun</wsse:Username> </wsse:UsernameToken> </wsse:Security>
</soap:Header> <soap:Body> ... </soap:Body> </soap:Envelope>
```

Example 3-4 shows the required content to support the policy.

Example 3-4 Required content to support policy

```
<?xmlversion="1.0" encoding="UTF-8"?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512">
<dpe:summary xmlns="" xmlns:dpe="http://www.datapower.com/extensions">
<dppolicy:domain xmlns:dppolicy="http://www.datapower.com/policy">
http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512
</dppolicy:domain>
<description>
Implements WS Security Policy 1.1- UsernameToken 1.0 and 1.1 support
</description>
</dpe:summary>

<wsp:ExactlyOne>

<!--UsernameToken 10-->
<wsp:All>
<sp:SupportingTokens>
<sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
ws-sx/ws-securitypolicy/200512/IncludeToken/Always">
<wsp:Policy>
<sp:WssUsernameToken10 />
</wsp:Policy>
</sp:UsernameToken>
</sp:SupportingTokens>
</wsp:All>

<!--UsernameToken 1.1 -->
<wsp:All>
<sp:SupportingTokens>
<sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
ws-sx/ws-securitypolicy/200512/IncludeToken/Always">
<wsp:Policy>
<sp:WssUsernameToken11 />
</wsp:Policy>
</sp:UsernameToken>
</sp:SupportingTokens>
</wsp:All>
</wsp:ExactlyOne>
</wsp:Policy>
```

Configuring on the DataPower device

The following procedure results in a policy to apply to the WSDL (the sp:UsernameToken assertion to the WSDL-Quote WSDL). Messages that the DataPower device processes will check for the presence of a UsernameToken in each service request that it receives.

Instead of an administrator manually editing the WSDL file, the DataPower WebGUI allows administrators to abstract the tasks of authoring XML policy expressions and allows them to select different policy elements from common sets that are kept in XML files in the DataPower file system.

The policy on the DataPower device checks for the presence of a UsernameToken header in request messages. Only messages that meet this criteria are forward to the WAS service.

Message protection policy

For the WSDL-Quote-Gold service, there are additional configuration steps on the DataPower device to support the more complex policy. This example requires that the key material for the sign and encrypt actions are available on the DataPower device in addition to the required configuration of the UsernameToken policy.

We assume that:

- BuyMyProduct.com has a public-private key pair that was issued by a sanctioned Certificate Authority (CA). This key pair is in use for securing messages within the existing J2EE environment.
- Developers of the J2EE application created a key pair and certificate. The key pair is stored in a custom Java keystore.

A public-private key pair is useful in this scenario, because the issuing party in a message exchange can sign or encrypt a message using the private key and can send its public key (or a reference to its public key) to the relying party with the message. The relying party can verify or decrypt the message with the public key. (Because the issuing party has the private key, the relying party knows that the message came from the issuing party.)

In this scenario, the requestor is the issuing party and BuyMyProduct.com is the relying party.

When the BuyMyProduct.com receives a message from the requestor, the message will be signed and encrypted by the requester with the private key (PRK1). The BuyMyProduct.com needs to have the public key (PBK1) from the requestor to verify and decrypt the message.

When BuyMyProduct.com sends a message to the requester, the message will be signed and encrypted by the BuyMyProduct.com with the private key (PRK2). The requestor needs to have the public key (PBK2) from BuyMyProduct.com to verify and decrypt the message.

When a customer registers for the gold service, guidelines are in place to exchange the required keys to manage the cryptographic actions. For illustrative purposes, the key pairs are:

- PRK1/PBK1 (for the issuing party, the requestor)
- PRK2/PBK2 (for the provider or relying party, BuyMyProduct.com)

Example 3-5 shows the required content to support the policy.

Example 3-5 Required content to support policy

```
<?xmlversion="1.0" encoding="UTF-8" ?>
<wsp:Policy xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
   xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512"
   xmlns:spe="http://www.ibm.com/xmlns/prod/websphere/200605/
      ws-securitypolicy-ext"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xmi=http://schema.omg.org/spec/XMI/1.0
   xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-wssecurity-utility-1.0.xsd">
   <dpe:summary xmlns="" xmlns:dpe="http://www.datapower.com/extensions">
      <dppolicy:domain xmlns:dppolicy="http://www.datapower.com/policy">
         http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200512
      </dppolicy:domain>
         <description>
            Implements WASWebServiceUsername WSSecurity default
         </description>
   </dpe:summary>
```

```
<wsp:Policy wsu:Id="binding-policy">
   <wsp:ExactlyOne>
      <wsp:All>
         <sp:AsymmetricBinding>
            <wsp:Policy>
               <sp:InitiatorToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/
                        ws-sx/ws-securitypolicy/200512/IncludeToken/
                           AlwaysToInitiator">
                        <wsp:Policy>
                           <wsp:WssX509V3Token10 />
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:InitiatorToken>
               <sp:RecipientToken>
                  <wsp:Policy>
                     <sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/
                        ws-sx/ws-securitypolicy/200512/IncludeToken/
                           AlwaysToRecipient">
                        <wsp:Policy>
                           <wsp:WssX509V3Token10 />
                        </wsp:Policy>
                     </sp:X509Token>
                  </wsp:Policy>
               </sp:RecipientToken>
               sp:AlgorithmSuite>
                  <wsp:Policy>
                     <sp:Basic128Rsa15 />
                  </wsp:Policy>
               </sp:AlgorithmSuite>
               <sp:Layout>
                  <wsp:Policy>
                     <sp:Lax/>
                  </wsp:Policy>
               </sp:Layout>
            <sp:IncludeTimestamp />
         </wsp:Policy>
      </sp:AsymmetricBinding>
         <sp:EncryptedSupportingTokens>
            <wsp:Policy>
               <sp:UsernameToken sp:IncludeToken="http://docs.oasis-open.org/
                  ws-sx/ws-securitypolicy/200512/IncludeToken/AlwaysToRecipient">
                  <wsp:Policy>
                     <sp:WssUsernameToken10 />
                  </wsp:Policy>
               </sp:UsernameToken>
            </wsp:Policy>
         </sp:EncryptedSupportingTokens>
      </wsp:All>
   </wsp:ExactlyOne>
</wsp:Policy>
   <wsp:Policy wsu:Id="request_parts">
      <sp:SignedParts>
         <sp:Body />
         <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
         <sp:Header Namespace="http://www.w3.org/2005/08/addressing" />
      </sp:SignedParts>
      <sp:SignedElements>
```

```
         <sp:XPath>/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         andlocal-name()='Envelope']/*[namespace-uri()='http://schemas.xml
         soap.org/soap/envelope/' andlocal-name()='Header']/*[namespace-uri()
         ='http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
         secext-1.0.xsd' andlocal-name()='Security']/*[namespace-uri()='http:
         //docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
         1.0.xsd' andlocal-name()='Timestamp']   </sp:XPath>
         <sp:XPath>/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and
         local-name()='Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/
         soap-envelope' andlocal-name()='Header']/*[namespace-uri()='http://
         docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0   .xsd'and
         local-name()='Security']/*[namespace-uri()='http://docs.oasis
         -open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd' and
         local-name()='Timestamp']
         </sp:XPath>
      </sp:SignedElements>
      <sp:EncryptedParts>
         <sp:Body />
      </sp:EncryptedParts>
   </wsp:Policy>
   <wsp:Policy wsu:Id="response_parts">
      <sp:SignedParts>
         <sp:Body />
         <sp:Header Namespace="http://schemas.xmlsoap.org/ws/2004/08/addressing" />
         <sp:Header Namespace="http://www.w3.org/2005/08/addressing" />
      </sp:SignedParts>
      <sp:SignedElements>
         <sp:XPath>/*[namespace-uri()='http://schemas.xmlsoap.org/soap/envelope/'
         andlocal-name()='Envelope']/*[namespace-uri()='http://schemas.xmlsoap
         .org/soap/envelope/' andlocal-name()='Header']/*[namespace-uri()='http
         ://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0   .xsd
         'andlocal-name()='Security']/*[namespace-uri()='http://docs.oasis
         -open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd' and
         local-name()='Timestamp']
      </sp:XPath>
      <sp:XPath>/*[namespace-uri()='http://www.w3.org/2003/05/soap-envelope' and
         local-name()='Envelope']/*[namespace-uri()='http://www.w3.org/2003/05/
         soap-envelope' andlocal-name()='Header']/*[namespace-uri()='http://
         docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0   .xsd'and
         local-name()='Security']/*[namespace-uri()='http://docs.oasis
         -open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd' and
         local-name()='Timestamp']
      </sp:XPath>
   </sp:SignedElements>
   <sp:EncryptedParts>
      <sp:Body />
   </sp:EncryptedParts>
   </wsp:Policy>
</wsp:Policy>
```

Configuring the DataPower device

The following procedure results in a policy to apply to the WSDL (the sp:UsernameToken assertion to the WSDL-Quote-Gold WSDL). Messages processed by the DataPower device perform the following tasks:

1. Check for the presence of a UsernameToken in each service request that it receives.
2. Sign the body of the request message.
3. Encrypt the body of the request message.

4. Sign the headers of the request message with the public key (PRK1) and public key (PUBK1) for the requestor.
5. Sign the body of the response message.
6. Encrypt the body of the response message.
7. Sign the headers of the request message with the public key (PRK2) and public key (PUBK2) for the provider or relying party.

Instead of an administrator manually editing the WSDL file, the DataPower WebGUI allows administrators to abstract the tasks of authoring XML policy expressions and allows them to select different policy elements from common sets that are kept in XML files in the DataPower file system.

Assumption: There are two key pairs that are needed for this message exchange, because BuyMyProduct.com must sign and encrypt the information that it sends back in the response to the requesting client. This required key material is already uploaded to the DataPower device.

The configuration needs an identification credential and validation credential that were created from the key material in the Java keystore.

To create the productQuoteService Web Service Proxy to use this policy, use the following procedure:

1. Log in to the WebGUI.
2. Click the Web Service Proxy icon on the control panel.
3. Select **productQuoteService** from the catalog.
4. Click the **Policy** tab.
5. Locate the WSDL-Quote-Gold service and click **WS-Policy**.
6. Click the **Sources** tab to define the additional source.
   a. Select **store:///policies/templates** and **wsp-sp-1-1-was-wssecurity-username-default.xml** from the URL selection lists.
   b. Click **Attach Source**.
   c. Click **Done**.

   You should see one external attachment beside the WS-Policy button.
7. Click the **Processing** tab to define the policy parameters for the key material.

   For signing messages:

   a. Select **http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702** from the Policy Domain list.
   b. Select **SignedParts** from the Assertion Filter list.
   c. Select **Signature Idcred** from the Parameter Name list.
   d. Specify the name of the identification credential in the Parameter Value field.

   For verifying messages:

   a. Select **http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702** from the Policy Domain list.
   b. Select **SignedParts** from the Assertion Filter list.
   c. Select **Verification Valcred** from the Parameter Name list.

   d. Specify the name of the validation credential in the Parameter Value field.

8. Click **Apply**.
9. Click **Cancel**.
10. Click **Save Config**.

For further reading see *DataPower Web GUI Guide,* 2007, IBM product documentation.

3.4.2 Service level monitoring: XML variation scenario

This section briefly discusses variations on the normal SLM method of configuration. This particular approach was developed initially with the goal of allowing the SLM configuration to be updated merely by changing a simple XML file rather than requiring the user to update the domain configuration. The technique can be applied in a variety of ways, as will become clear.

The core of the technique is to arrange a set of SLM statements that correspond to particular traffic levels (for example, 1 transaction per second (TPS), 10 TPS, 100 TPS, and so on), then use a stylesheet prior to the SLM action to evaluate each incoming message and associate it with one of the SLM statements.

The easiest way to understand the technique is to look at a working example. Example 3-6 is a sample of the XML that we want to use to control the SLM.

Example 3-6 Sample of XML to use to control SLM

```
<SLAtoSLMmapping>
    <sla name="bronze" slm="1TPS">
        <wsdl name="parlayx_terminal_location_service_2_3.wsdl" slm="2TPS">
            <operation name="getLocation" slm="10TPS"/>
        </wsdl>
    </sla>
    <sla name="silver" slm="3TPS">
    </sla>
    <sla name="gold" slm="5TPS">
    </sla>
    <sla name="platinum" slm="10TPS">
    </sla>
    <sla name="test" slm="1TENTH-TPS">
    </sla>
</SLAtoSLMmapping>
```

This design maps a service level agreement, such as bronze, gold, or silver, to criteria for determining which level of service should be permitted for each message.

The XML is fairly easy to interpret. First, we know the SLA based on the AAA information. A particular user ID is entitled to one of the defined SLAs. Since this example is for a Web Service Proxy, we also know the WSDL name and the operation. Therefore, the stylesheet simply looks for an <operation> element in the XML, then for a <wsdl> element, then finally for an <sla> element based on the current message. The result, of course, is the number of TPS, which is the “slm” attribute in each of those elements. Example 3-7 is a snippet of the stylesheet.

Example 3-7 Snippet of stylesheet

```
<xsl:variable name="wsdl" select="dp:variable('var://service/wsm/wsdl')"/>
<xsl:variable name="result">
```

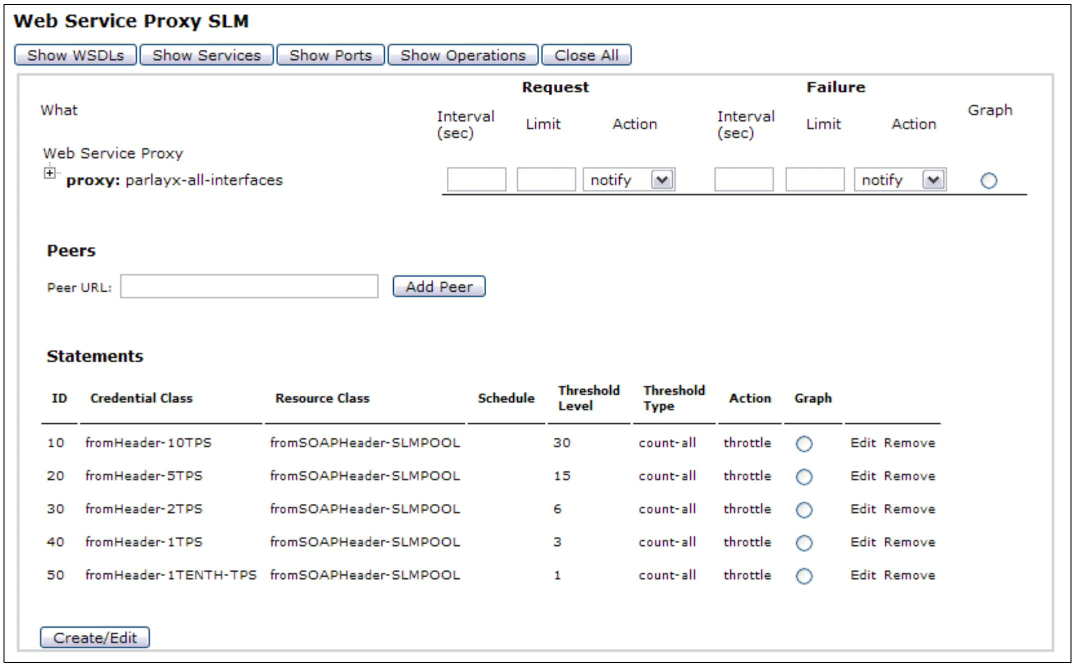
```
    <xsl:variable name="byOperation"
select="$mapdoc/SLAtoSLMmapping/sla[@name=$sla]/wsdl[@name=$wsdl]/operation[@name=$operatio
nName]"/>
    <xsl:choose>
        <xsl:when test="$byOperation">
            <root>
                <slm name="{$byOperation/@slm}"/>
                <slm-pool name="{concat($appId, '/', $wsdl, '/', $operationName)}"/>
            </root>
        </xsl:when>
        <xsl:otherwise>
            <xsl:variable name="byWsdl"
select="$mapdoc/SLAtoSLMmapping/sla[@name=$sla]/wsdl[@name=$wsdl]"/>
            <xsl:choose>
                <xsl:when test="$byWsdl">
                    <root>
                        <slm name="{$byWsdl/@slm}"/>
                        <slm-pool name="{concat($appId, '/', $wsdl)}"/>
                    </root>
                </xsl:when>
                <xsl:otherwise>
                    <xsl:variable name="bySLA"
select="$mapdoc/SLAtoSLMmapping/sla[@name=$sla]"/>
                    <xsl:choose>
                        <xsl:when test="$bySLA">
                            <root>
                                <slm name="{$bySLA/@slm}"/>
                                <slm-pool name="{$appId}"/>
                            </root>
                        </xsl:when>
                        <xsl:otherwise>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:otherwise>
            </xsl:choose>
        </xsl:otherwise>
    </xsl:choose>
</xsl:variable>
```

The pattern is clear: The result contains a bit of XML, as in Example 3-8.

Example 3-8 Result with a bit of XML

```
<root>
    <slm name="1TPS"/>
    <slm-pool name="bronze"/>
</root>
```

The <slm name=”1TPS”/> element is passed to the SLM action in this example as an HTTP header, and the <slm-pool name=”bronze”/> element is passed as a SOAP header. Both are removed later by another stylesheet. They could just as easily be stored in context variables. In any event, these values are consumed by the SLM action downstream from the stylesheet. The SLM action is configured as shown in Figure 3-13.



	Request			Failure			
What	Interval (sec)	Limit	Action	Interval (sec)	Limit	Action	Graph
Web Service Proxy ⊞ **proxy:** parlayx-all-interfaces			notify			notify	○

Peers

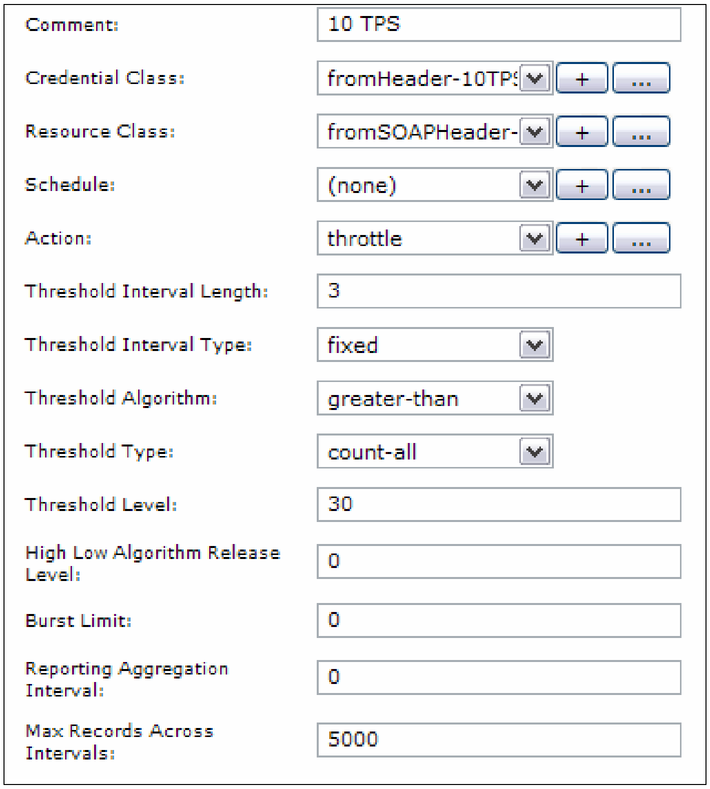
Peer URL:

Statements

ID	Credential Class	Resource Class	Schedule	Threshold Level	Threshold Type	Action	Graph	
10	fromHeader-10TPS	fromSOAPHeader-SLMPOOL		30	count-all	throttle	○	Edit Remove
20	fromHeader-5TPS	fromSOAPHeader-SLMPOOL		15	count-all	throttle	○	Edit Remove
30	fromHeader-2TPS	fromSOAPHeader-SLMPOOL		6	count-all	throttle	○	Edit Remove
40	fromHeader-1TPS	fromSOAPHeader-SLMPOOL		3	count-all	throttle	○	Edit Remove
50	fromHeader-1TENTH-TPS	fromSOAPHeader-SLMPOOL		1	count-all	throttle	○	Edit Remove

Figure 3-13 Configuration

Each statement has a credential class that picks up the SLM level (from the <slm name=”1TPS”/> element) for the message. The resource class is the same for each statement, since it simply picks up the value from the <slm-pool name=”bronze”/> element. Figure 3-14 is an example of one of these SLM statements.

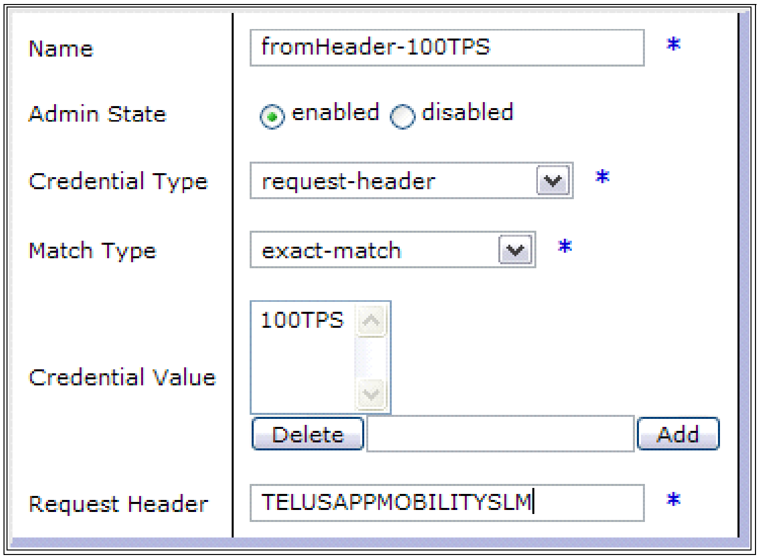


Comment:	10 TPS
Credential Class:	fromHeader-10TP!
Resource Class:	fromSOAPHeader-
Schedule:	(none)
Action:	throttle
Threshold Interval Length:	3
Threshold Interval Type:	fixed
Threshold Algorithm:	greater-than
Threshold Type:	count-all
Threshold Level:	30
High Low Algorithm Release Level:	0
Burst Limit:	0
Reporting Aggregation Interval:	0
Max Records Across Intervals:	5000

Figure 3-14 Example of SLM statement

Note that the threshold level is the TPS value times the number of seconds, which is 3 in this case. The rest of the statement is unremarkable.

The credential class has to be set up carefully so that it will match the value from the <slm name=”100TPS”/> element (Figure 3-15).



Name	fromHeader-100TPS *
Admin State	● enabled ○ disabled
Credential Type	request-header *
Match Type	exact-match *
Credential Value	100TPS
Request Header	TELUSAPPMOBILITYSLM *

Figure 3-15 Credential class setup

In this example the TPS value, 100TPS, is picked up from the HTTP header and matched against the literal value 100TPS shown in the list box. This literal value must match the value in the <slm name=”100TPS”/> in the XML.

The resource class just returns a string. In this example, it will either be something like “bronze”, “bronze/parlayx_terminal_location_service_2_3.wsdl”, or “bronze/parlayx_terminal_location_service_2_3.wsdl/getLocation”. Messages with the same string are grouped together for throttling, shaping, or logging.

Three possible variations

This technique is appropriate in several circumstances:

- The user must update the SLM policies without actually changing the configuration of objects. This may be due to user unfamiliarity with the Web GUI or the need to repeat test cycles and promotions when making minor changes to the object configuration.
- When there is a large mismatch between what SLM can conveniently do out of the box and what the model the user needs to implement. In this case you can devise a simple XML schema to control the SLM configuration and put logic in the stylesheet to evaluate each message according to the XML configuration.

The key design point is that the stylesheet evaluating each message produces two literal values. One value is used to activate a specific SLM statement for the message. The other value is used to group messages that should be throttled, shaped, or logged according to the same limits. You can pretty much do anything that you like in terms of the stylesheet and any XML configuration file as long as you end up with those two strings.

Some sophisticated actions can be taken with care. For example, you could adaptively assign TPS values to messages based on how many of each kind of message have passed through recently. Suppose that you pull the XML configuration (similar to the SLAtoSLMmapping above) from a system variable rather than from a file. Further, you record information in a system variable about each message as it passes through. Finally, you define a rule that periodically executes a stylesheet that copies and clears the historic information, evaluates it, and writes a new XML configuration into the variable mentioned earlier.

You could even drive the *scheduler* stylesheet with its own XML configuration that specified the policies for choosing which message classes to favor. Alternatively, you could base the XML configuration on information determined off box.

4

Security

The big benefit of Web services and XML is that these technologies make application integration easier and more intuitive. The counter to this is that this simplicity leads to new security exposures. Data must be protected from the public view and from tampering, control to critical services must be authorized, and malicious attacks must be detected and dealt with. WebSphere DataPower SOA Appliances offer a rich set of security features purposely designed to deal with these threats and exposures.

In this chapter we describe DataPower's security features, including:

- Protocol-based security, including SSL
- Message-based security, including digital signature generation and verification, as well as data encryption and decryption
- The Authentication/Authorization/Audit (AAA) framework for access control
- Federated Identity Management

The DataPower services addressed are:

- XML firewall (XF)
- Web services proxy (WS-Proxy)
- Web application firewall (WAF)

Additionally, we address the close cooperation between DataPower and Tivoli's security management products.

4.1 XML threat protection

Using self-describing eXtensible Markup Language (XML), intermediate network nodes are able to extract portions of the data stream and effect application-aware policies. Unfortunately, this has enabled a new opportunity for malicious attacks in the IT arena. So the emergence of XML has seen the rise of new vulnerabilities in Enterprise Information Systems. While we are just beginning to understand the repercussions of this type of attack, they are emerging because of the popularity of Web services in an SOA environment. Because of this, when exposing computer systems to XML traffic, numerous security issues must be carefully considered.

The DataPower appliance acts as an XML firewall, an application-level gateway, or a secure ESB to protect the enterprise applications against XML attacks. DataPower can handle more than just XML documents, but this chapter emphasizes XML traffic and its risks.

4.1.1 Security requirements in an IT environment

Examination of high-level requirements for security management, as shown in Figure 4-1, highlights the following aspects:

- The need to manage identity and security across a range of systems and services that are implemented in a diverse mix of new and old technologies

  A typical SOA has many points at which security policy is enforced and implemented. These policy enforcement points will be located both at the service connectivity level and within the implementations of the services themselves. Management of a policy across these various heterogeneous enforcement points requires an administrator to use a diverse set of resource-centric management interfaces and their associated security policy terminology and semantics.

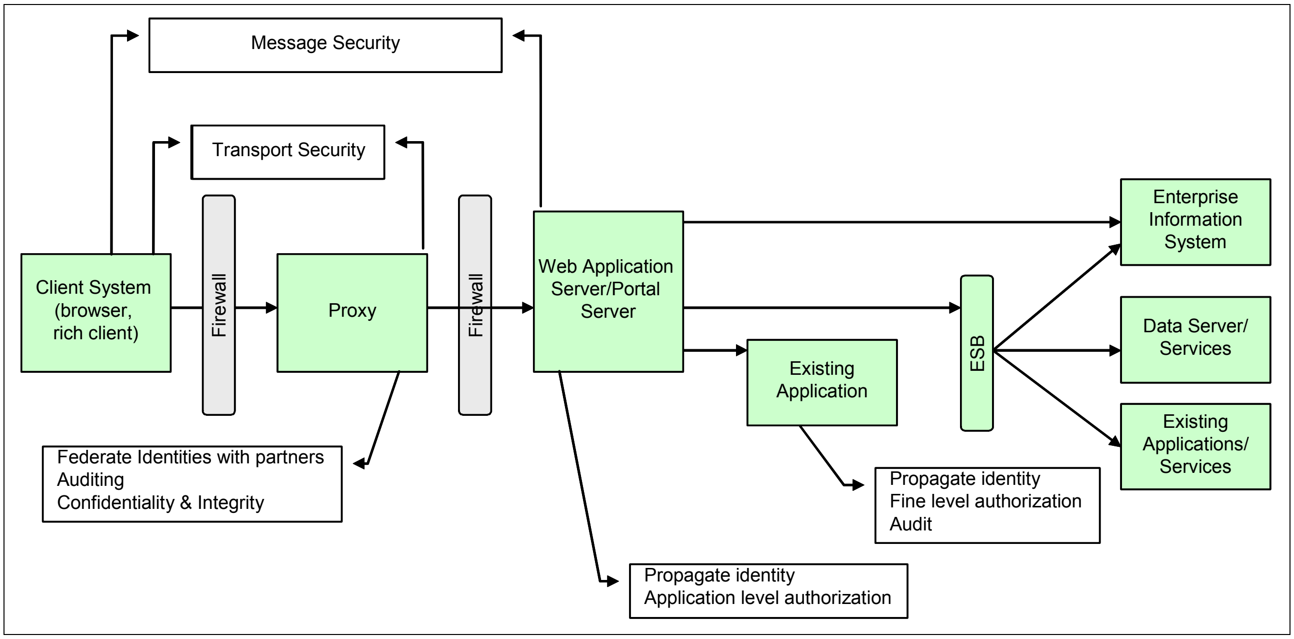

Message Security
Transport Security
Client System (browser, rich client)
Firewall
Proxy
Firewall
Web Application Server/Portal Server
Existing Application
ESB
Enterprise Information System
Data Server/ Services
Existing Applications/ Services
Federate Identities with partners
Auditing
Confidentiality & Integrity
Propagate identity
Fine level authorization
Audit
Propagate identity
Application level authorization

Figure 4-1 Security in a typical environment

For SOA to achieve its goal of business flexibility within an environment of governance and regulatory compliance, the definition and management of security policy need to be simplified. There must be consistent management terminology and semantics across the various enforcement points. This defined policy can then either be directly enforced by the enforcement points or automatically translated into something that the endpoints can understand and enforce.

- Protection of data in transit and at rest

  Protection of data from unauthorized modification and disclosure is a key requirement within IT architecture. Data needs to be protected because it is business or privacy sensitive or both. A policy must therefore be in place to ensure that data is protected in both transit and at rest, with consistent security measures applied.

  Data protection is especially important when data moves outside the organizational boundary, which can happen without the knowledge of the consumer. For example, an internal service might be replaced with an outsourced service, with data now flowing to the external organization. If the data is business or privacy sensitive, then the service provider might need to ensure that appropriate protection is in place to satisfy the policy requirements of the calling organization.

- The need for demonstrable compliance with a growing set of corporate, industry, and regulatory standards

  Auditing of transactions is required to provide the data needed for assessing compliance, which measures the performance of the IT environment relative to measures established by the business policies. This might include verifying the working system against a set of internally created policies and also against external regulatory acts.

  Complexity is increased in SOA where different applications from different sources or vendors are targeted for different levels of compliance. This is especially true when accessing services provided by an external organization, where the regulatory and compliance regime is different from the requesting organization.

  If possible, the audit data produced by the various policy enforcement points must be integrated into a single repository, or federated into a single logical view of the data. This will facilitate the production of the required audit reports, verification of compliance against policy, and investigation of security-related events.

- The need for user and service identities and propagation of these identities across the organization
- The need to seamlessly connect to other organizations on a real-time, transactional basis

4.1.2 XML threats

Unfortunately, with XML traffic, it is not guaranteed that only valid requests for valid services from genuine clients will get inside the enterprise boundary. Any client system can send a harmful message (an XML document in Web services) to a system. Enterprise-ready application servers are now susceptible to many of these attacks, leaving open a security hole and exposing the organization's applications to completely new threats.

As companies embrace XML-based Web services and service-oriented architectures, XML processing requirements are growing rapidly. The popularity of XML has invited an entire new class of attacks. For instance, XML denial-of-service (XDoS) attacks seek to inject malformed or malicious XML into middleware servers with the goal of causing the server to churn away valuable cycles processing the malicious XML. A variety of hackers are working to attack your systems and compromise them using vulnerabilities introduced by new technologies. In an SOA environment, an obvious place to attack is in the application data stream itself, the XML. For more information about XML attacks, see “XML threats” on page 210.

Potential SOA security breaches can come from anywhere, as shown in Figure 4-2. If a system can be accessed by outsiders (for example, through the Internet), an intruder could choose to send messages to a system in order to damage it or simply to consume resources. The same concerns apply to intranet or extranet access as well. Depending on the system, it is even possible that these intruders are authorized to use your system, but are trying to exploit that authorization in some inappropriate way. That is, the possibility of attacks (or other actions that can affect system response) from behind your own firewalls should not be discounted.

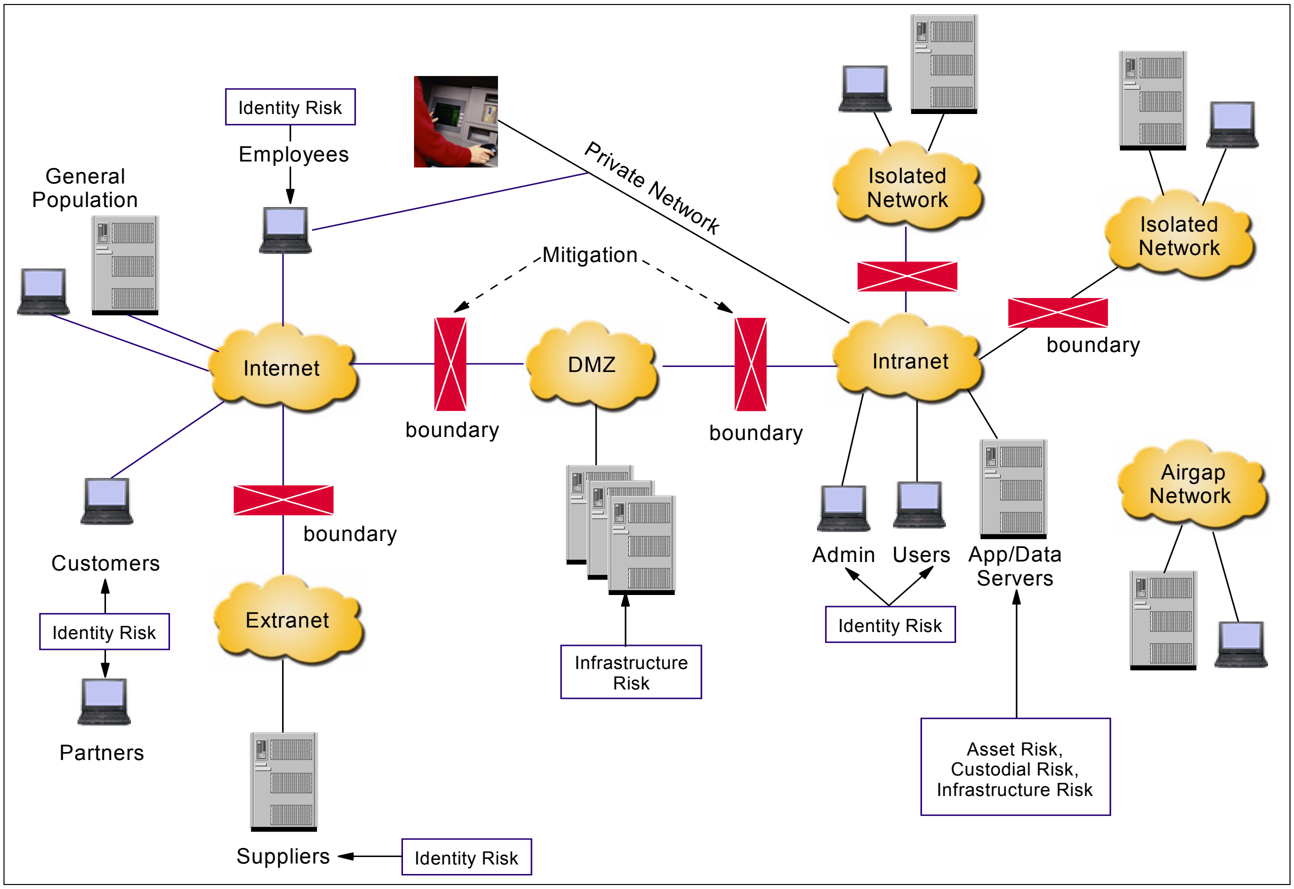


Figure 4-2 Extended Enterprise SOA and IT assets: Where are the risks?

Web security has evolved from strictly Internet protections to internal considerations, so we expect that application security will also evolve to consider both external and internal threats. The distinction between external and internal threats is largely meaningless today, given large companies with sites spanning the globe, mobile workforces, contractors, wireless networking, and more. Nonetheless, in this chapter we use the term *Internet-facing* to signify *outsider* access. Therefore, one needs to consider the traditional security models, taking into consideration services integration across domains requirements.

While the basic security issues (reliable authentication, authorization, DoS attacks, and so on) remain essentially the same, the existing solutions are not directly applicable. Systems hosting Web services, particularly public Internet-facing ones, should seriously consider the case for hardened gateway devices acting as XML firewalls to protect systems from XML threats.

What is needed is something that understands the business-level protocols and can secure them. Stated simply, we need to ensure that only valid requests for valid services from genuine clients penetrate the enterprise boundary. That is, an XML firewall is needed. That is where the DataPower appliance comes in.

These threats are not addressed either when using traditional Web service gateways that neither look for most XML threats nor limit the traffic.

This section illustrates how the above-mentioned security options can be accomplished with IBM DataPower appliances. The solution is to use DataPower as a proxy gateway. When acting in this role, the high-performance DataPower appliance can process incoming XML messages with validation turned on, while looking for XML threats. That XML-level protection enhances SOA security management.

Protection from XML threats, as depicted above, is clearly only one way in which enterprise systems need to be protected. A comprehensive SOA security strategy requires multiple levels of defense and policy enforcement. But for these types of threats, DataPower helps protect SOA implementations with fine-grained access control, achieving integration with improved security access and leveraging policy systems, thus allowing for enterprise SOA deployments and centralized security policy management.

4.1.3 Network-level security

Network-level security such as SSL/TLS, IPSec, SMIME, and so on, ensure that the message is transported between service endpoints in a secure manner. The service riles on underlying transport for security, which is IP for the Internet. in fact, the security is applied for each IP socket in the communication channel. In that way, everything being sent through will be encrypted by the sender and will need to be decrypted by the receiver, making use of digital certificates. That is, even message fields that do not need protection will be encrypted. This protection is limited to point to point, between the sender and the receiver points. XML or other data easily tunnels through the transport layer—the Internet Protocol (IP)—using standard HTTPs. HTTPs provides a binary level protection to the entire traffic flow by encrypting then decrypting each TCP packet.

Figure 4-3 displays the protection at the transport protocol between two points, but security still matters behind the firewall.

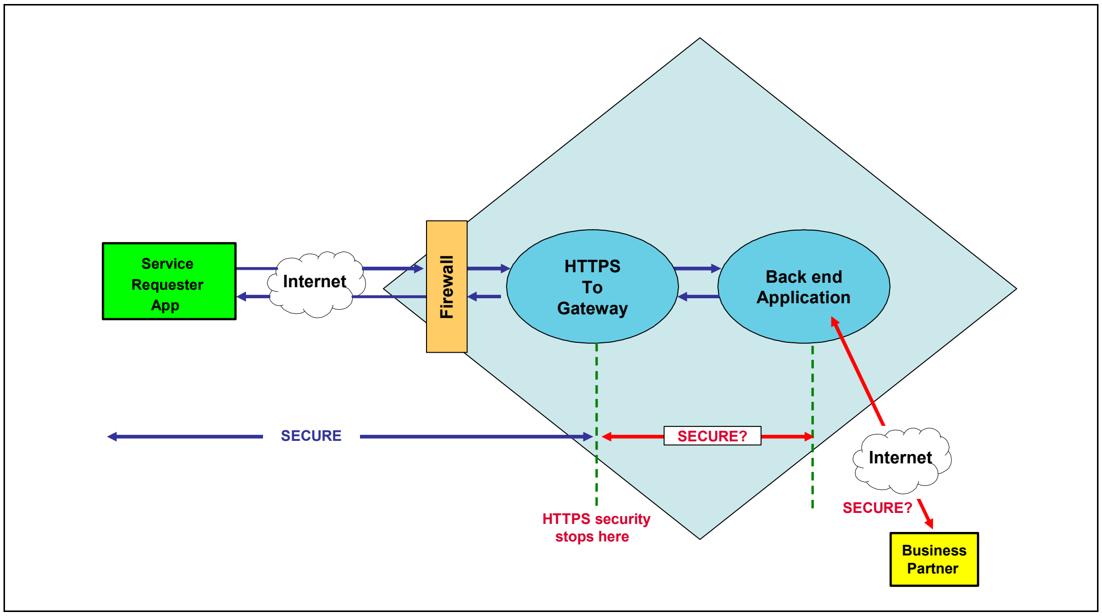


Figure 4-3 Securing the communication point to point with TLS

The dialog between the client and server points is represented in Figure 4-4.

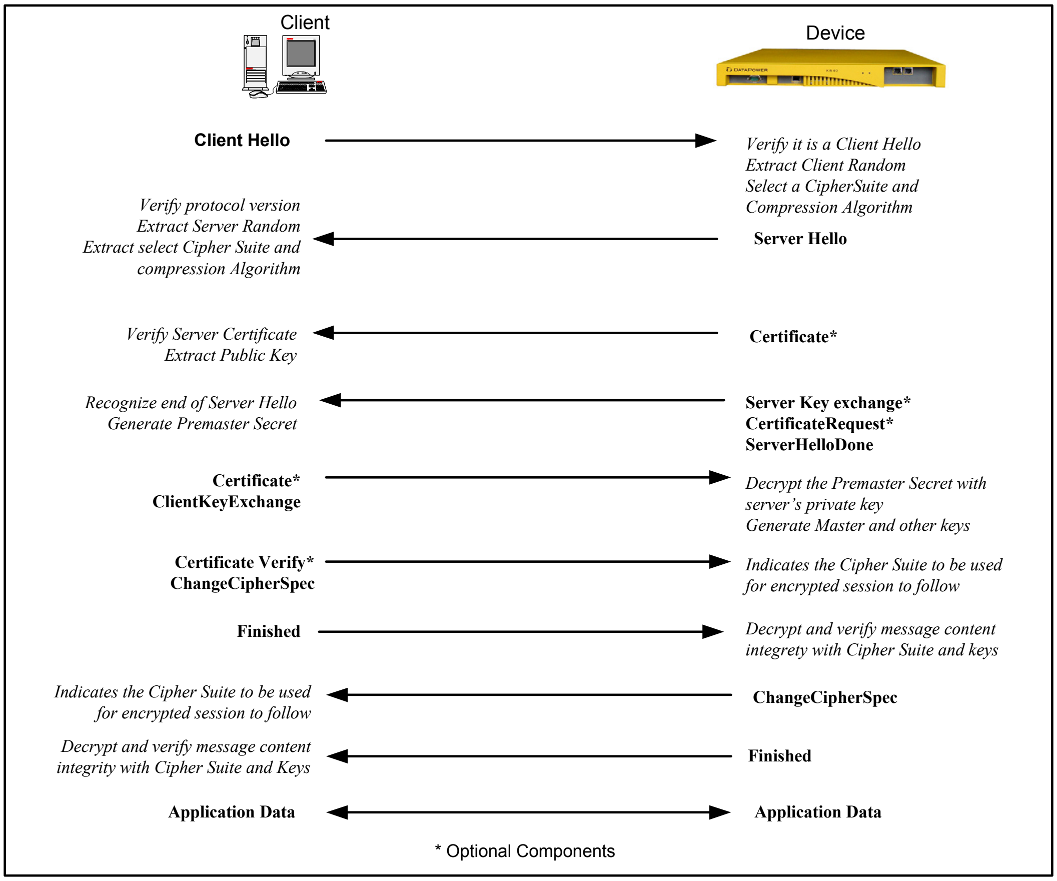

Client
Device
Client Hello
Verify it is a Client Hello
Extract Client Random
Select a CipherSuite and
Compression Algorithm
Verify protocol version
Extract Server Random
Extract select Cipher Suite and
compression Algorithm
Server Hello
Verify Server Certificate
Extract Public Key
Certificate*
Recognize end of Server Hello
Generate Premaster Secret
Server Key exchange*
CertificateRequest*
ServerHelloDone
Certificate*
ClientKeyExchange
Decrypt the Premaster Secret with
server's private key
Generate Master and other keys
Certificate Verify*
ChangeCipherSpec
Indicates the Cipher Suite to be used
for encrypted session to follow
Finished
Decrypt and verify message content
integrety with Cipher Suite and keys
Indicates the Cipher Suite to be used
for encrypted session to follow
ChangeCipherSpec
Decrypt and verify message content
integrity with Cipher Suite and Keys
Finished
Application Data
Application Data
* Optional Components

Figure 4-4 SSL dialog

SSL processing

SSL is a point-to-point security protocol supported by DataPower. Typically, this protection level is used to secure XML traffic through the Internet and inside the enterprise in certain circumstances. We can show how things are done by looking at the dialog between two points.

Client-to-device SSL

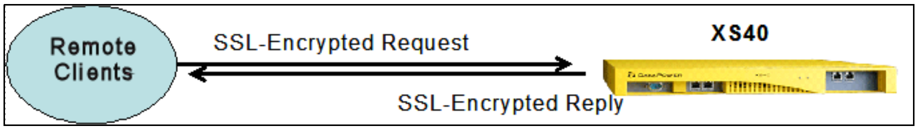


Figure 4-5 Client-to-device SSL communication

The main steps for this communication are:

1. The client sends a URL to the server beginning with https://.
2. The client wants to do SSL. The server sends the certificate to the client.
3. The client validates the certificate.
4. The expiration of the certificate is checked.

5. The identity in the certificate is checked to ensure that it is the same as that of the host that they are trying to reach.
6. The signer of certificate (could be signer chain) is checked against the trust store.
7. The certificate revocation list (CRL) is checked to see whether the certificate has been revoked.
8. The client encrypts a random secret message using the server's certificate and sends an encrypted message to the server.
9. The server uses its private key to decrypt the message and sends it back to the client.
10. The client verifies that this matches the secret message that it encrypted.
11. The client is assured that it is talking to the correct server.
12. The client and server now negotiate ciphers and begin encrypted communications over the transport.
13. In some cases we use *mutual SSL*, where both client and server verify each other.

Client-to-device SSL proxy profile

We need to configure the Crypto Identification Credentials from the DataPower console:

1. From Client-to-Device SSL Proxy Profile, choose **Reverse(Server) Crypto Profile**.
2. Select **identification credential**.

Device-to-server SSL

For device-to-server SSL:

1. The cryptographic certificate is supplied by the endpoint application server.
2. The DataPower device validates the certificate (often including certificate chain).
3. The matching private key is maintained by the endpoint application server.
4. The application server *may* request the certificate from the DataPower device and validate (mutual authentication).

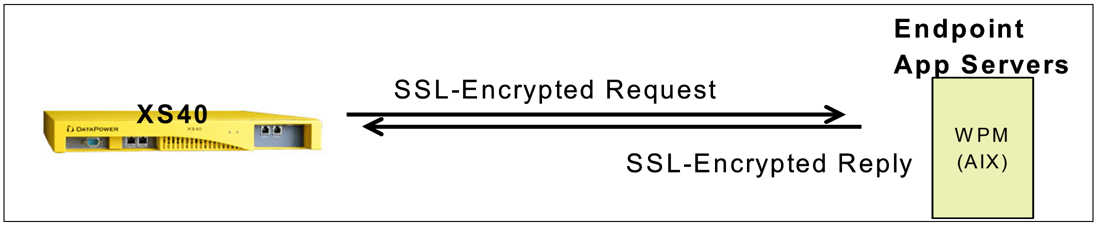


Figure 4-6 Device-to-server SSL communication

Client-to-server-via-device SSL

The main steps are:

1. The DataPower device SSL terminates the connection with the client and provides the certificate.
2. The DataPower device SSL initiates a new connection with the server.

3. The DataPower device can use a back-end server certificate if the DataPower device has copies of the back-end server keys on board.

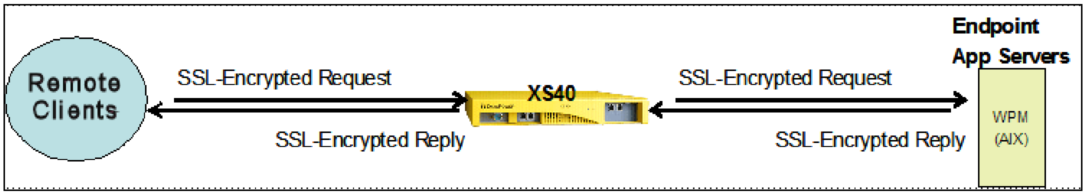


Figure 4-7 Client-to-server-via-device SSL communication

Device-to-external resource SSL

The main steps are:

1. The DataPower device SSL initiates a new connection with the external resource (config server, auth server, and so on)

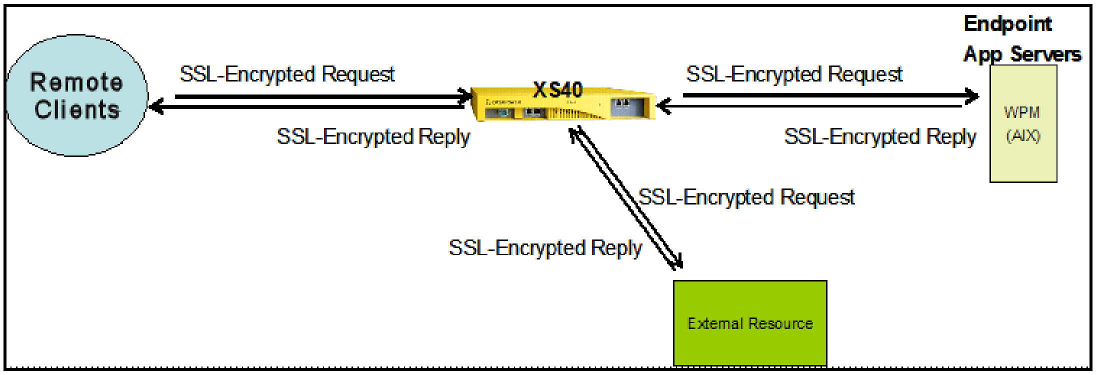


Figure 4-8 Device-to-external resource SSL communication

2. The external resource provides the certificate and *may* ask the DataPower device for the certificate.
3. The DataPower device validates the server certificate (actual certificate or CA certificate).
4. For the user agent configuration:
   a. Given a URL match, the user agent invokes the selected SSL Proxy Profile for outbound connections.
   b. The user agent invoked by the XML manager is used by the service.
5. SSL is applied:
   – SSL proxy profiles (client, server, or both) may be used by:
     - XML firewall
     - MultiProtocol Gateway (through Front Side Protocol Handler)
     - Web services proxy (through Front Side Protocol Handler)
     - User agent (launched by XML manager)

- SSL may be used through extension functions:
  i. URL-Open now takes an SSL Proxy Profile Argument.
  ii. SOAP-Call takes an SSL Proxy Profile Argument.
  iii. LDAP-* takes an SSL Proxy Profile Argument.
- Extension functions used by custom stylesheets

MQ and SSL

The steps are:

1. The remote MQ queue manager must first be configured to use SSL.
2. A client key is exported during that configuration process and uploaded to the DataPower device.
3. Both the key file and the password file must reside on the DataPower device.
4. The MQ queue manager object on the device must be configured to use the uploaded files.
5. The Cipher Suite Specification must agree with the remote QM.

4.1.4 Message level security within XML context

Messages may have to go through a few hops before they reach their final destination. It may not be reasonable to assume that every hop is secured using SSL. Intermediate nodes may need to see some part of the message (extracting identity/credentials, for instance) to determine where to route the request. The service provider needs to authenticate every intermediate point. One may require loose coupling of the security model from the underlying transport. Protection at the message level provides the capability to apply security only where we have to. Unlike the protection on a binary level, this protection takes into account every hop in the communication chain, as shown in Figure 4-9.

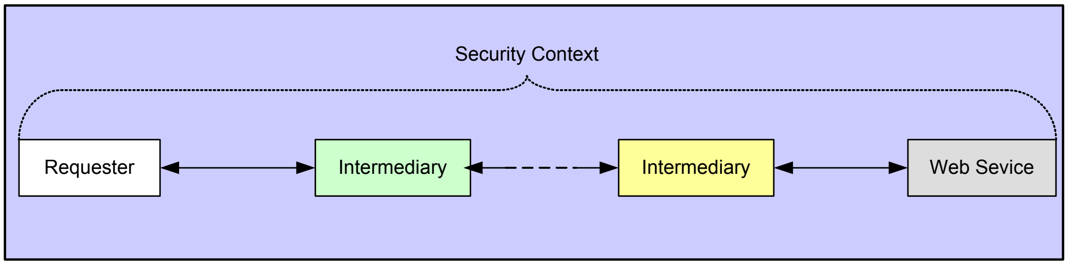


Figure 4-9 Security context taken in account on message level protection basis

Figure 4-10 shows an example of this, where a Web service flow is sent encrypted using Web Services Security (WS-Security or WSS) standards from a client through the Internet. The intermediary DataPower appliance decrypts and authenticates the message before forwarding it in the clear over the last mile hop to the eventual service provider. Note that the appliance could just have easily secured the last mile encrypted under a transport-level mechanism such as SSL if the solution dictates.

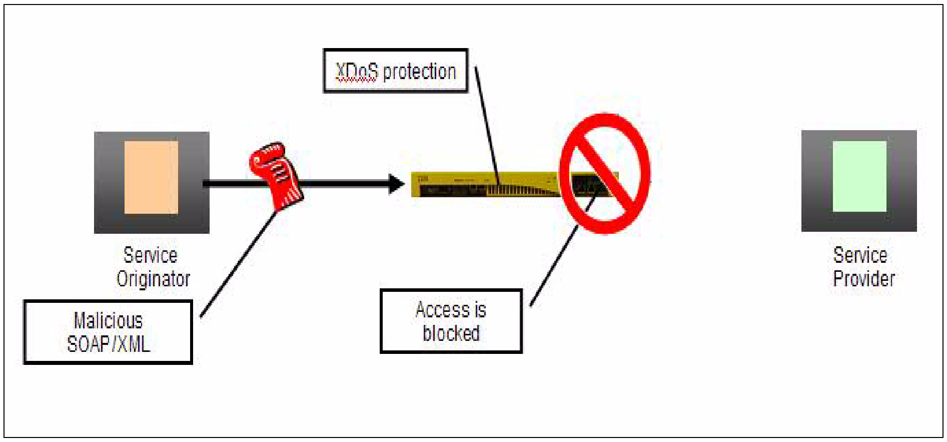


Figure 4-10 Application-layer thread protection at message level with DataPower XS40

With Web services, message level standards have emerged, defining how to secure SOAP messages. The main concepts handled are:

- Data integrity

  Digitally sign the SOAP XML document, providing integrity, authenticity, and signer authentication, based on the W3C recommendation specification XML Signature.

- Data confidentiality

  Process for encrypting data and representing the result in XML, based on W3C recommendation XML Encryption.

- Username token profile

  Describes how a Web service consumer can supply a UsernameToken as a means of identification by user name and password.

- X509 Certificate Token Profile

  X.509 authentication framework within WS-Security.

Elements of a solution

To address the requirements of data protection, an ideal solution needs to include both business and technical aspects.

From a business view, we need data protection, privacy, and disclosure control for managing the policies around how data needs to be protected in transit and at rest. This also includes interpretation of any privacy-specific policies.

From a technical point of view, we need confidentiality and integrity services: A standards-based service to handle the data protection issues end-to-end. For example, Secure Sockets Layer (SSL) is the most common example of a secure transport level scheme. With SSL, the entire data stream is protected at a protocol level below the application layer. SSL is typically the secured protocol used to protect traffic between a Web browser and a Web server.

For these requirements, the advantage of using WS-Security instead of SSL is that it can provide end-to-end message level security. This means that the messages are protected even if the message traverses through multiple services, or intermediaries. Additionally, WS-Security is independent of the transport layer protocol. It can be used for any SOAP binding, not just for SOAP over HTTP. The solution can be delivered as shown in Figure 4-11.

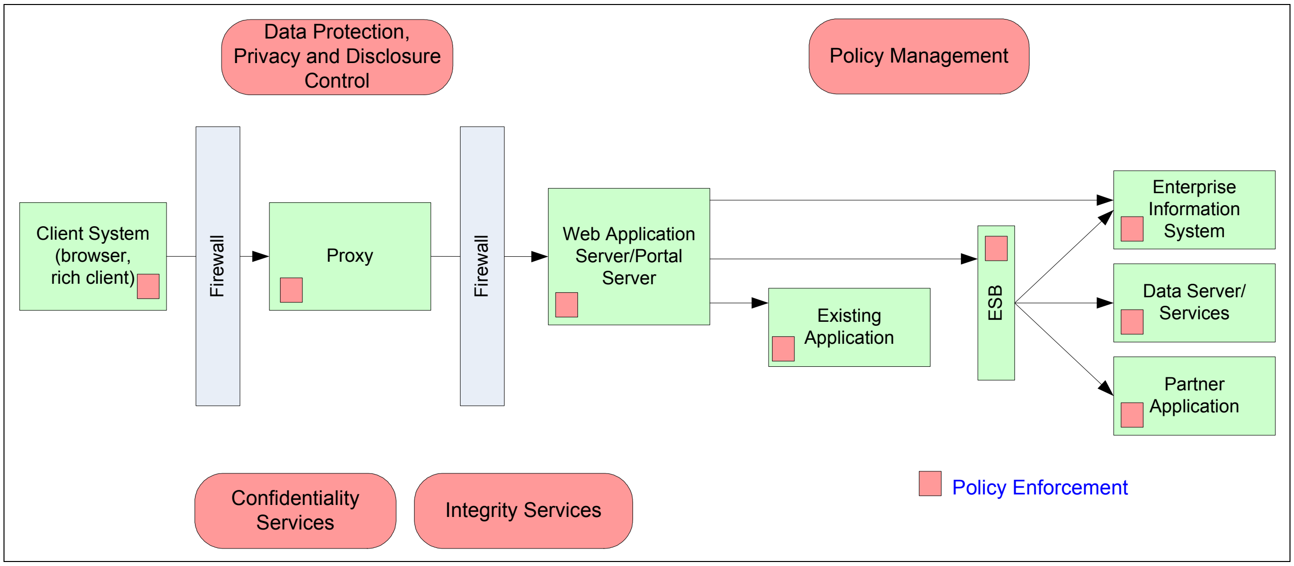


Figure 4-11 Addressing the requirements around data protection

To address these needs, the DataPower implementation of the WS-Security specification provides such open standards based message level security and access control functionality. It can filter, validate, encrypt, and sign messages, helping to provide more secure enablement of high-value applications.

DataPower can selectively share information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters.

4.2 IBM SOA security offering

HTTPs provides a binary level protection to the entire XML traffic flow between two points by encrypting and then decrypting each IP socket. But we need much more than that for providing reliable end-to-end XML traffic protection against new types of threats.

DataPower capabilities

The DataPower service allows the securing and encrypting of the communication channels between a service provider and a requesting client using Secure Sockets Layer by using the secure HTTP protocol named HTTPS.

The DataPower portfolio currently includes two devices that can address security issues. The model XS40 is an XML firewall and a Web services gateway wirespeed appliance built mainly for security purposes. It can perform industrial-strength security as well as more advanced functions, such as operations on message content. The model XI50 is a complete super-set of the XS40. But in addition, the XI50 can enable an ESB because of its added interoperability with connectivity services.

As stated in previous chapters, DataPower is a SOA appliance that improves security with hardware and brings benefits such as:

- Sealed network-resident device
- Optimized hardware, firmware, embedded OS
- Single signed/encrypted firmware image
- Cannot install arbitrary software
- High assurance, *default off* locked-down configuration
- Security vulnerabilities minimized (few third-party components)
- Hardware storage of encryption keys, locked audit log
- No drives/USB ports, tamper-proof case
- FIPS level 3 HSM (option)
- Under evaluation by Common Criteria EAL4
- Large financial and government customers usage

DataPower provides front-end processing for Web services requests. It can work in close cooperation with Tivoli's security management products. It is a B2B XML Security Policy Enforcement system that includes XML acceleration.

Additionally, DataPower appliances provide a sophisticated set of security capabilities, some of which are noted below.

XML Web services access control

DataPower appliances support a variety of access control mechanisms, including WS-Security, WS-Trust, X.509, SAML, SSL, LDAP, XACML, RADIUS, and simple client/URL maps. They can control access rights by rejecting unsigned messages and verifying signatures within SAML assertions.

Federated Identity Management

The appliances directly support IBM Tivoli Federated Identity Manager (TFIM) for capabilities such as mapping identities for downstream access, and IBM Tivoli Access Manager (TAM) for retrieve authorization, authentication, and service level authorization.

Message and data protection

DataPower appliances can selectively share information through encryption/decryption and signing/verification of entire messages or of individual XML fields. These granular and conditional security policies can be based on nearly any variable, including content, IP address, host name, or other user-defined filters. Whereas enforcing this level of security in hardware would cause massive performance overhead, DataPower's hardware performance allows the required cryptographic functions to perform with very little overhead.

Threat protection

DataPower includes XML threat protection as well as protection against non-XML threats such as Cross Site Scripting and SQL Injection.

4.3 WS-Security

WS-Security is an OASIS Standard and is the industry-adopted specification for securing SOA message exchange. This section tends to clarify how WS-Security is supported on the DataPower device and outlines the huge value engendered.

The WS-Security specification describes enhancements to SOAP messaging to provide message integrity and confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

This specification also provides a general-purpose mechanism for associating security tokens with message content. WS-Security allows for the use of multiple security token types, which provides great extensibility. For example, a client might provide one format for proof of identity and provide another format for proof that they have a particular business certification.

Additionally, this specification describes how to encode binary security tokens, a framework for XML-based tokens, and how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the tokens that are included with a message.

4.3.1 Goals and requirements

The goal of this specification is to enable applications to conduct secure SOAP message exchanges, such as those coming or going outside the enterprise boundaries. See Figure 4-12.

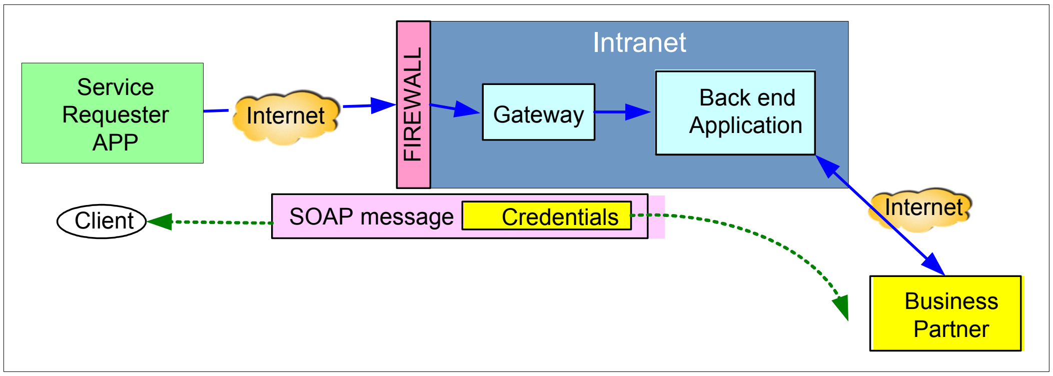


Figure 4-12 Soap message is secured on each hop

This specification provides a flexible set of mechanisms that can be used to construct a range of security protocols. The specification intentionally does not describe explicit fixed security protocols.

As with every security protocol, significant efforts must be applied to ensure that security protocols constructed using this specification are not vulnerable to any one of a wide range of attacks. The examples in this specification are meant to illustrate the syntax of these mechanisms and are not intended as examples of combining these mechanisms in secure ways.

The focus of this specification is to describe a single-message security language that provides for message security that may assume an established session, security context, or policy agreement.

The Web services security language must support a wide variety of security models. The following list identifies the key driving requirements for this specification:

- Multiple security token formats
- Multiple trust domains
- Multiple signature formats
- Multiple encryption technologies
- End-to-end message content security and not just transport-level security

4.3.2 WS-Security and DataPower

The DataPower appliance applies more functionality of WS-Security than quite possibly any other hardware or software solution on the market today. Therefore, it offers the best protection for part or all of a Web service message. The DataPower device provides application-level security between Web services with a set of mechanisms for securing SOAP message exchanges. DataPower:

- Associates security-related claims with a message
- Provides end-to-end message security
- Is totally standards complaint and does not introduce additional security protocols
- Implements the most popular security technologies, providing many options to enforce WS-Security

4.3.3 Encryption/decryption capabilities

Typically, when sensitive data must be sent over the wire, one has two chances to secure the communication. DataPower provides one option that uses Secure Sockets Layer, which secures the entire communication. However, HTTP-based security simply is not enough. HTTP and its security mechanisms only address point-to-point security. When messages might cross multiple hops, more complex solutions provided by DataPower are needed that maintain a secure context over a multi-point message path. There are two possibilities that can help solve this:

- WS-Security encryption based on the W3C recommendation for XML Encryption. The WS-Security specification defines the usage of XML Signature and XML Encryption:
  - Message integrity is provided by XML Signature in conjunction with security tokens to ensure that modifications to messages are detected.
  - Message confidentiality leverages XML Encryption in conjunction with security tokens to keep portions of a SOAP message confidential.
- XML standard encryption

With both options, DataPower can provide message-level security, incorporating security features in the SOAP message and working in the application layer, thus ensuring end-to-end security. While WS-Security encryption is the Web services security standard for encrypting SOAP messages, XML Encryption is the standard for encrypting an XML message, or even just part of it.

DataPower follows a two-step approach for encrypting data for both XML standard encryption and WS-Encryption:

1. Generate a shared secret key using the public key contained in the trusted party's certificate (asymmetric encryption).
2. Use the key that was previously generated to encrypt the sensitive data within the SOAP message (symmetric encryption).

A best practice is to have either DataPower or the other party use asymmetric encryption to generate and protect the key to be shared, while using symmetric encryption to perform encryption of the data, thus speeding up the performance. When receiving the encrypted message, the trusted party uses its private key to decrypt the session key and then uses it to decrypt the message. This mechanism ensures that only the trusted party can perform the decryption of the message, being the only one owning the correct private key.

Action types

The action types are:

- Encrypt action: An encrypt action performs full or field-level document encryption. Implementation of an encrypt action requires certain parameters that are supplied during service configuration. The DataPower interface provides a procedure to add an encrypt action to document processing rule.
- Decrypt actions: A decrypt action performs full or field-level document decryption. Implementation of a decrypt action requires certain parameters that are supplied during multiprotocol gateway configuration, a DataPower service. A DataPower interface provides a procedure to add a decrypt action to a document processing rule. The Decrypt Key drop-down list is employed to select a cryptographic key to use for decrypting the content.

4.3.4 Digital signature

A digital signature is a very important feature of Web services security that ensures message integrity. A digital signature uses cryptographic means to assert the integrity of data.

Data protection management identifies the resources needing protection and the controls required on those resources. For example, a Public Key Infrastructure (PKI) can be used to authenticate the users accessing a portal from an external network. DataPower provides a cryptographic key stored in tamper-resistant hardware. We can also leverage DataPower to store its own certificates within its cryptographic key store, as well as partner certificates. The keys from these certificates are then used to digitally sign and encrypt the security tokens and some content of the messages. DataPower can provide the file system hosting these key stores so as to be protected from external access, and this file system is encrypted and protects from any intrusion.

Digital signature schemes normally give two algorithms, one for signing, which involves the user's secret or private key, and one for verifying signatures, which involves the user's public key.

Action types

The action types are:

- Sign action: A sign action digitally signs documents. In our case it mainly signs the incoming SOAP message before encrypting the message and sending it back to the client.
- Verify action: The decrypted SOAP message contains the signer certificate that can be verified to make sure that this is an acceptable message. Implementation of the verify action requires certain parameters that are supplied during the DataPower service configuration.

X509 Certificate Token Profile

The X509 Certificate Token Profile specification describes the use of the X.509 authentication framework within WS-Security. The XML Signature specification (XMLSig) calls out a general XML syntax for signing data with flexibility and many choices. This section details constraints on these facilities so that SAML processors do not have to deal with the full generality of XML Signature processing. This usage makes specific use of the xs:ID-typed attributes present on the root elements to which signatures can apply, specifically the ID attribute on <Assertion> and the various request and response elements.

Note: This profile applies only to the use of the <ds:Signature> elements found directly within SAML assertions, requests, and responses. Other profiles in which signatures appear elsewhere but apply to SAML content are free to define other approaches.

4.3.5 Other WS-* specifications supported

Additional specifications, supported by DataPower appliance and illustrated in Figure 4-13, address further security aspects:

- WS-Policy: Policy framework
- WS-Trust: Defines security token exchange mechanism (integrate to TIFM)
- WS-SecureConversation: Focuses on defining a security context

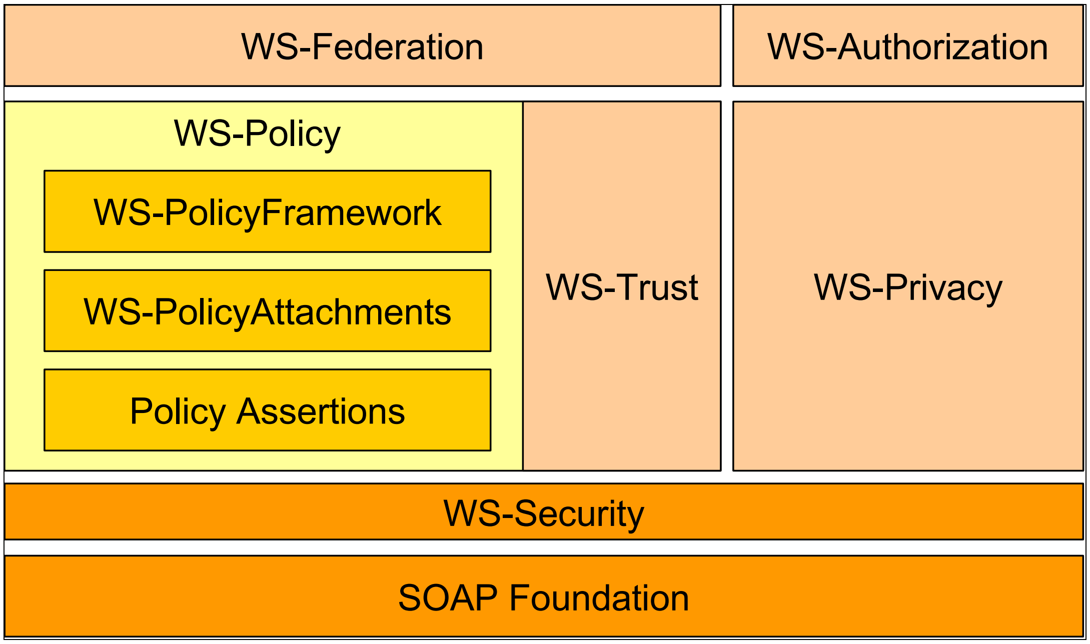


Figure 4-13 Web services security specifications supported by DataPower appliance

These specifications provide the capabilities discussed in the following sections.

WS-Policy

WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web services-based system. WS-Policy defines a framework and a model for the expression of these properties as policies. Policy expressions allow for both simple declarative assertions as well as more sophisticated conditional assertions.

WS-Policy defines a policy to be a collection of one or more policy assertions. Some assertions specify traditional requirements and capabilities that will ultimately manifest on the wire (for example, authentication scheme and transport protocol selection). Some assertions specify requirements and capabilities that have no wire manifestation yet are critical to proper service selection and usage (for example, privacy policy and QoS characteristics).

WS-Policy provides a single policy grammar to allow both kinds of assertions to be reasoned about in a consistent manner. WS-Policy stops short of specifying how policies are discovered or attached to a Web service. Other specifications are free to define technology-specific mechanisms for associating policy with various entities and resources.

Subsequent specifications provide profiles on WS-Policy usage within other common Web services technologies.

WS-Policy is also covered in 3.2.2, “Web Service Policy Framework (WS-Policy Framework)” on page 30.

Note: The specification that makes up WS-Policy is available for download from IBM developerWorks at:

http://www.ibm.com/developerworks/library/specification/ws-polfram/

WS-Trust

WS-Security defines the basic mechanisms for providing secure messaging. WS-Trust specification uses these base mechanisms and defines additional primitives and extensions for security token exchange to enable the issuance and dissemination of credentials within different trust domains.

The main goal of WS-Trust is to enable applications to construct trusted (SOAP) message exchanges. This trust is represented through the exchange and brokering of security tokens. This specification provides a protocol agnostic way to issue, renew, and validate these security tokens, and supports a range of security protocols.

In order to secure a communication between two parties, the two parties must exchange security credentials (either directly or indirectly). However, each party needs to determine whether it can *trust* the asserted credentials of the other party. Thus, WS-Trust adds extensions to WS-Security, which provides:

- Methods for issuing, renewing, and validating security tokens
- Ways to establish assess the presence of and broker trust relationships

Using these extensions, applications can engage in secure communication designed to work with the general Web services framework, including WSDL service descriptions, UDDI businessServices and bindingTemplates, and SOAP or SOAP2 messages.

For a valid WS-Trust SecurityContextToken (SCT) request, a DataPower AAA policy can generate the appropriate security token response. This processing works as an on-box WS-Trust Secure Token Service (STS) that is backed by WS-SecureConversation.

A WS-Trust token requires a symmetric shared secret key to initialize the security context. This processing can handle issue, renew, validate, and cancel operations against a request security token or request security token collection.

Note: The specification that makes up WS-Trust is available for download from IBM developerWorks at:

http://www.ibm.com/developerworks/webservices/library/specification/wstrust/

WS-SecureConversation

A requester can be authenticated by reference to an established WS-SecureConversation security context that must already exist. WS-SecureConversation credentials are taken from the WS-SecureConversation context token.

After obtaining a set of credentials that attest to the identity of the client, a DataPower AAA policy can optionally map these credentials to a more useful format by performing custom processing on the received credentials and generating a requested WS-Trust token. This token can then serve as a WS-SecureConversation token. The authentication credentials can then be mapped via any following WS-SecureConversation exchange.

ID propagation across DataPower ESB via WS-Trust

The identity of the user is passed on from the integrated applications through all the hops with the request. Passing the user identity enables the front points to apply the authentication/authorization security policies.

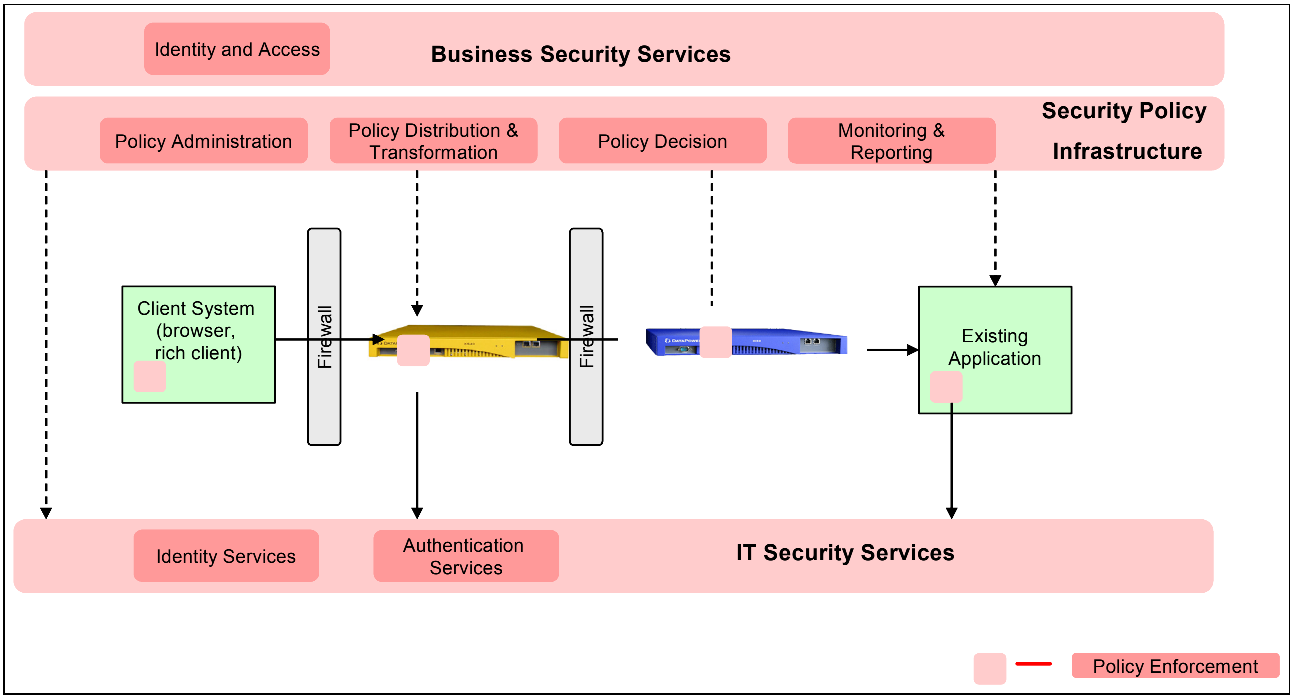


Figure 4-14 Identity propagation

This is a very important aspect, because as part of the request flow from service consumer to provider through the DataPower ESB, the user context needs to be maintained and the security of the identity information ensured. For example, consider a use case where a request is coming from an internal service consumer using DataPower to invoke a service provider, as shown in Figure 4-14. The request arrives carrying a username token that carries identity information for the user USER1. DataPower calls the Security Token Services provided by Tivoli Federated Identity Manager to exchange the security token format to one supported by the service provider. For example, the username token is exchanged for a Security Assertion Markup Language (SAML) token. In addition to changing token type, the Federated Identity Manager STS can map the incoming identity information to an identity suitable for the service provider, for example, the USER2 identity is mapped to the service provider identity of USER1. DataPower has built-in capability for WS-Trust to invoke the Federated Identity Manager STS.

4.3.6 Exceptional added value

The DataPower Security Gateway appliance has some powerful intrinsic abilities relevant for XML security processing. In XML threats protection, the DataPower Security Gateway appliance offers some exceptional advantages by comparison to typical solutions. Two are described bellow.

WS-Security acceleration

IBM performance labs have demonstrated a significant performance gain when using DataPower to handle some or all WS-Security processing in a message flow. This is the result of the hardware optimization used in both cryptographic and XML processing.

For clients who wish to use WS-Security and have performance concerns, DataPower provides an ideal solution. The DataPower appliance can consume Web services using WS-Security at network speeds and then pass the messages on to the service provider without all of the WS-Security overhead. The same is true for the responses (when request/response is in use) where Web services responses can have WS-Security information added. Figure 4-15 shows this.

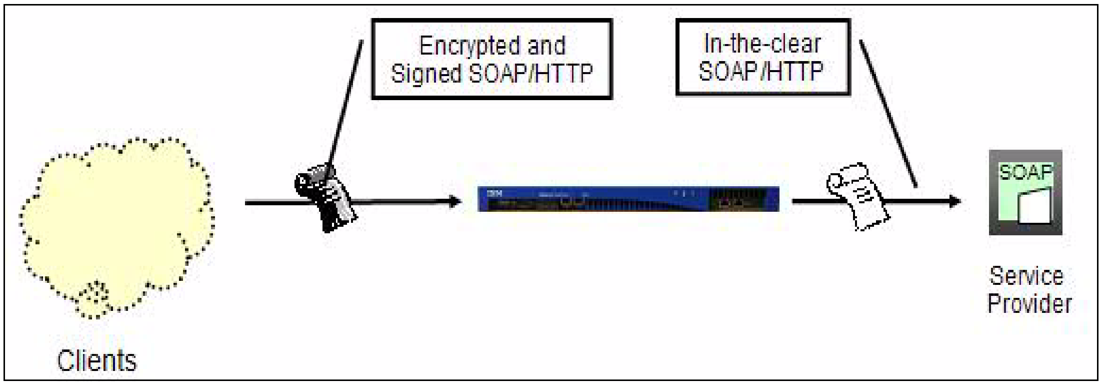


Figure 4-15 Functional acceleration with DataPower XI50/XS40

This proxying approach is very useful when identity propagation is used to bridge between multiple security contexts. For example, if the back-end Web service is hosted on WAS, DataPower can be used to extract the identity from a client credential, such as a SAML assertion, and presented to the WAS Web Service as an easily understandable LTPA token.

There are many other examples of how DataPower can be used to extract, authenticate, and propagate identities. One must keep in mind that defining the correct security solution for any environment is an optimization exercise between protection level, performance, and maintainability. In complex distributed environments, expect the optimal security solution to be a mix of network-level, message-level, and data-level protection.

This leads us to our next recommendation. For an application server based system using WS-Security where performance is a concern, consider using DataPower to handle this processing while establishing trust to the back-end service (for example, mutually authenticated SSL between DataPower and the application server).

WS-Security flexibility

When using DataPower to provide WS-Security as a proxy, we gain flexibility along three key dimensions. First, DataPower today (and in the near future) simply supports more of the WS-Security standards than our typical application server-based products. Secondly, DataPower fundamentally includes an XSL processing engine, so it is quite feasible to process Web services messages that are not quite in a format that an application server can understand. DataPower can alter those messages or more flexibly interpret them as needed. Thirdly, by offloading the WS-Security processing to DataPower, we eliminate the need to configure and reconfigure the back-end server (any application server) to understand different WS-Security messages. It is all transparent to the back end.

With regard to the first point, DataPower and other IBM products (such as WAS) will continue to be enhanced to support newer portions of the WS-Security standards, but at least for the near term, DataPower has features that are not found in other IBM products.

This leads us to these recommendations:

- For clients who wish to use WS-Security but find that their application server is unable to support the standard of interest, they should evaluate DataPower to see whether it does support this standard or can be altered to support it.
- Clients expecting to frequently change the WS-Security formats expected or those with problematic messages (for example, nonstandard in some minor way) should evaluate DataPower as a solution to provide the needed flexibility.

Closely related to the concept of flexibility is the concept of credential transformation. Since Web services requests often transition security domains, the credentials that arrive inbound to a boundary server often require transformation before being understood by the recipient or prior to sending on within the network. This transformation can occur along two dimensions: the technology and the naming. By technology we mean changing a credential from one type to another. For example, the sender might use digital signatures to provide identity information, while the internal systems expect username tokens.

The second transformation is simpler but no less important. The name that represents a subject may change. For example, your *identity* to IBM might be your IBM serial number, but your identity to your bank could be your bank account number or social security number. Identity transformation addresses both issues. Traditionally, with IBM there have been two ways of approaching this problem:

- Custom developed ad hoc code.
- Leverage a product like Tivoli Federated Identity Manager (TFIM).

Now DataPower provides a more robust means of providing credential transformation with Web services processing. DataPower can perform a fairly large class of credential transformations (both technology and naming) using built-in function as well as custom XSLT transforms. DataPower can also call out to products like TFIM to perform that translation. The obvious question is when to apply which technology. The answer is fairly straightforward.

> **Recommendation:** If DataPower is already in use for Web services security, when credential transformation is required, the native DataPower function should be used when sufficient. However, DataPower should be configured to contact identity-oriented products such as TFIM, particularly those requiring complex logic. TFIM should also be used when transformations occur in multiple places (for example, in DataPower and WAS) so that common function can be leveraged across the enterprise.

More information

Because Web services security is a quickly evolving field, it is essential for developers and designers to regularly check for recent updates.

4.3.7 Resources for further reading

For more information see:

- Web Service Security: SOAP Message Security 1.1 (WX-Security 2004) Version date: February 1, 2006

  http://docs.oasis-open.org/wss/v1.1

- Make SOA real with IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower SOA Appliances, Part 2: Use WebSphere DataPower SOA Appliances extension functions for certificate-based XML standard encryption

  http://www.ibm.com/developerworks/webservices/library/ws-soa-real2/

- *Understanding SOA Security Design and Implementation*, SG24-7310

  http://www.redbooks.ibm.com/abstracts/sg247310.html?Open

- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364

  http://www.redbooks.ibm.com/abstracts/REDP4364.html?Open

- The XML Signature Workgroup home page can be found at:

  http://www.w3.org/Signature/

- The XML Encryption Workgroup home page can be found at:

  http://www.w3.org/Encryption/

- WS-Security specification 1.0

  http://www.ibm.com/developerworks/library/ws-secure/

- White paper of the Web services security roadmap

  http://www.ibm.com/developerworks/webservices/library/ws-secmap/

- OASIS WS-Security 1.0 and token profiles

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

- Understanding SOA Security Design and Implementation

  http://www.redbooks.ibm.com/abstracts/sg247310.html?Open

4.4 Authentication, authorization, and audit (AAA)

This section describes the authentication, authorization, and audit (AAA) capabilities provided by DataPower XS40 and XI50 devices. We show how AAA is implemented by the DataPower appliance.

AAA framework

The acronym AAA stands for authentication, authorization, and audit. The framework set up by these concepts constitutes the cornerstone of access enterprise assets control by all users. Access control is about figuring out whether the identity claimed by an incoming message is indeed valid (authentication), and whether that identity is allowed to access the resource being requested (authorization).

DataPower appliances provide different means for implementing AAA. For more detailed information consult the DataPower XS40 or XI50 WebGUI documentation. Figure 4-16 illustrates the AAA framework.

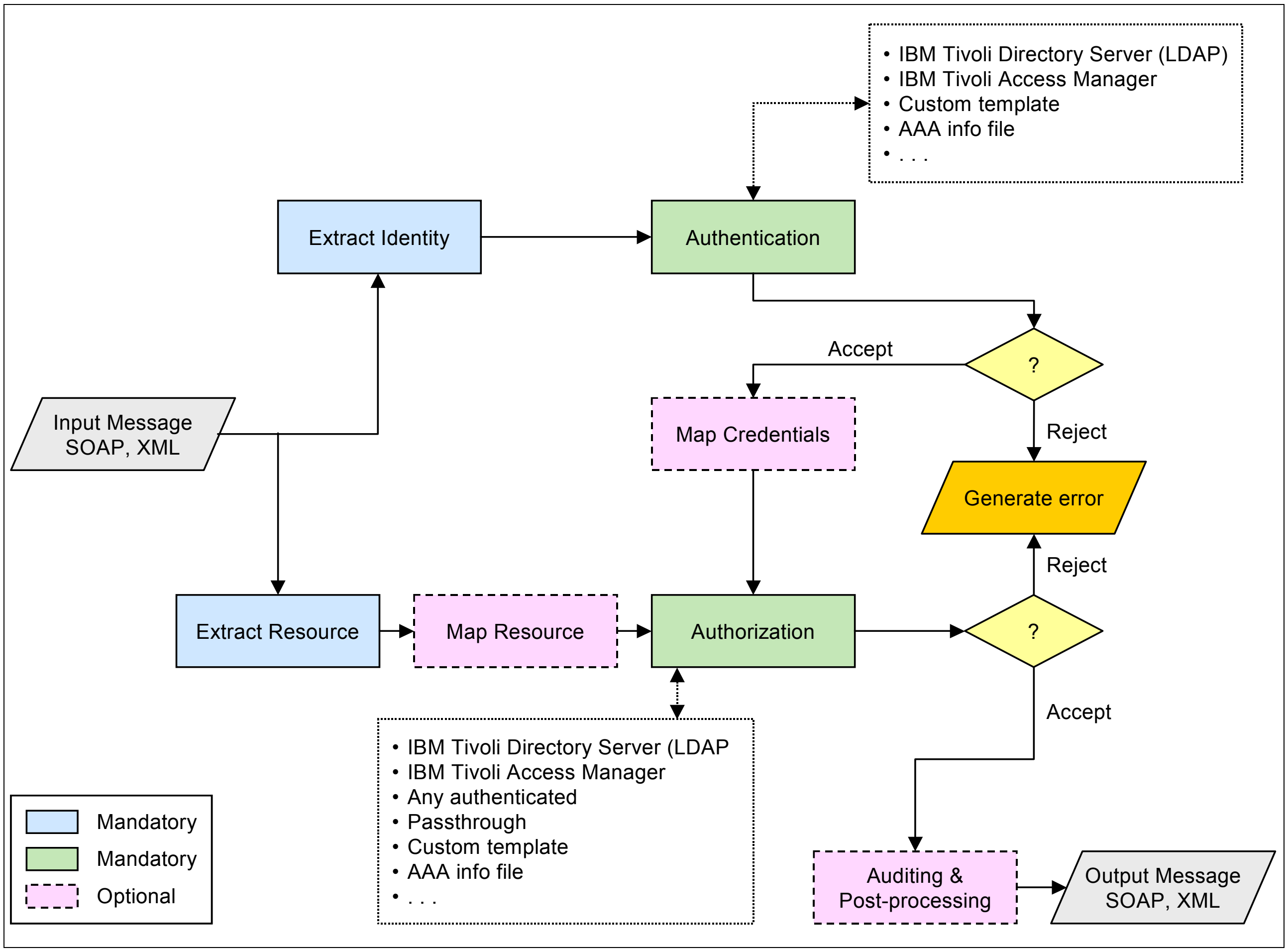


Figure 4-16 AAA policy processing

This AAA framework makes a clear separation between authentication, authorization, and audit services in a loosely coupled way. DataPower appliances comprise these services loosely coupled, described in following paragraphs.

For example, as a SOAP/XML message comes in, one has first to extract the identity claim, such as the user name and password, from the HTTP basic authentication header, and the resource, such as the Web services URL endpoint being accessed. Then one authenticates the extracted identity with either an on-board or off-board identity server, such as an LDAP directory. The result is a valid credential for the claimed identity. Both the credential and the resource can be optionally mapped using rewrite rules, then submitted during the authorization step to a policy server.

In most security systems, authorization checking is only performed if authentication is successful. If authorized, one may do some post-processing for audit and accounting purposes, then pass the message on to the back-end application server. Otherwise, the message is rejected.

So an AAA policy offers the opportunity to use custom methods for most of the seven phases of AAA processing. These phases are:

1. Extract identity (EI).

   Establish an asserted identity from the contents of the message or protocol envelope.

2. Extract resource (ER).

   Establish a requested resource (commonly a URI).

3. Authenticate (AU).

   Authenticate the asserted identity. This is approved or disapproved by the authenticating agent.

4. Map credentials (MC).

   Transform or replace the extracted identity with another value, which may or may not be based in whole or in part on the extracted identity. This may also be none, which leaves the extracted identity intact.

5. Map resource (MR).

   Transform or replace the extracted resource with another value, which may or may not be based in whole or in part on the extracted resource. This may also be none, which leaves the extracted resource intact.

6. Authorize (AZ).

   Approve or disapprove the use of the mapped resource by the an agent identified by the mapped credentials.

7. Post processing (PP).

   Perform additional transformations or processing on the results of the previous steps.

To handle the identity checking and related authorization, we need to represent and communicate it within the SOA infrastructure. Below are the main steps in processing AA.

Identity representation

The steps are:

1. Use a single identity representation mechanism and security policy across the entire architectural domain (whether that be project wide or enterprise wide).
2. Map a domain-wide identity down to the implementation identity (in the implementing application).

In practical terms, a mixture of the two approaches is quite likely the only option, due to heritage application restrictions.

Identity communication

Use case: Identity propagation across ESB:

- Security context

  A security context is passed with all calls to the service via the communication transport (for example, RMI/IIOP, HTTP, or JMS).

- In the message

  The identity information and credentials are passed in the call to the service (that is, within the protocol—for example, in the SOAP header).

Extract identity

The extract identity phase retrieves the identity claim from the message. Within the standard AAA Framework, EI methods can include:

- XML standards-based identity claims such as WS-Security, SAML artifact, or SAML assertions, XML Signature DN
- Transport layer information such as HTTP Authentication header, SSL client certificate, HTTP cookie value
- Network metadata such as client IP address

None of these methods require any custom processing or programming by a user. A custom method can be used to retrieve a custom identity claim. This section discusses the technical requirements to implement a custom identity claim.

- Inputs: The custom processing stylesheet receives the entire XML message as input. A range of DataPower extension functions (such as dp:auth-info() and dp:variable()) provide access to protocol header values.
- Outputs: The AAA system will consider any non-null value returned as the extracted identity. If the stylesheet returns no value, AAA will consider this as an error. Only auditing and post processing activities, if any are configured, will take place in this event.

If a custom method is used for extracting an identity but a standard AAA policy method is used for authenticating, the custom method should provide certain information for the authentication method chosen.

Extract resource

The standard AAA policy can extract resources using any of the following:

- URL
- HTTP method (such as GET/POST)
- URI of toplevel element in the message
- Local name of request element
- XPath expression

A custom processing method is not available for this step.

Authenticate

This step authenticates the extracted identity to obtain a set of credentials through any one of these standard methods, requiring no user development:

- On-board certificate or XML file listing valid identity claims
- Standards-based integration such as LDAP, RADIUS, XKMS, Kerberos, SAML assertions
- Integration with major access control vendors

Custom authentication processing is also available.

- Inputs: The AAA system delivers the result of the EI phase wrapped in several tags.
- Outputs: The custom process should return a credential to signify authentication. This may take any form, including XML. To signify denied authentication, the custom process should return null or empty. The response from the authenticate phase is passed to the map credentials phase, then to the authorize phase. If there is no credential returned from the authenticate phase, and none is created at the map credential phase, then the authorize phase receives an empty map credentials phase. The standard AAA system then fails authentication and authorization.

Map credentials

Optionally, map the credentials from the authenticate step to a different format, suitable for the authorization step.

- Inputs: The map credentials phase receives the result of the authenticate phase as its input, wrapped in several tags.
- Outputs: The AAA system imposes no requirements on the output of the map credentials phase. Any result is accepted. If the map credentials phase returns nothing, the standard authorize phase may fail, even if the Any Authenticated option is chosen. The presence of mapped credentials at the authorization stage represents successful authentication to the standard AAA system. A custom authorization method could change this behavior, however.

Map resources

This works the same way as map credentials, but this time mapping the extracted resource, such as rewriting the requested URL. Another example, the XS40 can concatenate the SOAP endpoint and the Web service operation to create a synthetic URL. This allows a traditional Web access control server (which only knows URLs as resources) to be used for authorizing access to the Web services.

- Inputs: The map resources phase receives the result of the extract resource phase.
- Outputs: The AAA system imposes no restrictions on the output of a map resources phase. If no resources are returned, the empty element will be forwarded to the authorization phase for processing.

Authorize

During this step, the mapped credential and mapped resource is used to authorize access. A number of standard methods are provided:

- XML file listing authorized credential/resource pairs
- Standards-based integration such as LDAP group membership, SAML queries, and SAML attributes from authentication
- Integration with major access control vendors

It is possible to use a custom processing method to accomplish authorization.

- Inputs: The authorize phase receives four elements within a container for input. The ancillary information can be significant, such as when employing SAML. Check this value carefully, as it may contain information useful to the custom process.
- Outputs: The AAA system looks for a root <approved> node within the response from a custom authorize stylesheet. The response can be as simple as just <approved />. If the root node is not <approved>, the AAA system considers the authorization phase to fail, and thus the entire AAA action. To be explicit, the custom stylesheet can return a <declined> node for rejection. The contents of the <declined> node typically contain the reason for the decline.

Post processing

Within the standard AAA Framework, post processing provides for easily configured accounting and auditing. Post processing offers these features:

- Counters for authorized or rejected messages
- Logging of authorizations or rejections
- Security post processing, such as inserting a SAML assertion or a Kerberos ticket
- Custom processing

The post processing phase receives the output of the authorize phase. Post processing also receives the following variables:

$identity	Output of the extract identity phase
$credentials	Output of the authenticate phase
$mappedcreds	Output of the map credentials phase
$resource	Output of the extract resource phase
$mappedres	Output of the map resource phase

The output of the post processing phase is returned as the result of the AAA processing action.

Using multistep probe

To view the contents of any phase, enable the multistep probe for the firewall or Web services proxy service employing a document processing policy containing a AAA action.

AAA action

An AAA action is a DataPower object that references a specific AAA policy. It can be used on an existing processing rule. An AAA action can be implemented on every kind of DataPower service:

- XML firewall
- Multi-protocol gateway
- Web Service Proxy
- XSL Proxy

While creating an AAA action, a reference to an existing AAA policy must be provided. If a AAA policy does not exist, it must be created, in keeping with the requirements and constraints of the current application.

> **Note:** An AAA policy is a DataPower object that can be referenced by different AAA actions, provided that they all (policy and actions) belong to the same domain.

Configuring an AAA policy

This section describes the steps required to define an AAA policy. Note that some steps are optional, depending upon the requirements of the application.

1. Extract identity. This step is mandatory. This is the very first step that you must define in an AAA policy. The purpose of this extraction is to answer the question: Who are you?

   Data may be extracted from different sources and using different methods:

   – SOAP message
   – XML content
   – HTTP header
   – Automatically by DataPower or through a custom template (XSL stylesheet)

   It is possible to select several extraction methods. The DataPower AAA framework executes these methods in the following order:

   a. HTTP’s authentication header
   b. Password-carrying usernametoken from WS-Security header
   c. Derived-key UsernameToken element from WS-Security header
   d. BinarySecurityToken element from WS-Security header

   Therefore, it is possible to use a single AAA policy, which is in charge of extracting identity from different sources. These sources can implement different kinds of identification methods.

2. Authenticate. This step is mandatory. In this step, you define the method that is used to authenticate the asserted identity. There are many methods for authentication, a few of which are:
   - Using a Lightweight Directory Access Protocol (LDAP) enabled directory server, such as IBM Tivoli Directory Server.
   - Using IBM Tivoli Access Manager for e-business.
   - Using a DataPower AAA info file: An XML file that contains authenticated identities and some values related to them.
   - Using a custom template: Authentication may be managed through an XSL stylesheet that is responsible for authenticating the asserted identity. An empty output of this XSL means authentication failure to the DataPower AAA framework.
3. Map credentials. This step is optional. It may not be possible to use the extracted identity to authorize the use of the requested resource, especially when authorization is managed by an external authority. Interoperability between systems (DataPower device and system in charge of authorization) may require mapping between credentials. The supported methods that can be used for this mapping are:
   - IBM Tivoli Federated Identity Manager
   - AAA Info file
   - XPath query
   - Custom template
   - WS-SecureConversation
4. Extract resource. This step is mandatory. The goal of this step is to retrieve the requested resource service, which is essential to complete authorization. The purpose of this extraction is to answer the question: What do you want to do? Several items are proposed to identify a resource, for example:
   - Recover resource from local name of the requested of the message.
   - HTTP operation.
   - XPath expression.
5. Map resource. This step is optional. Interoperability between systems (DataPower device and system in charge of authorization) may require mapping between resources. Here are the supported methods and formats that can be used for this mapping:
   - The IBM Tivoli Access Manager for e-business objectspace representation
   - AAA info file
   - XPath query
   - Custom template
6. Authorize. This step is mandatory. Credentials and resources, which both may have been remapped, are submitted to the authority in charge of dealing with authorization. This authority may be the DataPower device itself or an external entity, such as:
   - IBM Tivoli Access Manager for e-business
   - An LDAP server, as IBM Tivoli Directory Server
   - DataPower AAA info file

- Custom template

  Authorization may be managed through an XSL stylesheet. Two different outputs are also possible for this XSL:

  - Element <approved/>, which means authorization success to the DataPower AAA framework,
  - Another element, as <declined/>, which means authorization failure

7. Post processing. This step is optional. Tasks may be set as a final step of an AAA policy. For instance, it is possible to add a WS-Security UserName token in the message.
8. Auditing. This step is optional. Audit consists of logging information that may assert that both authentication and authorization have succeeded or failed.

4.4.1 Mechanisms supported by AAA

In this section we discuss mechanisms supported by AAA.

Security Assertion Markup Language (SAML)

These are assertions and protocols for the OASIS Security available at:

http://www.oasis-open.org/specs/index.php#samlv2.0

The Security Assertion Markup Language defines the syntax and processing semantics of assertions made about a subject by a system entity. In the course of making or relying upon such assertions, SAML system entities may use other protocols to communicate either regarding an assertion itself or the subject of an assertion. This specification defines both the structure of SAML assertions and an associated set of protocols, in addition to the processing rules involved in managing a SAML system.

SAML assertions and protocol messages are encoded in XML and use XML namespaces (XMLNS). They are typically embedded in other structures for transport, such as HTTP POST requests or XML-encoded SOAP messages. The SAML bindings specification (SAMLBind) provides frameworks for the embedding and transport of SAML protocol messages. The SAML profiles specification (SAMLProf) provides a baseline set of profiles for the use of SAML assertions and protocols to accomplish specific use cases or achieve interoperability when using SAML features.

For an additional explanation of SAML terms and concepts, refer to the SAML technical overview (SAMLTechOvw) and the SAML glossary (SAMLGloss). Files containing just the SAML assertion schema (SAML-XSD) and protocol schema (SAMLP-XSD) are also available. The SAML conformance document (SAMLConform) lists all of the specifications that comprise SAML V2.0. All SAML assertions may be signed using XML Signature.

Encryption of the <Assertion>, <BaseID>, <NameID>, and <Attribute> elements is provided by use of XML Encryption (XMLEnc). Encrypted data and optionally one or more encrypted keys must replace the plaintext information in the same location within the XML instance. The <EncryptedData> element's type attribute should be used, and, if it is present, must have the value http://www.w3.org/2001/04/xmlenc#Element.

Any of the algorithms defined for use with XML Encryption may be used to perform the encryption. The SAML schema is defined so that the inclusion of the encrypted data yields a valid instance.

You must set up a policy enforcement point (PEP) to handle authorization decision (AuthZ) with a policy decision point (PDP). The PDP can be any security assertion markup language

(SAML) AuthZ provider who can handle a SAML authorization query and produce a proper SAML authorization query response. For more details on SAML refer to:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

More information about the SAML specification is available from:

http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security

XACML PEP/PDP

There is an increasing demand today for a common language for expressing a security policy with a common policy language throughout an enterprise. XML is a logical choice as the basis for a standards-based security-policy language, due to its extensibility with syntax and semantics within the scope of security and policy management.

The Extensible Access Control Markup Language (XACML) is an XML-based language that is designed to express security rules, policies, and policy sets that define the access rights of users (subjects) to resources. The XACML specifications were developed through a collaborative effort involving various companies including IBM, Sun™ Microsystems, and Entrust. XACML was ratified by the Organization for the Advancement of Structured Information Standards (OASIS) over five years ago. The goal of XACML is to provide a common policy language that allows organizations to enforce all of the elements of their security policy across all of the components of their IT infrastructure.

After authentication comes authorization. Authorization means making a decision about whether an authenticated identity is allowed to access a resource. Two key inputs to making an authorization decision is an authorization policy and the security attributes of an authenticated user/system.

Policy decision and enforcement

To make an authorization decision, policies need to be in place. These policies are enforced by a PEP that relies on the decision made by a PDP. An example of a PEP is DataPower in the role of an enterprise service bus, which allows access to services based on the authorization decisions received from the relevant PDP.

Access to a resource is typically controlled by an appropriate resource manager, which is the logical policy enforcement point corresponding to that resource. Administrators will update security policies through the resource managers or administer policies through appropriate policy decision points. In a typical deployment, you can find several enforcement points. Each of these enforcement points can have its own mechanisms to enforce security for the incoming requests.

In this scenario, the DataPower device usually works in the role of a policy enforcement point and coordinates (via a logical process called the context-handler) requests from a requestor, and queries an external PDP for a permit or deny decision and allows or prevents access to the protected resource.

The enforcements points rely on PDPs to make decisions. These PDPs contain the security policies defined in the infrastructure. The requirements and thus the policies are different, depending on the security domains and the application platforms.

In some cases, such as for an XACML authorization scheme, the DataPower device can also act as the internal PDP (as well as a policy authorization point that holds the XACML policies) and provide a permit or deny decision based one or more XACML policies stored on the device. For example:

- Processing policy for a Web service
- Integrating Tivoli software products Tivoli Access Manager for e-business (TAM) or TIFM and Tivoli Directory Server (TDS)

This maps the different steps of the DataPower AAA framework to specific IBM products and standards that may be used in each of these steps. For instance, for authentication, you may choose to use TFIM in a situation where you are implementing federated identity. DataPower acts as the point-of-contact server for that particular use case and defers the authentication to TFIM. Likewise, DataPower can defer authorization to TAM. (It should also be noted that when using TFIM, that TFIM will also use TAM for the same purpose.) Likewise, in the audit step, you can choose to log audit messages to the IBM Common Event Infrastructure.

Implementation

DataPower makes use of XACML authorization PEP and PDP to control access to a protected resource. The DataPower device usually works in the role of a PEP and coordinates (via a logical process called the context-handler) requests from a requestor, and queries an external PDP for a permit or deny decision and allows or prevents access to the protected resource.

During the authorization phase of an AAA policy, one can select Use XACML Authorization Decision. The AAA policy acts as an XACML PEP. The PEP allows the XACML PDP to decide whether to authorize access. The DataPower device supports the *mandatory to implement* and *optional* functions that are listed in Section 10.2.8 of the OASIS Standard, eXtensible Access Control Markup Language (XACML) Version 2.0, dated February 1, 2005.

Summary

One challenge of having multiple PEPs and PDPs in the infrastructure is that they are often administered by different entities. Providing integrated and centralized decision capabilities reduces the administrative tasks related to policy management.

There are many points of enforcement in the enterprise where a policy could be enforced over the wire from the Internet, within the extranet or by mail, VPN, remote access, and so on. DataPower as a security appliance helps to manage the configuration from centralized or decentralized endpoints, thus becoming a significant solution in controlling the costs of managing and implementing a robust and effective security policy. DataPower fits into the future direction to consolidate tools that communicate and operate via XACML language and enforce via PDP and PEP.

XACML scenario

A government agency provides medical and other services to veterans and has geographically distributed hospitals and physicians that need authorized access to private patient records over the Internet. This use case involves a physician A from ABC hospital making a request to read the medical record of her patient. Only the patient’s physician from ABC hospital has the security to access the records.

1. The physician logs in to her PC and uses the hospital software system to update the patient’s record located remotely at a government agency data center. The ABC hospital software generates a SOAP request (with a WS-Security header containing the doctor's username and password) to update the patient data in the government agencies' databases. A standard schema namespace http://www.medico.com/record.xsd is used for all transactions dealing with medical records, (Typically, DataPower would be leveraged to

digitally sign the record to ensure integrity and also encrypt it to ensure privacy, but that is out of the scope of this scenario.)

2. The request reaches the DataPower box (via the appropriate firewalls and routers) and is intercepted by the DataPower device acting as the PEP.
3. Extract Identity: The DataPower device extracts the user name and password from the WS-Security header. In a more typical scenario, the record would have first had its digital signature verified by DataPower and then encrypted prior to identity extraction.
4. Map credentials: None. (Output credential in the authentication is used.)
5. Authentication: The PEP uses the user name and password to authenticate the doctor using an XML file stored on the DataPower device. The output credential from the xmlfile is used in the authorization phase. This could very well have been any other authentication scheme returning an output credential like TAM, and so on.
6. Transform input: The PEP forms a XACML request using an XSL stylesheet XACML_Request_Transformer.xsl to transform the input SOAP message into a XACML request. The XSL stylesheet extracts the output credential obtained from the AU step from the DataPower device (using DataPower built-in functions and variables) and places it in the subject tag in step 7. The resource and action tags are copied directly from the SOAP request (using xsl:copy-of select=). The subject, resource, and action that form an XACML request may be structured differently in the SOAP input depending on requirements. In that case the spreadsheet would have to do more work than a simple copy of tags from the input.
7. The XACML request is of the following form (Example 4-1).

Example 4-1 XACML request

```
<Request xmlns="urn:oasis:names:tc:xacml:2.0:context:schema:os">
   Subject> <!-This is the identity that needs to access the resource -->
      <Attribute AttributeId= >
         <Attribute Value></Attribute Value>
      </Attribute>

   </Subject>

   <Resource> <!-The protected resource that needs to be accessed -->
      <Attribute AttributeId= >
         <Attribute Value>   </Attribute Value>
      </Attribute>
   </Resource>

   <Action> <!-The action the identity wants to take on the resource -->
      <Attribute AttributeId= >
         <Attribute Value>   </Attribute Value>
      </Attribute>
   </Action>
   <Environment/> <!-- This is optional -->
</Request>
```

8. Extract resource (ER) is not really required because in this case the resource is being extracted directly from the input by the XSL stylesheet. No resource mapping is performed.
9. Authorization: The XACML request is forwarded to the deny-based policy decision point, which compiles the policy using the XMLManager (or retrieves a compiled policy from the cache) and validates the XACML request against the XACML policy. This particular

XACML policy is of the following form. Policies can vary and the XACML language is quite flexible, but that discussion is out of scope of this book. Look in the appendix for the actual policy used.

Example 4-2 Authorization

```
<PolicySet   xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os">
   <Target>
      <Resource> <!-If the target resource (or subject) is not matched the PDP 
rejects-->
      <Attribute AttributeId= >
         <Attribute Value>
            </Attribute Value>
      </Attribute>
      </Resource>
      </Target>
<!-Multiple rule results are combined by RuleCombining Algorithm - Policies 
contain rules'
   <Policy PolicyId="urn:vha:ReadWrite:policy" RuleCombiningAlgId=> <!-several 
rule combining algorithms are possible>
      <Target/>
<!-Rules target matches attribute value to a fixed tag (designator) or a dynamic 
XPATH expression (selector)  - Rule returns permit or deny depending on the 
"Effect" parameter '
      <Rule RuleId="urn:vha:ReadWrite:rule" Effect="Permit">
         <Target>
            <Resource>
               <ResourceMatch
                  <AttributeValue>
                     <AttributeSelector> OR <ResourceAttributeDesignator>
               </ResourceMatch>
            </Resource>
         </Target>
      </Rule>
   </Policy>
</PolicySet>
```

10. Permit or deny returned by PDP to PEP, which returns a SOAP message to the client. The PDP could optionally return obligations that the PEP must fulfill successfully for a permit (not shown in Example 4-2).

Resources for further reading are:

- *Understanding SOA Security Design and Implementation*, SG24-7310

  http://www.redbooks.ibm.com/abstracts/SG247310.html?Open

- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*

  http://www.redbooks.ibm.com/abstracts/REDP4365.html?Open

- The XACML specification

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml

- OASIS eXtensible Access Control Markup Language (XACML) Version 1.1

  http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf

- *WebSphere Service Registry and Repository Handbook*

  http://www.redbooks.ibm.com/abstracts/SG247386.html?Open
- Creating XACML Based Authorization using DataPower, Case Study by Karen Punwani

Kerberos and SPNEGO

Standards for Web services security are required to ensure interoperability in heterogeneous environments. The WS-Security standard and its dependents such as a WS-Trust and WS-Security Policy are important standards in this area. Web services' security interoperability should be achievable via these security open standards, but there are extra steps required to achieve interoperability between Microsoft .NET and IBM WebSphere technologies.

To this point, WS-Security is security token agnostic, allowing the use of a variety of security token types. Kerberos is an integral part of the security system in Microsoft Active Directory® server, so it has become popular in intranet scenarios, and DataPower supports most types of Kerberos tokens. For example, single sign-on (SSO) throughout an enterprise has to take into account user desktop authentication. It is desirable for users to authenticate to their desktops and then have implicit authenticated access from those desktops to other enterprise resources, such as Web applications, in this scenario against such back-end services as WebSphere Application Server.

But the proliferation of user IDs and passwords combined with the increasing number of applications in a user domain has increased the complexity of implementing such complicated security scenarios as SSO. To this end, many organizations have turned to Kerberos technology. The underlying principle of Kerberos-based security is that machines trust other machines through a third-party Kerberos Key Distribution Center (KDC). The KDC has two major components—an authentication server (AS) and a ticket granting server (TGS). The authentication server shares a secret with each client and server in the Kerberos realm (domain). These shared secrets are used to derive the symmetric keys used to exchange encrypted information between a client or server and the KDC. If a user authenticates to the KDC, that identity and authentication status are then used to request services that are in that trusted realm. Users can then access services on other machines that participate in this trusted network without again being prompted for a user ID and password. Kerberos authentication is viewed as one way to reduce some of the complexity that is seen by users.

The SPNEGO protocol (Simple and Protected Negotiation mechanism) enables an HTTP-based Kerberos authentication solution within DataPower. This means that instead of requiring an application to be Kerberos-aware, DataPower can act as a proxy and provide the primary interface to the SPNEGO Trust Association Interceptor (TAI) to authenticate users.

It should not be construed that DataPower is being positioned as a SSO server. Instead, one should look to Tivoli for single sign-on. But SPNEGO is now an option with DataPower when a solution has Active Directory and the solution requires single sign-on to Web services applications from the desktop.

To enhance DataPower administration security, access to a DataPower device can now be controlled through use of SPNEGO tokens. When a user’s browser presents a valid SPNEGO token, the user sees the control panel, not the login panel. The user acquires a valid SPNEGO token by first visiting the Microsoft authentication page, which issues a token.

Auditing

Auditing includes maintaining detailed, secure logs of critical activities in a business environment. These critical activities can be related to security, content management, business transactions, and so on. Examples of security-related critical activities that can be audited are login failures or successes, unauthorized or authorized access to protected resources, modification of security policy, non-compliance with a specified security policy, health of security servers, and so on.

An audit logging service provides mechanisms to submit, collect, persistently store, and report on audit data submitted as events. The events can be in a common format, such as *Common Base Event* (CBE).

Which events are audited and stored is defined in an audit policy. This policy needs to define which events are important, how long to keep the data, and whether to keep the audit data in a tamper-resistant form. Audit data must be collected for all of the security services.

Challenges

Figure 4-17 illustrates the challenge of providing access to consistent audit information across the environment.

Audit information is required to be gathered from every component along the transaction path. That is, important events need to be logged and available for real-time or later forensic review. These events can be security specific, for example, authentication, authorization, identity mapping, identity provisioning, and policy management-related, or they can be of a more general nature, such as data or application access.

There are a number of challenges when implementing an audit:

- Audit information is often physically located on the server that generates the event. For example, for all the different components, such as proxy, Web application server/portal server, and the ESB, a back-end database might write events to each component's log file. This distribution of audit log information makes it difficult to later trace the events of a transaction end-to-end, because each log needs to be accessed. Additionally, different tools might be required to access this log information.
- Audit log format is often different on every component that generates an event. This is especially true when there is a heterogeneous mixture of middleware and applications. Real-time or forensic inspection of these logs becomes a difficult process of trying to understand the different log formats and collate related events.
- Compliance to internal or regulatory policy is very difficult to check. There is no automated way of knowing whether the security events that are being generated by the individual components are indicating compliance with policy.

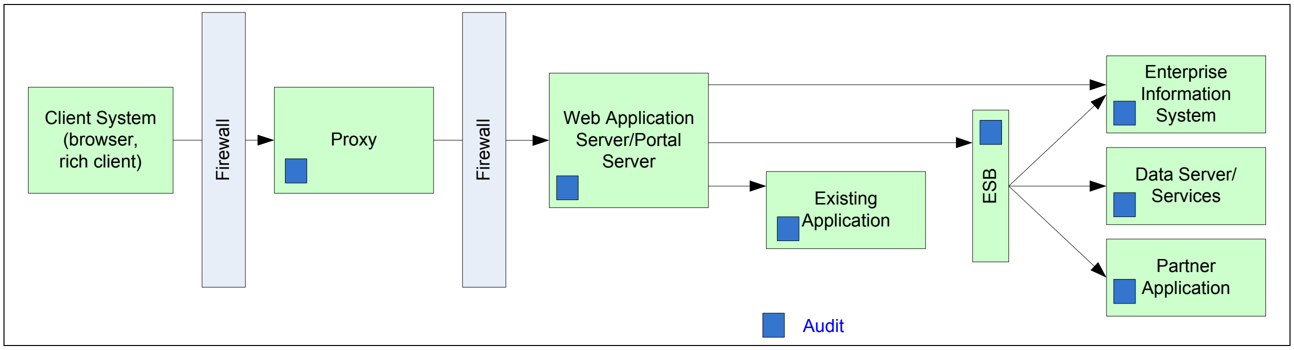

Client System (browser, rich client)
Firewall
Proxy
Firewall
Web Application Server/Portal Server
Existing Application
ESB
Enterprise Information System
Data Server/ Services
Partner Application
Audit

Figure 4-17 Audit example

The DataPower devices can participate in the end-to-end auditing chain. Care must be taken to generate the correct log information and then verify compliance of the end-to-end system with internal and regulatory policy. In addition to out-of-the-box logging, one can build application-level logging by configuring log targets.

Log targets capture logging messages that are posted by the various objects and services that are running in the system. Target types enable additional capabilities that include rotating files, encrypting and signing files or messages, and sending messages to remote hosts. Messages in log targets can be restricted by object filters, event category, and event priority.

Many log target types can be used including SNMP, SMTP, caching, file on the device, and so on. By default, a log target cannot accept messages until it is subscribed to one or more events.

One can use DataPower logs as the raw material to build higher level auditing capabilities by building application level logging. Application level logging is used as building blocks for specific auditing capabilities. In this way DataPower can participate in the end-to-end audit trail. In addition, offboard logging (out of memory) is also feasible.

Resources for further reading

More information about CBEs can be found at:

http://www.ibm.com/developerworks/library/specification/ws-cbe/

4.5 Integrating IBM DataPower and Tivoli for SOA Security

IBM has a good set of management solutions around security and identity management, one of the top concerns of enterprises deploying SOA. For example, Hercules accelerates the XML layer in SOA, but the total ROI for the customer is not realized until the full SOA implementation is done. IBM is uniquely positioned to help customers do so.

Hercules is tightly integrated with Tivoli security products (Federated Identity Manager, Tivoli Access Management, and Tivoli Identity Manager). Tivoli provides SOA Security Management with:

- Policy management: WS-Policy, WS-SecurityPolicy
- Federated Identity Management: Liberty, SAML 2.0, WS-Federation, WS-Security
- Auditing and compliance for SOA: Compliance Automation)
- User provisioning: WS-Provisioning/SPML 2.0.

XML-level protection enhances SOA security management. This simply maps the different steps of the DataPower AAA framework to specific IBM products and standards that may be used in each of these steps. For instance, as in Figure 4-18, for authentication, you may choose to use TFIM in a situation in which you are implementing federated identity. DataPower acts as the point-of-contact server for that particular use case and defers the authentication to TFIM. Likewise, DataPower can defer authorization to TAM. (It should be noted when using TFIM that TFIM will also use TAM for the same purpose.) Likewise, in the audit step, you can choose to log audit messages to the IBM Common Event Infrastructure.

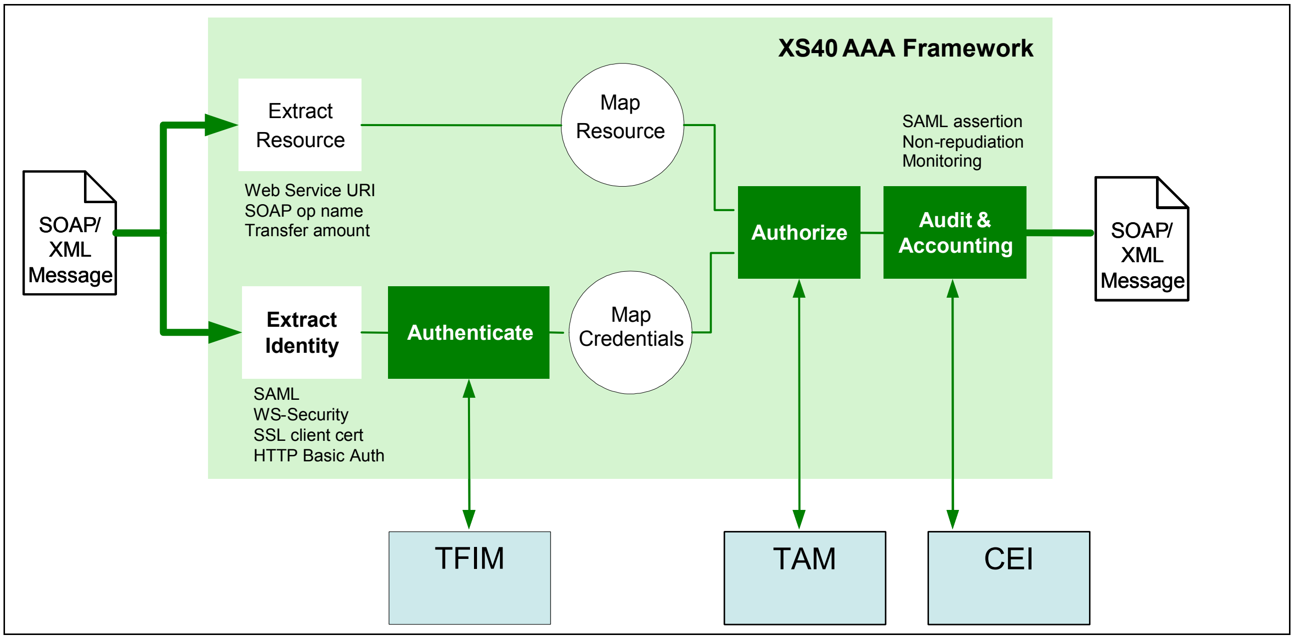


Figure 4-18 Combination of DataPower and Tivoli Solutions for AAA Framework

Figure 4-19 shows a variety of users who are connecting to the applications at the bottom right. In this secured environment DataPower acts as the point-of-contact server (where requests actually arrive at the system). DataPower can then use the services of TFIM or TAM to authenticate and authorize access to the applications. DataPower would then pass authenticated and authorized requests on to the applications running on J2EE application servers, for example.

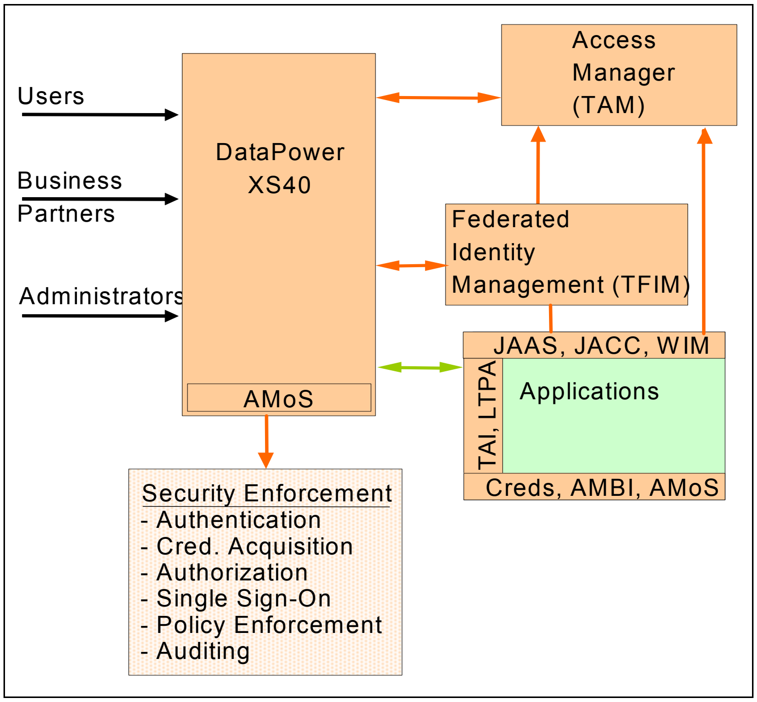


Figure 4-19 DataPower and Tivoli Integration

TAM provides authentication and authorization.

TFIM provides federated identity between organizations.

4.5.1 Using WebSphere DataPower, Tivoli Security tools, and open standards

Figure 4-20 summarizes the use of security capabilities offered by DataPower and Tivoli.

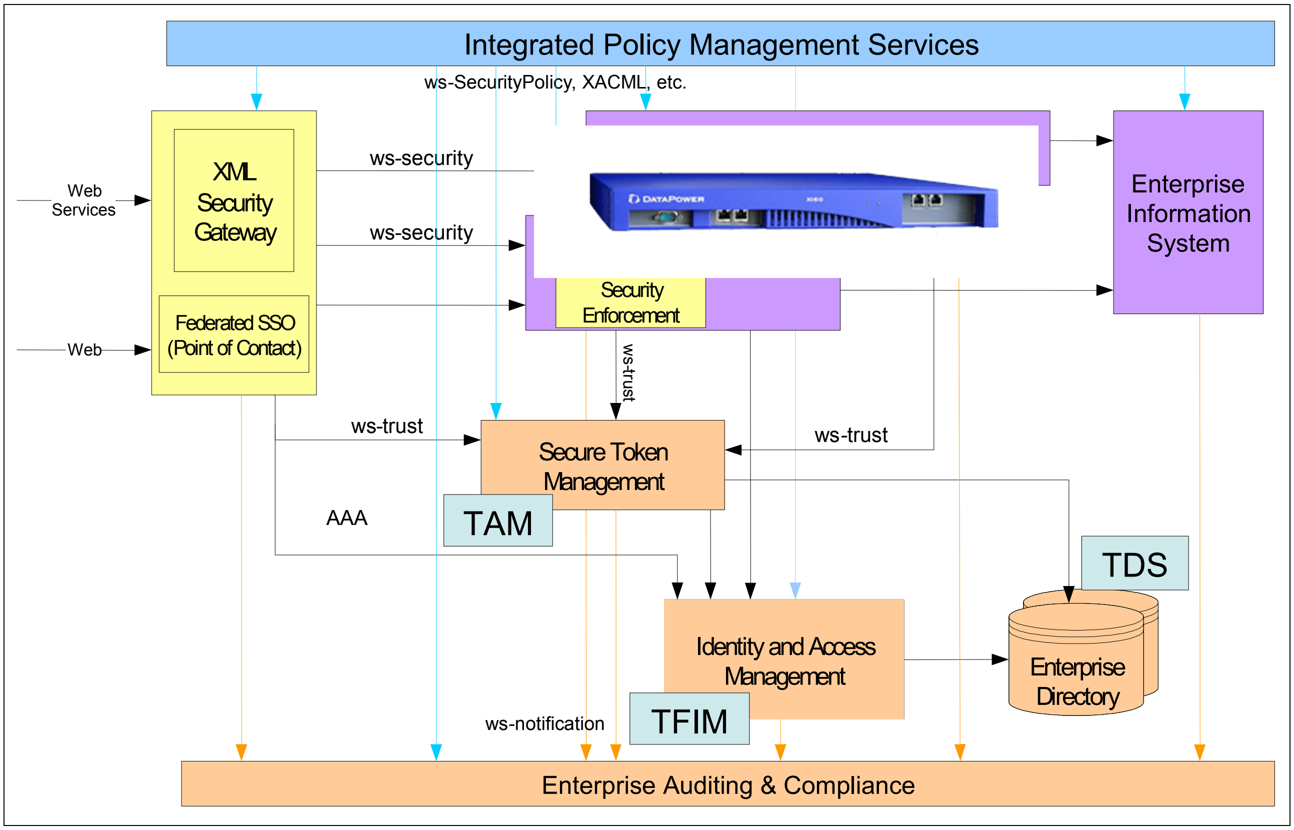

Integrated Policy Management Services
ws-SecurityPolicy, XACML, etc.
Web Services
Web
XML Security Gateway
Federated SSO (Point of Contact)
ws-security
ws-security
Security Enforcement
ws-trust
Enterprise Information System
ws-trust
Secure Token Management
ws-trust
AAA
TAM
TDS
Identity and Access Management
Enterprise Directory
ws-notification
TFIM
Enterprise Auditing & Compliance

Figure 4-20 Integrating security capabilities

4.5.2 AAA using Tivoli Access Manager (TAM)

The use of Tivoli Access Manager for e-business with DataPower is described in this section. TAM's powerful access control mechanisms offer a comprehensive access management solution and complements DataPower's strong Web services capabilities.

We can configure a Web service in TAM as well as build a TAM access control policy to regulate access to the service. The steps required to configure DataPower to use a TAM infrastructure include generating the necessary configuration files and keystores, as well as integrating these configuration artifacts into a DataPower processing policy.

Use

Use Tivoli Access Manager for secure Web SSO and XML Web services. XS40 provides the first line of XML defense and enforces access policy stored in TAM.

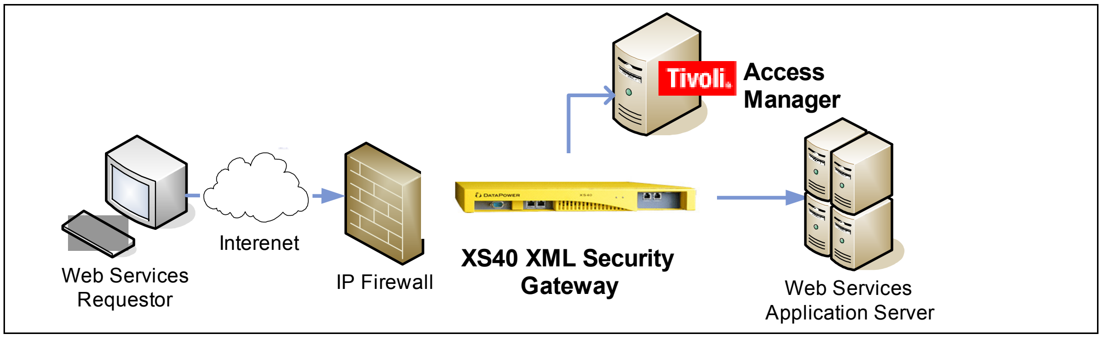


Figure 4-21 Using Tivoli Access Manager

11. Federated identity among partners

For example, an insurance company allows a retailer's employees to check their insurance profile, stored in internal application servers, and not available previously.

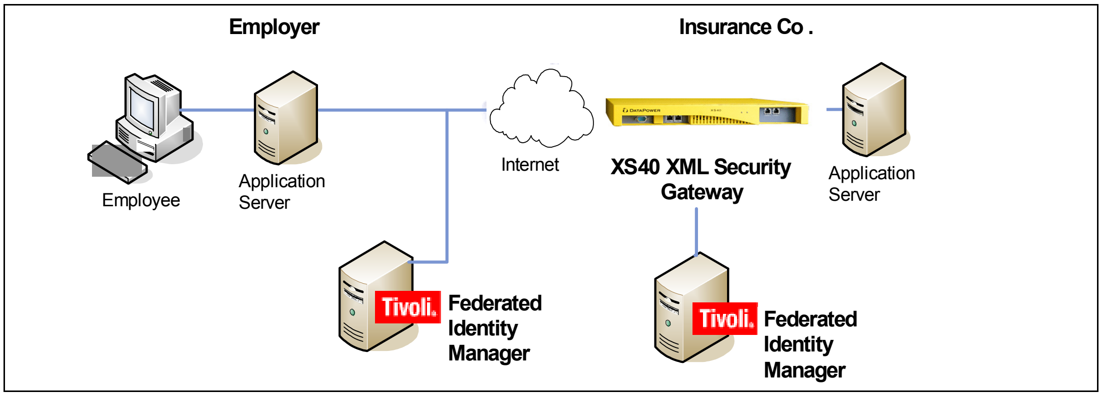

Employer
Insurance Co .
Employee
Application Server
Internet
XS40 XML Security Gateway
Application Server
Tivoli. Federated Identity Manager
Tivoli. Federated Identity Manager

Figure 4-22 Using Tivoli Federated Identity Manager

4.5.3 IBM Tivoli Access Manager for e-business

Tivoli Access Manager for e-business is the flagship IBM enterprise security product and is recognized as an industry leader. TAM offers centralized policy management, scalability, and flexibility to provide enterprises with secure and easily managed network infrastructure.

TAM and DataPower

To use TAM to perform authentication and authorization, a TAM (client) object must be configured on the DataPower appliance. When TAM is specified in a AAA processing policy, the TAM client is used to communicate with the TAM server infrastructure.

Using a TAM client consists of two steps—creating the necessary TAM configuration files and then using them to create a DataPower TAM client. The TAM files that are created are:

- A configuration file that contains TAM infrastructure information
- A keystore file that contains the certificates necessary for communication with the TAM server
- A stash file (contains the password for the keystore)

A DataPower TAM client object uses these files to interact with the TAM server specified in the configuration file. The TAM client configuration files can be used by multiple TAM clients in the same DataPower domain, but cannot be shared amongst other domains.

Note that the remainder of this section discusses the TAM infrastructure of the management zone, as illustrated in Figure 4-23.

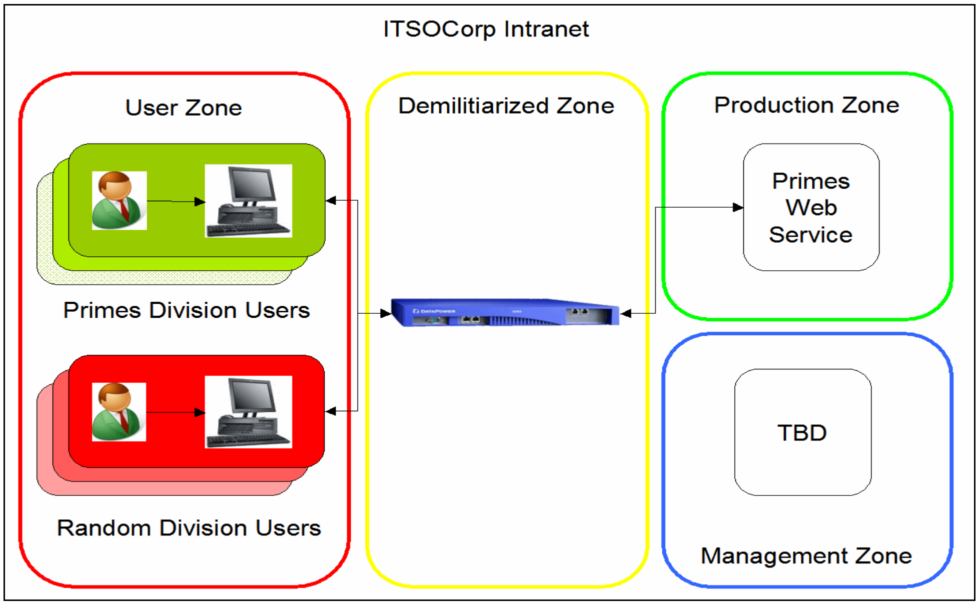


Figure 4-23 Intranet deployment architecture

TAM and Web services overview

To enforce the security policy defined in “Federated Identity Management” on page 80 using TAM, you need to design a virtual representation of the primes Web service. In a TAM environment, the entity that represents resources like the Web service is called a *protected object space*. TAM applies the defined security policy to the protected object space by associating appropriate access control policies to the objects in the object space.

4.5.4 AAA using LDAP Directory

This section describes the methods used to perform authentication and authorization operations using an LDAP-enabled directory server. Building on the concepts defined in the previous sections, we discuss building additional processing rules at the operation level of a Web Service Proxy.

IBM Tivoli Directory Server (TDS)

Tivoli Directory Server is the primary IBM LDAP accessible directory server. TDS is a robust and scalable enterprise identity management product.

The remainder of this section discusses the TDS infrastructure of the management zone illustrated in Figure 4-24.

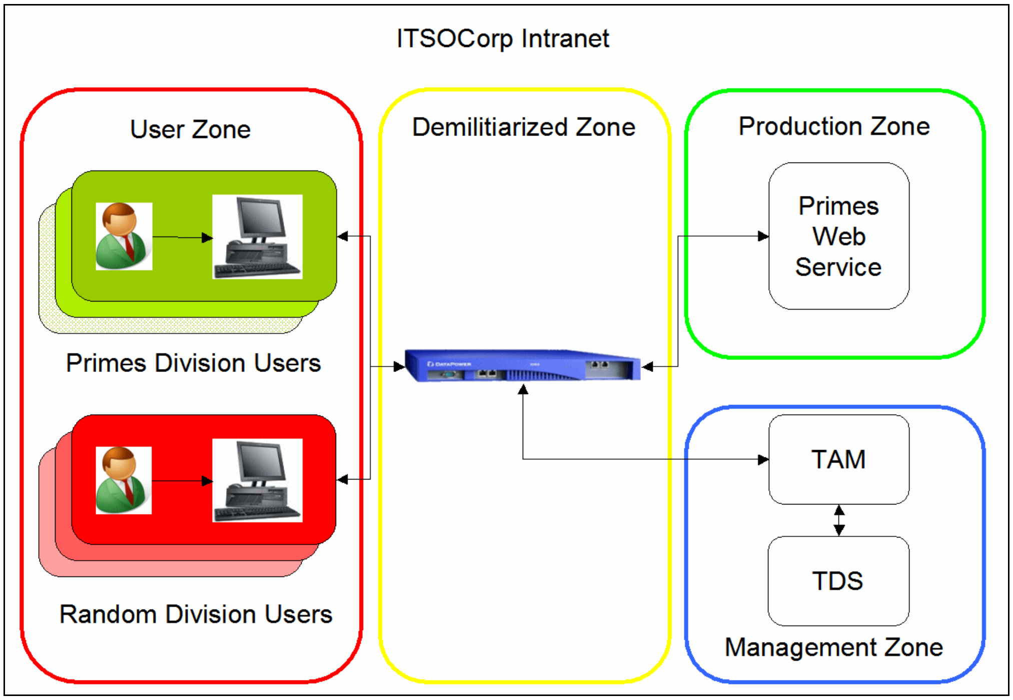


Figure 4-24 Deployment architecture using Tivoli Directory Server

TDS and DataPower

A DataPower AAA policy can use TDS (or any LDAP accessible directory server) for authentication and authorization. The AAA policy retrieves the user's password from the directory server and compares it with the password extracted from the request. Conversely, for authorization, DataPower checks for the user's membership of a specified group in the directory.

4.5.5 Summary

This section described the concept of the AAA policy with respect to DataPower SOA Appliances. In short, DataPower AAA policy objects provide a powerful mechanism to enforce access control for SOA deployments. The wide-ranging support of user identification, authentication, and authorization techniques of the AAA policy object offers a great deal of flexibility.

4.6 DataPower security scenarios

While examining every possible threat is beyond the scope of this book, we want to include an example to demonstrate how DataPower can protect the WebSphere Message Broker in a way that WMB itself cannot.

As background, we examine three different scenarios for using DataPower as part of the security infrastructure. These scenarios describe common cases for using DataPower, describe how the scenario is implemented with DataPower technology, and show why it is the best choice in each case.

- Using DataPower as an XML firewall
- Using DataPower as a Web services Gateway
- Using DataPower to secure service integration with WMB

The use case is based on a DataPower XS40 appliance. One could also use the DataPower XI50 if you require its integration capabilities.

4.6.1 Scenario one: DataPower typical security

This scenario illustrates IBM WebSphere DataPower as an XML firewall or Web services Gateway offering, with the purpose of protecting enterprise IT assets protection through XML security. See Figure 4-25.

- Comprehensive Web services security on transport and message levels
- Comprehensive XML threat protection

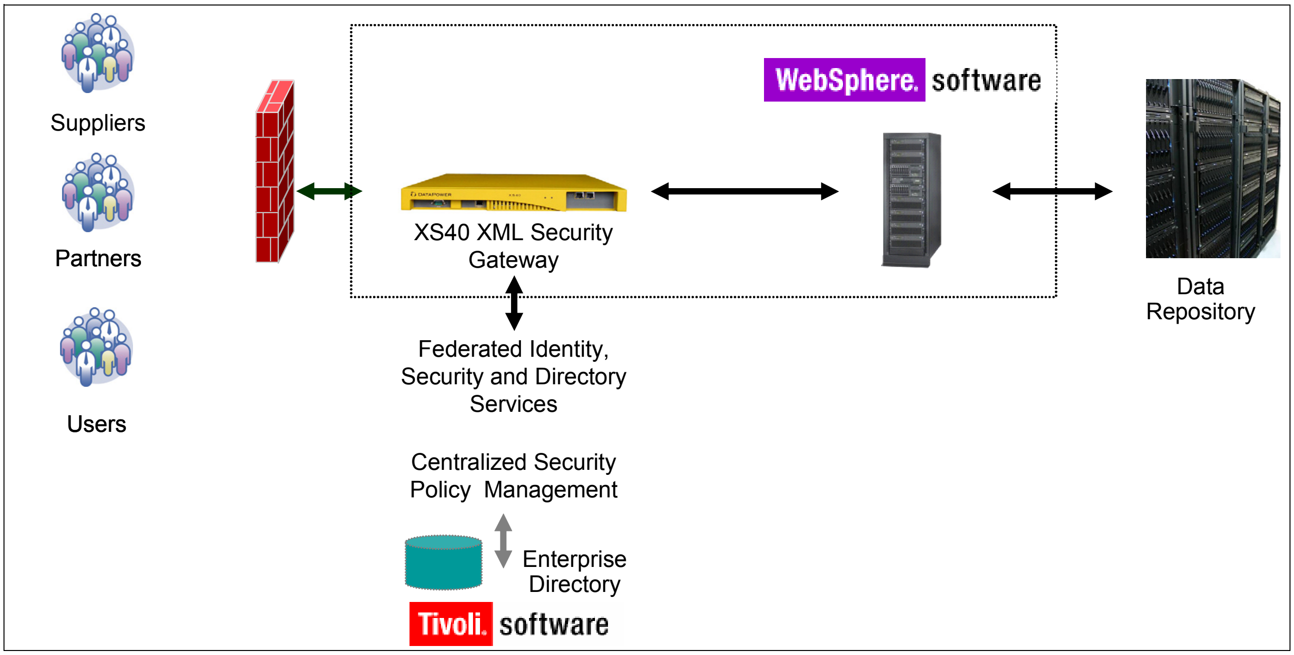

Suppliers
Partners
Users
WebSphere. software
XS40 XML Security Gateway
Data Repository
Federated Identity, Security and Directory Services
Centralized Security Policy Management
Enterprise Directory
Tivoli. software

Figure 4-25 DataPower securing the back-end Web applications

As a first line of defense to securely implement external Web services, DataPower secures many applications and aggregate user interactions. Positioning a DataPower appliance in this manner give us a tremendous advantage to face the challenges below:

- Cross-enterprise interoperation: How can security be shared across organizational boundaries?
- Federated interoperability: For example, where do you store shared security states within a distributed network?
- Human and automated service invocations: How do components authenticate and apply access control to each other and to human operators?

- Dynamic service binding (as a result of loose-coupling) enablement: Implies the need to implement runtime security context generation and binding (for example, dynamic trust relationships).
- Global architectural layers impact: Enterprise, application, services, network, and transport layers.

DataPower XS40 simplifies the secure integration of SOA applications by tying the SOA proposition to enterprise infrastructure security management solutions, enterprise identity management solutions, audit and compliance solutions, and other types of management solutions and tooling. Hercules provides one element of a secure SOA deployment. To be useful in an enterprise setting, it has to integrate or pre-integrate with some of the management solutions out there.

4.6.2 Scenario two: WebSphere DataPower as an XML firewall

The DataPower appliance for security is ideally suited for placement in a DMZ. It meets all of the essential requirements of a bastion host. As it is stated earlier in this book, the DataPower appliance is not a general-purpose computer. It is a purpose-built combination of hardware and software that meets a narrowly defined set of needs. It has no user installable software. It does not provide for general-purpose computing. It does not run a windowing system, Telnet, ftp, or any of the other services that expose systems to attack. Instead, the appliance provides a minimally functional system that works well at one thing—processing inbound/outbound data requests.

By taking this radically different approach to design, the DataPower appliance avoids many issues related to the DMZ deployment of bastion hosts. Instead of taking a general-purpose computer and hardening it through hundreds of difficult steps while installing specialized software, IBM clients can simply deploy the device on their network. The device itself is secure by default. In fact, it cannot even be initially configured without first connecting a serial link device. Further configuration is done using a Web browser interface, but that can be limited to accept requests over only one port and one network interface, avoiding exposure to outside traffic. And, even then, administrative actions require authentication.

The device itself has no hard disk, no open ports (other than the network cards), no USB interfaces, and even includes tamper resistant screws. While no device can be safe against all possible attacks, we consider this to be one of the most secure non-military grade devices we have seen.

When building Internet-facing systems, usually the first item to consider is the importance of a DMZ. A standard basic DMZ topology consists of two firewalls with a hardened bastion host between them. The bastion host terminates inbound network traffic and may perform authentication or authorization depending on the business requirements. The key point relevant to this discussion is that the bastion host sits between the two firewalls and is as hardened as possible. Thus, for example, running WebSphere Application Server or the Web services Gateway on such a host is not appropriate. Clients typically run either a Web server or proxy server in the DMZ. In the context of a Web services system needing a point-of-contact server for routing and access control logic, this yields this *standard* topology.

Web services are based on SOAP/XML/HTTP, which is very firewall friendly. What if you have a giant hole in the firewalls? Figure 4-26 shows standard SOA topology.

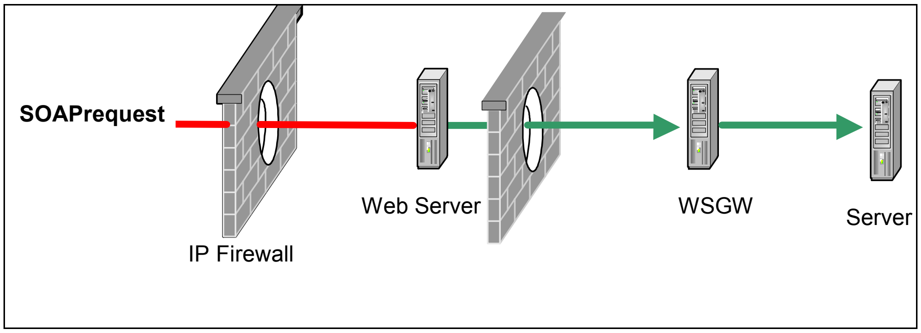


Figure 4-26 Standard SOA topology

What is unfortunate about this topology is that requests must pass through the Web server, which provides little value other than being appropriate for DMZ deployment. The WebSphere Gateway (WSGW) does most of the real work. This is where DataPower comes in. Given what we have already discussed regarding its design as a hardened appliance (even more hardened than any reasonably hardened commercial grade operating system and Web server), we now have a new option. Our modified *standard* topology for Web services systems then becomes as shown in Figure 4-27.

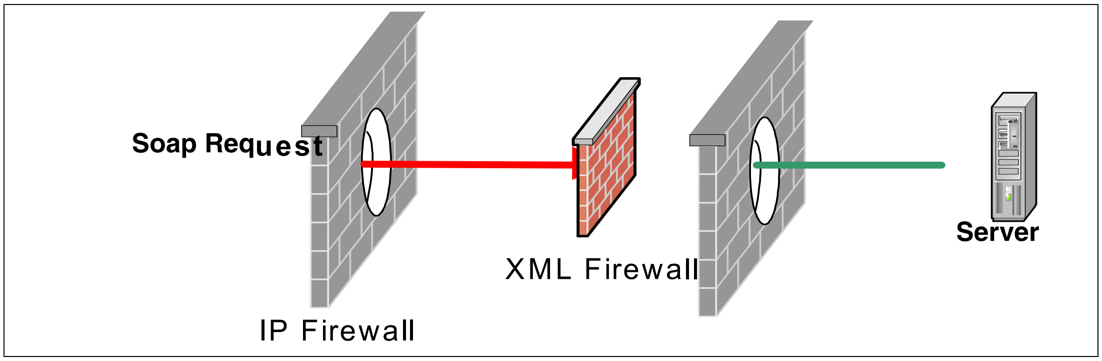


Figure 4-27 SOA topology with DataPower

Notice that this topology eliminates the Web server as well as the Web Services Gateway. Not only is this more efficient, it is also more secure—a win-win situation. This leads us to the following recommendation.

Recommendation: DataPower is appropriate for DMZ deployment. It should be used instead of WSGW in Internet-facing Web services applications.

For testing, we chose a very simple but reasonable attack using a single SOAP message. This message is well formed, and can be sent to any Web service. The problem with the message is that it contains a large number of xml:ns tags. Such a message is easily generated using various programming tools. Sending this to a system does not take a high degree of skill.

When the SOAP message was submitted to this Web service, application server CPU immediately raced to peak usage and remained there. Memory usage for the application server java.exe process also quickly spiked to its maximum configured JVM™ heap size. The application server quickly ran out of memory, became unresponsive for an extended period of time, and then crashed with a javacore.

This is a rather staggering situation. A single attacker can send a single SOAP message that will cause the product application server to fatally crash after first consuming enormous amounts of resources. If such a system was facing an untrusted network (for example, the Internet) it would be supremely vulnerable.

To protect this system, we turn to DataPower. A Web services proxy has been configured on the DataPower box to front the Web service running on WAS. The default values were left intact on the Single Message XML Threats pane. Below is only a small sample of the many XML threat protection configurations available.

XDOS protection

When the application server was reinitialized, and the same request sent through with the DataPower proxy's host and port, the request was rejected by DataPower in less than one second, without WAS ever having to see or expend any resources processing it.

XDOS protection log

DataPower provides robust protection for the particular threat. While we cannot show them all here, DataPower protects against many other threats as well.

To truly harden a system using Web services, several important security steps need to be performed. These (recommended by consultants) include inspecting messages for well-formedness, validating schema, verifying digital signatures, signing messages, implementing service virtualization to mask internal resources via XML transformation and routing, and encrypting data at the field level. In addition, there are other types of monitoring for the types of attacks mentioned above that are needed, as well as configurations to ensure compliance with service level agreements. These intensive actions can be performed by DataPower at the front end at a significant savings in back-end resource consumption and configuration.

In fact, the case for DataPower acting as an XML firewall protecting your systems from XML threats is so strong that we make the following recommendation.

> **Recommendation:** All Internet-facing systems that provide for inbound Web services requests should use DataPower as their XML firewall, even when performance or security is considered unimportant.

There are three basic firewall types:

- Static back end
- Dynamic back end
- Loopback

This pattern represented in Figure 4-28 offers tangible benefits such as:

- Availability as a DMZ component
- Masks internal resources: Host assets not directly accessible
- Secures transport layer: Offers SSL transport
- Validates messages
  - XML well-formedness
  - SOAP schema compliance
- Protects against denial of service: Examines messages for malicious format
- Really fast transform and filter services: Offloads expensive XML parsing from application servers
- Service level management

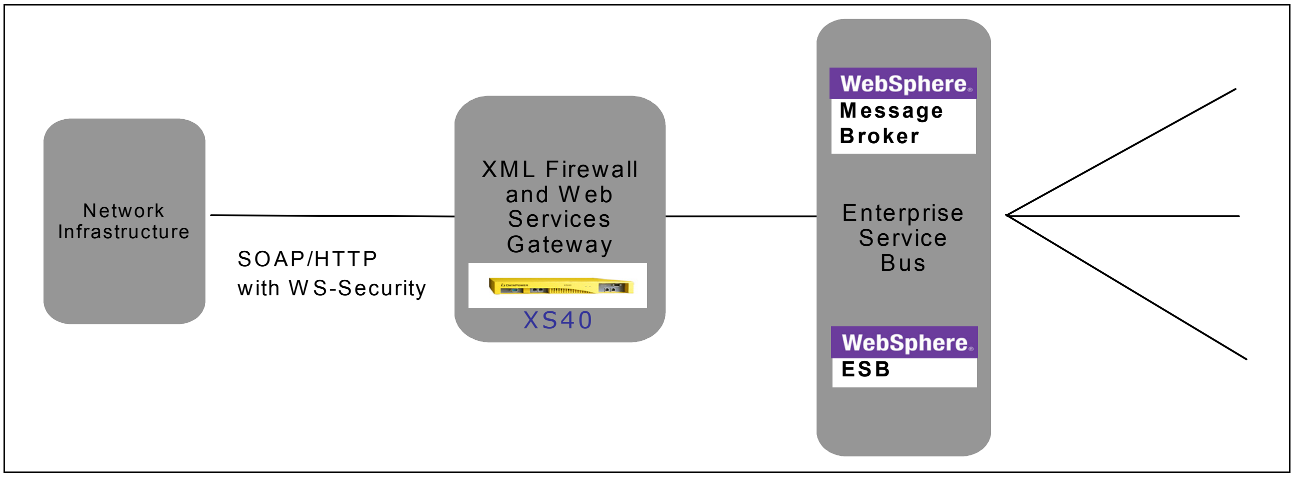


Figure 4-28 DataPower DMZ placement

4.6.3 Scenario three: DataPower as a Web application firewall

This is one of the major DataPower Security Gateway services. We introduced it in 2.6, “Web application firewall service” on page 20.

Overview

The new and emerging tremendous challenge that we face today is not exposing Web user interfaces, but much more direct access to enterprise business logic in the form of Web services. In some cases, core business logic is now accessible outside of the enterprise for the first time. While such access enables greater flexibility and integration with partners, it also exposes business to far greater risks.

Web applications come with intrinsic weakness that hackers can exploit to attack them.

To overcome such concern, you need to use IBM WebSphere DataPower SOA Appliances Services to enforce security within your enterprise. The DataPower Security Gateway Web application firewall (WAF) service can be deployed in front of multiple Web servers to protect Web applications. In this case, DataPower monitors application traffic, performs a wide set of checks for Web application attacks, and reacts in real time. On the WebGui, this service is represented by the icon shown in Figure 4-29.

Figure 4-29 Web application firewall icon

Based on the typical deployment scenario diagram, in Figure 4-30 we depict the DataPower WAF service described in this section.

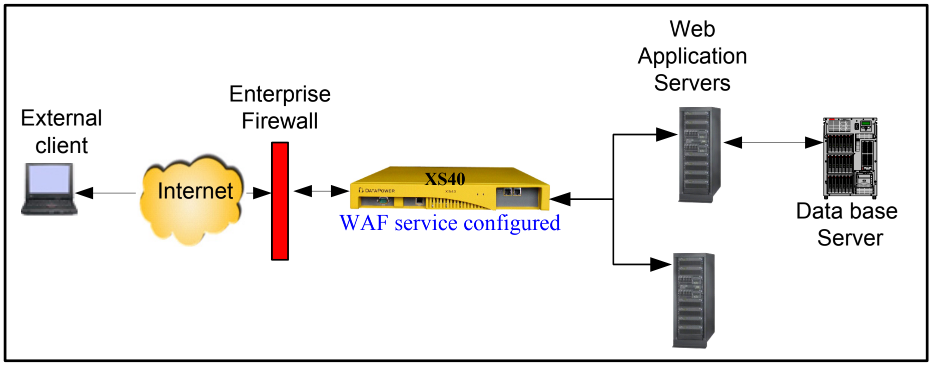


Figure 4-30 DataPower WAF service described in this section

The WAF service provides features to:

- Protect a back-end Web application from attacks using the built-in threat protection.
- Protect access to the Web application firewall.
- Validate parameters from an HTTP request using name-value profiles.

The DataPower WAF service is used to offload some functions from Web-based service applications executing on application servers and to protect access to the enterprise back-end Web applications. For example, authentication, limiting requests, and parameter validation are typically Web application tasks, which can be configured with little effort on the DataPower appliance. These tasks are often performed on J2EE application servers such as WebSphere Application Server.

How it works

Deployed on the company DMZ or intranet, DataPower WAF service enforces security within the enterprise by executing a custom security policy on messages that go through HTTP traffic, before sending or receiving them to or from a back-end Web application. The DataPower Web application firewall service protection against malicious attacks is based on URL encoded strings. This allows defeating a wide range of application-layer attacks by providing immediate protection for applications against targeted vulnerabilities.

WAF service is configured to virtualize a back-end Web application, handle rate limit requests, and enforce an AAA policy. In fact, the WAF service can listen for requests on multiple TCP ports to virtualize or proxy back-end Web applications. The WAF service can also require Web clients to send requests over the Secure Sockets Layer.

In situations when you need to limit the number of requests being sent to the back-end Web service, you can create a rate-limiting policy to control the number of requests.

You can also require clients to provide credentials when trying to access the Web application, which can be enforced using a DataPower AAA policy.

1. A Web browser, as an external client, connects to the Web application firewall service.
2. Once the user has been authenticated, the request is forwarded to the back-end Web applications.
3. The Web application firewall service uses an AAA policy to validate users. This easy-to-apply policy saves time and provides immediate protection for production applications against Web application technology threats. Many of these threats simply pass through the Web-based typical security infrastructure.
4. In a production environment you would also need to secure the connection from the Web application firewall service to the back-end Web application, using either a security token or SSL. Of course, this is optional.

The WAF service offers better support for HTTP-based traffic. For XML-only traffic, we would rather use other DataPower services such as XML firewall, Web Service Proxy, or multi-protocol gateway. HTTP threat protection is different from XML threat protection.

Web application security policy

The Web application security policy is defined using three maps:

Request	References a Web request profile
Response	References a Web response profile
Error	References an error rule

A map is chosen to execute based on the matching rule. More than one map can execute for a given request. Multiple maps can be defined per request and response. Maps execute based on the matching rule definition. Each profile implements the security policy configuration (AAA, HTTP threat protection, rate limiting, session management, and more).

Configuring the DataPower WAF service

Unlike other services on the DataPower appliances, the Web application firewall service does not have a service policy. Instead, it uses a custom Web application firewall security policy. The configuration of this policy does not use actions or have a GUI editor.

Host virtualization

DataPower appliances are typically deployed in the DMZ, which lets them perform pre-processing on incoming messages before they enter a company intranet. Web clients should not know the back-end endpoint of your Web application because if the back-end endpoint changes, your Web clients need to be notified of those changes, and because malicious users can send multiple requests to try to overwhelm the back-end Web application. To hide the Web application end-point address from Web clients, you can define multiple TCP ports on which the WAF service listens for requests. This step is required when you are creating the WAF service.

Defining the WAF service front-end and back-end information

You first need to create a WAF service on the DataPower appliance. Start by defining the front-end and back-end information for the service. The Web application firewall rewrites the client host address and port number URL with the remote host address and port number.

In Figure 4-31, the front-side settings have SSL on, so the external client must connect using https. The part of the URI after the port number is the same URI sent to the back-end Web application.

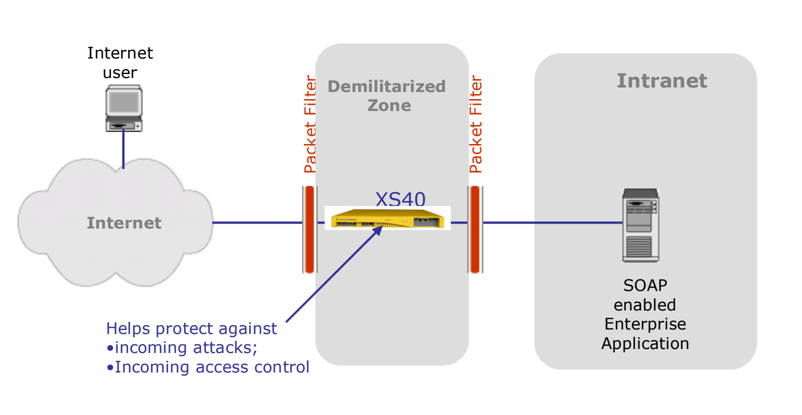


Figure 4-31 Front-side settings have SSL on

Create an AAA policy to protect access to the WAF

In the next steps, you create an AAA policy that extracts the credentials from the message protocol and performs authentication using an AAA file.

Authentication

To authenticate:

1. Create a new AAA policy.
2. Create a new access control policy.
3. Configure an access control policy.
4. Define how to extract a user's identify from an incoming request.
5. Define how to authenticate the user identity stated within the incoming request message.
6. Check the credential mapping method.

Note: In a production environment, an authorized user's list usually resides in a corporate directory server, such as a Lightweight Directory Access Protocol (LDAP) server. For test purposes, the list of authorized users can reside in an XML file named AAAInfo.xml. Since the file is included (disable in production) with every DataPower appliance, use it only for testing.

Authorization

Configure the authorization step in the access control policy to allow any authenticated user.

1. The Define how to extract the resources page specifies how the access control policy determines which resource the client has requested, for example, the URL sent by the client.
2. Check the resource mapping method.
3. The Define how to authorize a request page determines the access rules based on the resource and user.

Auditing

At the end, the AAA policy wizard lets you configure monitoring, logging, and post-processing options.

The authorized counter and the rejected counter keep track of how many requests were allowed or denied access, respectively. The rejected counter has an additional feature to block further requests if a threshold is reached.

The logging section keeps track of the request and response message details for any authorized or rejected access attempts. The amount of message detail logged depends on the log level that you configure.

Post-processing

The post-processing section describes actions that you can apply to the message after the authentication, authorization, and auditing steps.

Rate limiting

High-volume Web sites may need to limit requests during certain periods. For example, a concert ticket Web site may experience a spike in traffic when concert tickets first go on sale. The WAF service lets you restrict the number of request messages within a given time interval. Requests over the specified number can be rejected, shaped, or logged. You can also limit the number of users connected and number of connections per user.

Summary

Securing the HTTP traffic in your company with the DataPower Web application firewall service provides you with a powerful and simple management solution of your back-end Web applications. You can use it to virtualize the endpoint address, handle rate limit requests, and enforce access control. You can configure these items using the Web application firewall service without writing any custom code.

4.6.4 Scenario: Securing a WebSphere Message Broker

In this section we describe the application of the IBM DataPower Security gateway to the services integration scenario. We highlight the integration of service consumers to a variety of applications and service providers of different types. This provides an organization with new ways to leverage and better exploit their applications and data, minimizing the need for multiple point-to-point solutions.

The scenario in Figure 4-32 shows DataPower working in conjunction with WebSphere Message Broker. WebSphere Message Broker may be required for any of several reasons, for example, for the extensive predefined message sets, for the mapping capabilities of DataStage® TX, the need for application-specific code (Java or C), or for complex event processing. The DataPower appliance is used to provide a high-performance Web services security gateway.

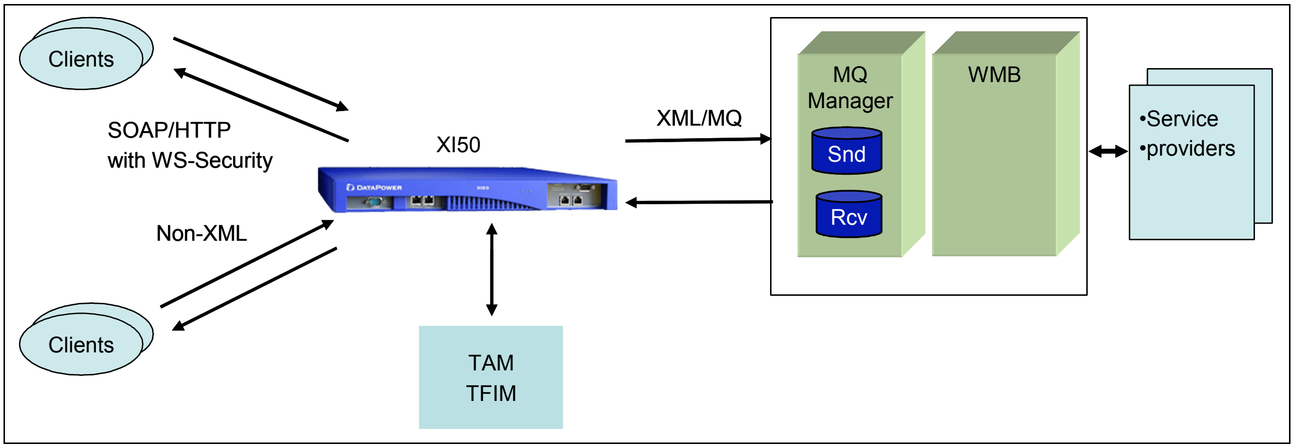


Figure 4-32 Securing WebSphere Message Broker with a DataPower XI50

Clients connect to the DataPower appliance using SOAP over HTTP. The DataPower connects to WMB using MQ. Protocol transformation is a capability of DataPower XI50.

This solution shows:

- Using WMB for existing code, or support for SWIFT, and so on.
- HTTP in MQ Out.
- WMB hosting an XML Web service.
- Use MPG or WS-Proxy, preferably.
- All the security infrastructure is based on XI50.

Alternative

The solution with DMZ placement should ideally be the one represented in Figure 4-33.

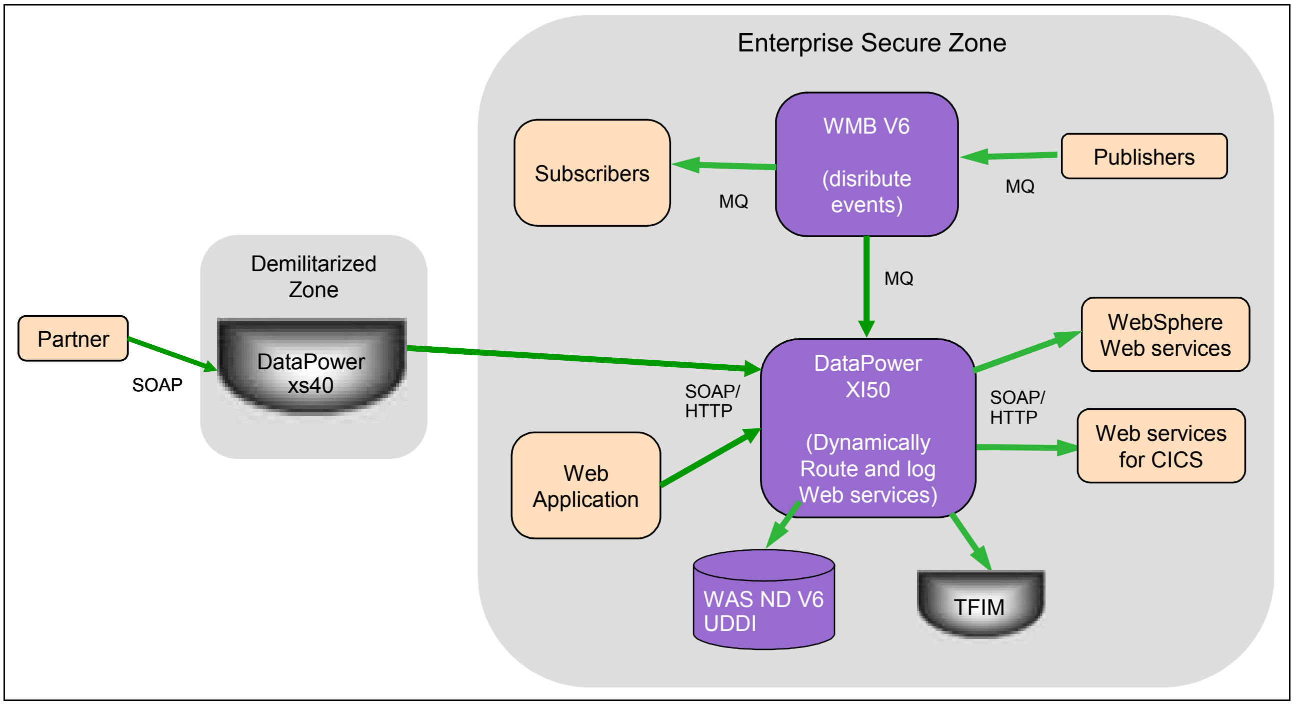
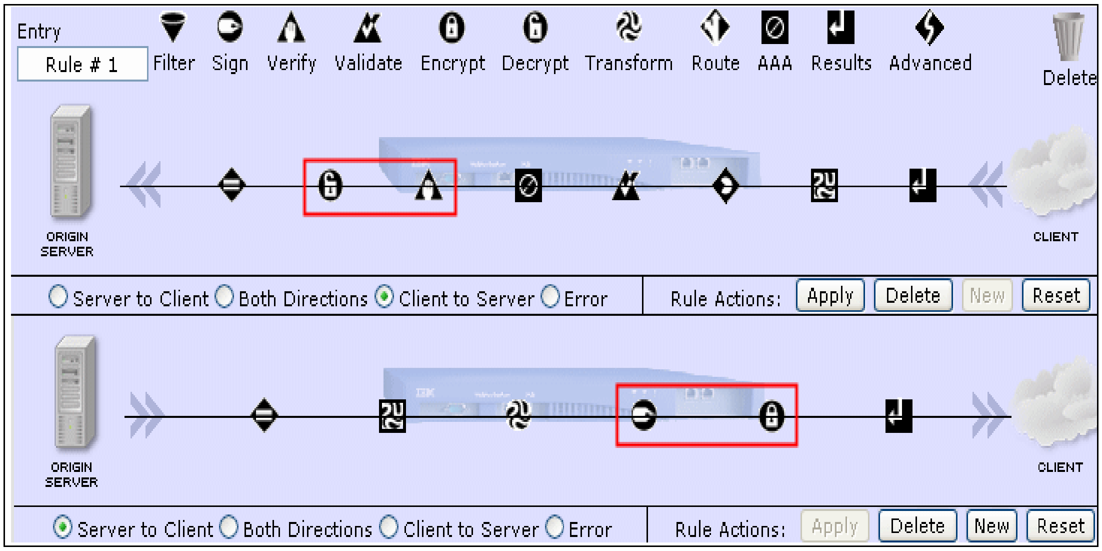


Figure 4-33 Federation ESB scenario with DataPower XS40

This solution shows:

- Using WMB for distribute events, and so on.
- WMB hosting an XML Web service.
- Use MPG or WS-Proxy, preferably.
- All the security infrastructure is based on XS40.

Figure 4-34 illustrates the end-to-end scenario.



Entry
Rule # 1
Filter
Sign
Verify
Validate
Encrypt
Decrypt
Transform
Route
AAA
Results
Advanced
Delete
ORIGIN SERVER
CLIENT
○ Server to Client ○ Both Directions ● Client to Server ○ Error
Rule Actions: Apply Delete New Reset
ORIGIN SERVER
CLIENT
● Server to Client ○ Both Directions ○ Client to Server ○ Error
Rule Actions: Apply Delete New Reset

Figure 4-34 Shows both rules of ITSO_MPGW multi-protocol gateway

In this section, a multiprotocol gateway has been created as the core transformation engine. It will be enhanced to process WS-Sec messages as follows:

1. Request decryption.
2. Request verification.
3. Response encryption.
4. Response signing.
5. Policy enforcement.

   DataPower includes its own AAA info XML file that can be used to store rudimentary access control policies locally. However, it can also call out to external policy servers in order to make authorization decisions.

   > **Recommendation:** DataPower should be used as the policy enforcement point for Web services authorization. It should interact with a central policy decision point, such as Tivoli Access Manager.

6. ESB security solution.

   The ESB security solution addresses the following requirements:

   – Security for service consumers connecting to the broker via HTTPS and MQ

   – Security for broker connections to service providers via HTTPS and MQ

   – Transport-level message protection

   – End-to-end propagation of user IDs in message security context

   – Service-level authorization for ESB integrated with existing TMS's authorization providers such as Siteminder Policy Server and eDirectory LDAP

   – Web services gateway for service consumers connecting to the broker based on WS-Security standard

   – End-to-end identity management

   – Definition of enterprise-wide security roles and security policies

Component architecture for the ESB Security solution is shown in Figure 4-35.

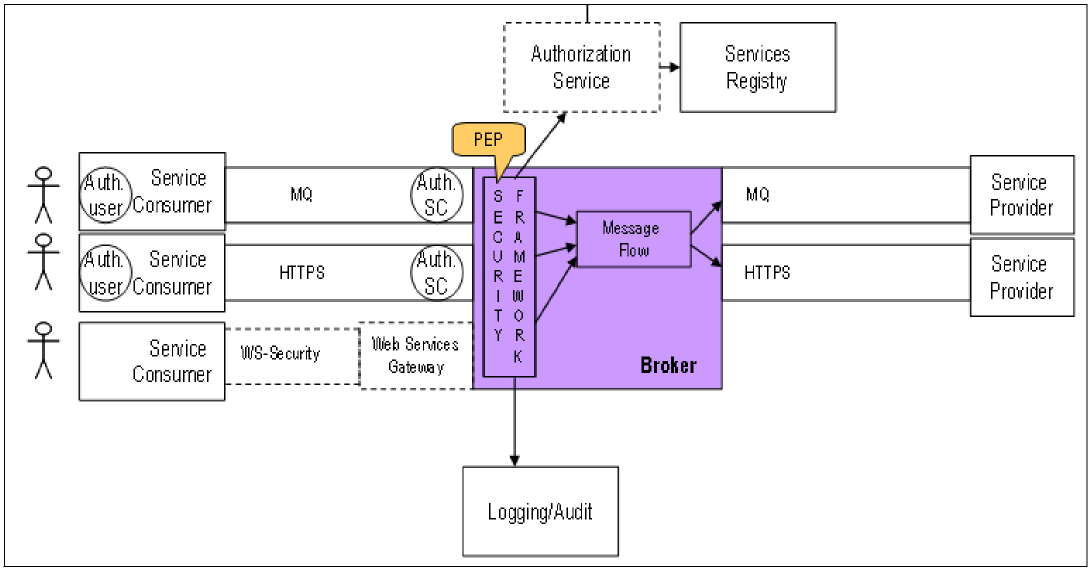


Figure 4-35 Component architecture

7. Security for service consumers connecting to the broker.

   Security for service consumers connecting to the broker is based on an implied trust security model:

   – The ESB trusts all service consumer applications to implement the appropriate user authentication.

   – The user identity is propagated by the service consumer applications to the broker in a trusted fashion.

Based on the architectural decision AD01, the broker will require authentication of service consumers. For phase 3 the broker will use the following transport-level authentication mechanisms for service consumers:

- Two-way SSL for HTTPS connections: Authentication is based on the x509 Certificate presented by the service consumer application during the SSL handshake.
- MQ level security: Identification information included in MQ headers by MQ queue managers used by service consumers to send the message to the broker.

Figure 4-36 pattern offers tangible advantages such as:

- Enhanced WS-*, WS-Security support
- Web services gateway functions
- Protocol transform – HTTP<->MQ
- High performance non-XML and XML transforms

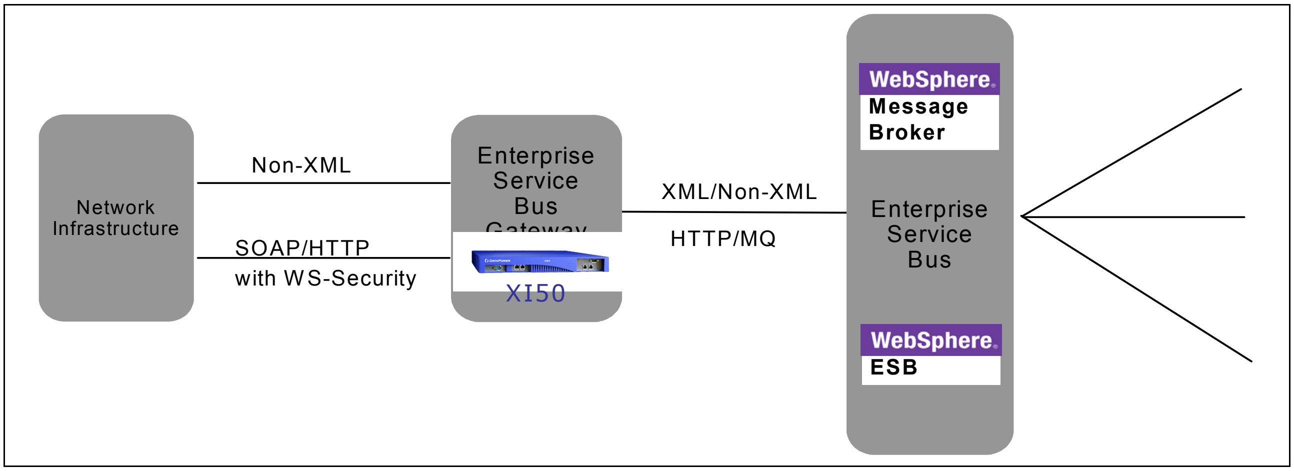


Figure 4-36 ESB pattern

4.7 Summary

WebSphere DataPower SOA Appliances enforce enterprise security policies uniformly on SOA traffic spread throughout the IT environment. DataPower's breadth of functionality combined with its hardened appliance form factor make it an ideal solution for SOA security. This is demonstrated through both its industry leadership and numerous deployments in the most secure IT environments in the world.

The use of the DataPower Appliance provides security-conscious IBM clients with new and powerful options for creating secure systems—in some cases, new options where no feasible option previously existed. The results are:

- The highest levels of Web services security with negligible performance overhead
- The highest levels of SOA standards compliance
- Centralized management and enforcement of security policies

5

Integration patterns

For small to large integration scenarios, IBM WebSphere DataPower appliances can simplify deployments, improve performance, and enhance the security of SOA implementations in the enterprise integration arena. The three product offerings of XA35, XS40, and XI50 provide a broad solution that addresses several integration scenarios. In addition to standalone scenarios, WebSphere DataPower SOA Appliances complement other products—both IBM and third party—to support each stage of the SOA life cycle.

But DataPower can also be implemented in non-SOA architectures and serve as a key component in messaging-based gateways with XML and non-XML binary traffic. Combined with its strengths in security, the XI50 appliance is a very powerful offering for managing messaging flows entering and traversing the enterprise.

5.1 Enterprise service bus implementation patterns and SOA

This section describes technical criteria for selecting an ESB, shows how DataPower can implement an ESB, and then surveys common ESB implementation patterns.

DataPower has excellent coverage for these SOA tenants, as the appliance:

- Supports the abstraction view with its proxy/gateway architecture
- Is highly message oriented, with thorough support for SOAP, raw XML, and unprocessed (binary) documents
- Has full Web services support the correct granularity
- Is network aware, residing between the network and application layer with full support for message-based routing, transformation, protocol mediation, filtering, and extensive security.

DataPower is a premier SOA appliance that is built from the ground up to be a network-aware mediator of future-based messaging paradigms. Its surprisingly small footprint in the enterprise inventory, compared to its value add, is a standards-based success story that is elaborated upon further in this chapter.

5.1.1 DataPower as an ESB

There can be one or many technology components in any one ESB architecture. There can be many ESBs in an enterprise. The justification for how an ESB is constructed may be based on several constraints, such as existing system requirements, leftover infrastructure from acquisitions, or old integration technology not yet sunsetted due to ROI considerations. Regardless, DataPower brings many attributes to an ESB, which provides added value to the enterprise. Of note, DataPower can incrementally provide the following to an enterprise ESB:

- Enhanced throughput, reduced latency for XML processing and security processing.
- The very nature of XML and Web services (for example, SOAP) presents a number of architectural difficulties that ESB products must overcome. While XML provides many advantages over other wire-encoding schemes (its ubiquity and simplicity, for example), it is fundamentally more cumbersome to deal with. Parsing XML is tedious and rapidly becomes a performance bottleneck. Transforming XML from one schema to another is challenging. Similarly, understanding and processing SOAP envelopes requires complex XML processing and is generally accompanied by a tremendous amount of object de-serialization—the creation of in-memory objects that ESB products can understand.
- DataPower appliances are targeted toward specific functional processing focused on XSL processing. DataPower's hardened appliance architecture has the potential to achieve higher performance than traditional general software stacks sitting atop general hardware, in the domain of its core primary mediation services. It leverages purpose-built software, firmware, and hardware to build a synergistic pipelined system centered around processing message packets based on open standards.
- Securing the edge of the ESB via XML firewall and XML threat protection (eligible for DMZ deployment).
- Higher level of security assurance, including DoS protection.
- Advanced Web services gateway functions.
- Remote service management (CLI, signed and encrypted logging, and so on).
- Enhanced WS-* (in particular, WS-Security support).

- Web services gateway functionality (eligible for DMZ deployment).
- Optimized any-to-any transformation, including non-XML via WTX.
- Other integration client support
  - WS MQ
  - Database ODBC interface
  - TIBCO EMS
  - IMS

As such, they differ from traditional software approaches of hub-centric or application server centric integration architectures along these three dimensions. That is, DataPower appliances are a hardened solution, targeted toward specific functional processing.

DataPower provides core ESB functionality, which is to provide a service infrastructure within an organization, and it can also be used to federate ESBs within an enterprise. There are three possible placements of DataPower in an ESB scenario, shown in Figures Figure 5-1 through Figure 5-3 on page 128. Federated ESBs can be serviced through either internal gateway placement or the DMZ.

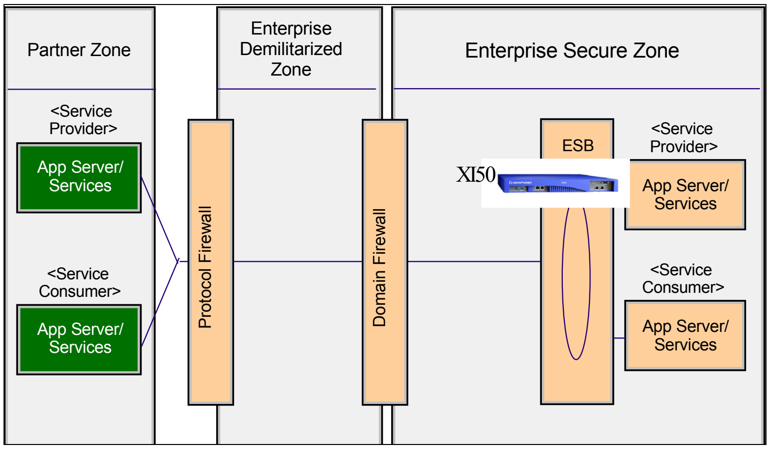


Figure 5-1 Standard ESB

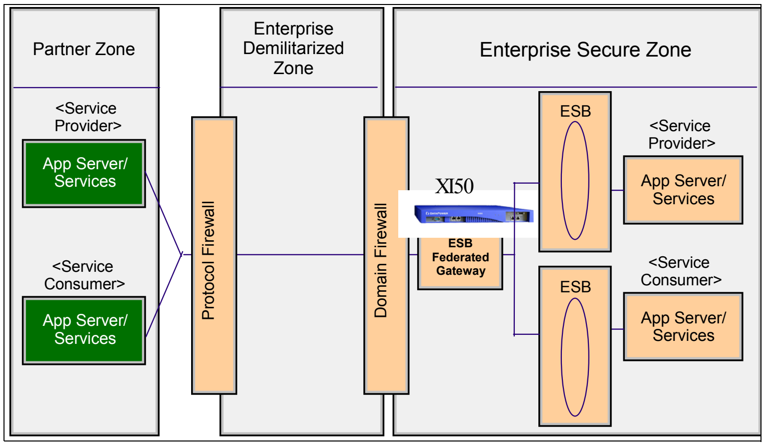


Figure 5-2 ESB federated gateway

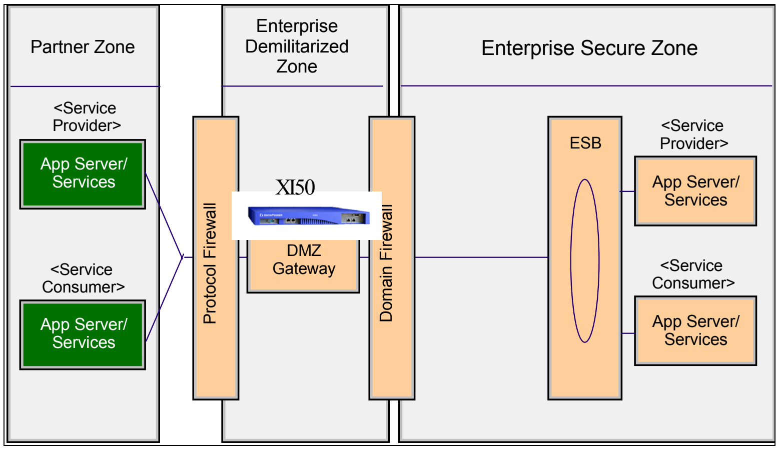


Figure 5-3 DMZ gateway

DataPower can provide fundamental ESB functionality within and at the edge of the ESB. Placement of DataPower in the DMZ would be required to allow for proper credentials around secure access.

5.1.2 Application awareness in the ESB

An application awareness is exploited in the ESB to more intelligently route and process data requests as they traverse the network. That is, the ESB serves as a point of intermediation between client and server or consumer and provider. Such intermediation provides these key benefits:

- Service virtualization
- Improved manageability

  Offloading function into the network enables centralized (and therefore simplified) management of the function and corresponding configuration. For example, style sheets, security, caching, and routing policies can all be centrally managed at service intermediaries versus decentralized across the enterprise server cluster.
- Monitoring and managing messages as they flow on the bus
- Routing messages to destinations on the bus
- Converting protocols to allow service requestors and service providers to communicate
- Transforming messages to allow for different information formats, and for content enrichment and filtering
- Securing messages as they pass on the bus
- Providing connectivity to application services via open standards and proprietary interfaces

The remainder of this section describes each of these functional areas in more detail, listing DataPower's key strengths in each area.

5.1.3 Monitoring and managing

Regardless of what else happens to a message from the time it leaves a service consumer and arrives at a service provider, perhaps the most common requirement for an ESB is to log information about the arrival, path, and content of messages that enter and leave the ESB.

Therefore, an ESB should provide efficient, flexible logging facilities that allow messages to be captured and logged, based on a number of user-defined filters.

ESB general monitoring

Of special interest to enterprise system developers is the ability to track messages as they flow throughout the enterprise, especially to determine where time and resources are being spent in complex scenarios involving several layers of enterprise systems such as SOA appliances, application servers, database servers, and enterprise information systems. Thus, the ability to integrate with enterprise monitoring software through standard protocols such as SNMP and ARM is also a key requirement.

DataPower possesses a variety of monitors, allowing for constant feedback on the processing of messages that flow through the device. Monitors can be configured to generate log messages at a given log level, when count or latency thresholds are reached, or when events are triggered.

General monitoring requirements almost always call for some type of aggregation of runtime statistics associated with messaging flows. DataPower's count monitors can increment a counter every time messages of a particular type pass through a service based on the following message types. The more popular are client IP addresses, request URL, HTTP method/header values, and an unlimited choice of any element/attribute found within the header/body in the packet. A DataPower administrator can fully configure the message types that trigger a count monitor, and can create more than one count monitor for any given service.

A desirable use case for ESB monitoring includes remote management of the DataPower appliances in a production environment. DataPower supports several methods of remote appliance management, including SNMP, WSDM, WS-Management, and proprietary SOAP API.

The ultimate monitoring solution for services-based management as services traverse the ESB is to have support for end-to-end monitoring of the services themselves. For IBM, the industry-leading tool for this requirement is ITCAM for SOA, which can partner with DataPower in services management scenarios.

ESB service level management (SLA) via monitoring

Another management function that is prevalent in ESBs is the ability to shape traffic request patterns. Administrators could, for example, set a policy regarding how they want traffic to be handled within the ESB. Prioritization could place certain requests ahead of others, or requests could be throttled based on service-level policies.

Count monitors can be deployed within DataPower, which can throttle (reject) or shape (delay) traffic when count or latency thresholds are reached or events triggered. Classic use cases where DataPower count monitors can be used to meet SLA requirements include:

- Generation of notifications when the number of requests from any single client reaches a threshold
- Generation of notifications and emission of throttle requests when any single client fails authentication a set number of times within a given period
- Invoking traffic-shaping throttles that delay requests arriving at a rate greater than a set number of times, to maximize the likelihood of successful completion

Duration monitors increment a counter every time that a configured amount of time passes during the processing of messages of a particular type (as defined above). The administrator can fully configure the message types that trigger a duration monitor, and can create more than one duration monitor for any given service. Duration monitors can distinguish the direction of the message. These monitors can be queried through the SOAP interface.

Duration monitors can support the same use cases as count monitors and can shape traffic with delay or throttle requests. Relevant examples are:

- Protecting back-end application resources when application latency reaches a configured threshold
- Emitting shaping traffic events when processing latency reaches a configured threshold, so as to meet service level agreements

5.1.4 Routing

Routing is one of the more complex yet useful facilities offered by an ESB. For most practical purposes, there are two major reasons why a message may need to be routed, both of which an ESB should support:

- Quality of service: Not all services are created equal, nor are all customers. When a service level agreement exists for a service, it is sometimes necessary to prioritize requests based on information about the request, about policies that relate to that request, and about the level of service that is available from the services on the bus.
- Support of specific functionality or affinity: In many cases, there are alternate providers of a service that differ in service version or implementation. Session state may exist on a particular destination, requiring affinity-based routing.

In both cases, there are several ways in which the information comprising the message should be examined, leading us to several other routing capabilities that must be supported by an ESB:

- Content-based routing: Routing based on the actual message body, or on the content of headers specific to the transport or protocol over which the messages are traveling
- Context-based routing: The ability to rely in other sources of information to determine the routing path of messages
- Aggregation and disaggregation: The ability to split or recombine messages

DataPower has strong support for content-based routing and can interrogate either the header or body for routing instructions. Because it leverages the near wire-speed of its XSL processor, its decisions on IP and port decisions are extremely fast. For static instructions, the fundamental object that drives this functionality is the XPath routing map, which is an object found in the OBJECTS/XML processing/XPath routing map. The XPath routing map enables XPath-based forwarding on the part of the device. That is, the selection of a target Web or application server can be based upon the contents of the XML document being processed.

An XPath routing map consists of one or more forwarding rules, arranged in a sequential list. Each forwarding rule consists of an XPath expression accompanied by an IP address-port pair (specifying the destination address) and a binary flag designating the intent to secure the wire with SSL.

With XPath-based forwarding enabled, the device employs user-provided XPath expressions to scan the contents of incoming XML documents. If the document contents match an XPath expression contained in a forwarding rule, the document is forwarded to the specified IP address-port. A candidate document is sequentially evaluated against each of the XPath map's forwarding rules, with the first match providing the forwarding instructions.

Content-based routing is fundamentally dynamic in nature, and follows the proxy pattern where the initial inbound message flow is halted at the device (that is, within the DMZ), a proxy is established, and the flow is re-initialized. It is at this point that the routing instructions are established and IP/port determination is made prior to sending on the proxied message. This routing pattern should not be confused with the static routing capabilities of DataPower in configuring a VLAN.

DataPower's multi-protocol gateway (MPGW) can be configured to heavily leverage the dynamic routing pattern. This gateway can support more than one client (or front-side) protocol, and it can also support more than one back end, server, or protocol. With support of many-to-many protocols and routes, the MPGW will be making decisions on different IP/port destinations that map to the specific protocol that is being converted or maintained. Thus, by default the MPGW will be employed in some fashion in many scenarios that require dynamic routing, particularly when there are requirements for protocol conversion. See Figure 5-4.

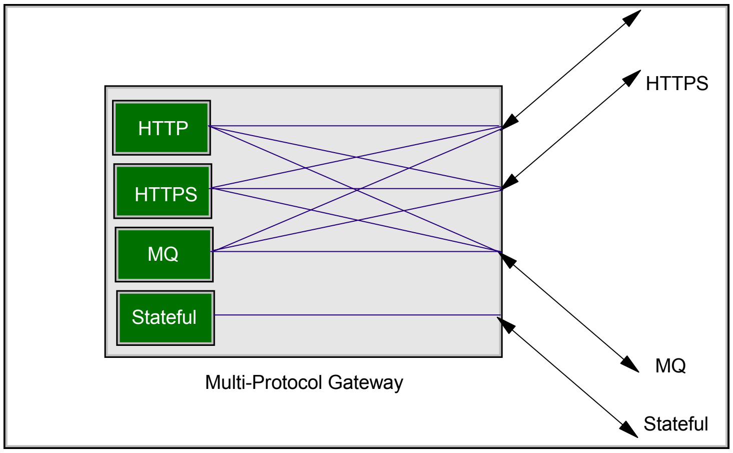


Figure 5-4 Dynamic back-end gateway architecture

5.1.5 Protocol conversion

A common justification for an ESB is the need to move messages from one transport to another. This is commonly referred to as protocol conversion. Its most common example is in moving messages from HTTP to MQ or vice versa. An ESB should not only support common cases of protocol conversion, but should also correctly handle failure cases when dissimilar protocols are used together, such as when a synchronous protocol like HTTP is matched with an asynchronous protocol like MQ.

DataPower is a good choice for managing the interoperability among applications that converse via different protocols. It can act as a proxy to provide controlled access to the ESB. A common use is to expose services to external parties as well as allow internal applications to access external services in a secure and controlled manner. Such services continue to be represented in many protocols, which may not be the same as what is used internally. In fact, for security purposes, any external service request entering a domain should be proxied even if protocol conversion is not a requirement.

For any protocol conversion requirements, ESB architecture or other, the DataPower multi-protocol gateway service (2.3, “Multi-protocol gateway service” on page 15) provides the necessary functionality to handle the protocol conversion (or protocol bridging) and is housed only in the XI50. The MPGW service supports the following client-side and server-side protocols:

HTTP Including GET and POST. POST can contain XML, SOAP, DIME, and SOAP with Attachments.

HTTPS Including GET and POST. POST can contain XML, SOAP, DIME, and SOAP with Attachments.

MQ messages The gateway can use GET and PUT queues to communicate using MQ messages, if licensed only on the XI50.

WebSphere JMS Supports the default WebSphere JMS server.

FTP The gateway can act as either an FTP client, polling a remote FTP server for requests, or as an FTP server, accepting incoming FTP connections.

IMS The gateway can accept incoming IMS protocol requests and can initiate IMS connections on the back side, if licensed only on the XI50.

TIBCO EMS Supports TIBCO Enterprise Messaging Service, if licensed.

NFS The gateway can poll an NFS-mounted directory for the presence of new files and place responses on an NFS-mounted directory.

A typical protocol conversion example for which DataPower XI50 is optimized is a SOAP/HTTP/HTTPS request that comes in and a back-end EJB service provider that is listening and communicating over JMS. See Figure 5-5.

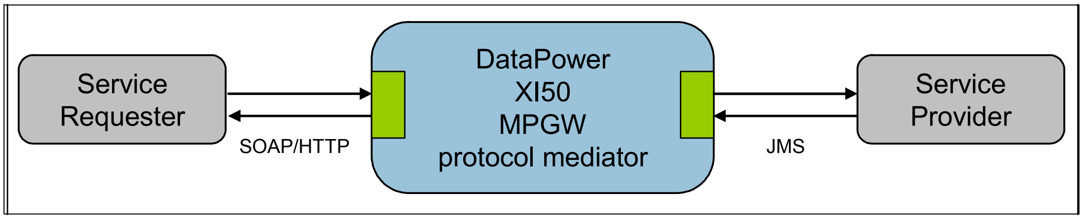


Figure 5-5 DataPower protocol mediation pattern

As the protocol mediator, DataPower must address the situation where HTTP/HTTPS traffic is inbound and has the capability of mediating the following message requests of request/response (asynchronous or synchronous) or fire-and-forget (see “Types of message requests” on page 136). Regardless of message type, DataPower has the ability to become a high-performance HTTP proxy whether the client is in blocking mode or not.

Typically, today’s Web-based scenarios are asynchronous, but back-end systems are transaction oriented and prefer to consume their messages synchronously for transactionality. So there are state management responsibilities for the mediator as well as protocol conversion. A distributed architecture can push these mediation duties to each provider, but it is a more efficient and effective architecture to have a centralized mediation point to provide mediation services and off load this work away from the application servers.

For example, leading J2EE servers such as WebSphere Application Server (WAS) have a mediation engine with WebSphere ESB that has protocol conversion capability. But the mediation engine puts load on the application processors and can degrade performance, particularly when message transformation and translation duties are included. It is application friendly to provide the protocol that each respective application front end prefers while pushing the mediation duties to devices that are dedicated for such services. With DataPower, combining mediation duties with its other security and messaging proxy duties is an extremely efficient architectural decision.

Usage:

- Protocol mediation has support in several of its primary services but in varying degrees based on the service (see product documentation for more detail).
- MPGW: Multiple protocol support is provided in the following:
  - Front Side Handlers
    - HTTP
    - HTTPS
    - FTP Server
    - FTP Poller
    - MQ
    - Raw XML
    - JMS
    - NFS Poller
  - Back-end URLs
    - HTTP/S
    - MQ
    - WAS-JMS
    - TIBCO-EMS
    - FTP
- Limited in XML firewall (HTTP/HTTPS)
- WS-Proxy
  - Endpoint handlers: Front Side Handlers
    - HTTP
    - HTTPS
    - FTP Server
    - FTP Poller
    - MQ
    - Raw XML
    - JMS
    - NFS Poller)
  - Limited back-end choices
- URL-opening a style sheet adds more opportunity
  - HTTP/HTTPS
  - FTP
  - ICAP
  - NFS
  - SMTP
  - SNMP
  - TCP
  - MQ
  - SQL
  - TIBCO-EMS
  - WAS-JMS

5.1.6 Message transformation

A major ESB capability is the ability to transform the content of a message into another format or extend or filter its content based on additional information. There are several different types of message transformations that can be supported by DataPower in the ESB:

- XML-to-XML transformation with XSLT

  XML is now widely used as the markup language that provides metadata around data points contained within a document, which is the message traversing the ESB. For transformation requirements, XML-to-XML translation is the simplest case of transformation, the purest in the sense that proprietary formats are avoided and all standards based, and is the fastest-growing transformation requirement in the industry. XSLT is the language the contains translation instructions for the XML document message, and included is a sub language called XPATH for extracting the XML information. With the advent of schema validation found in XSLT 2.0, XSL engines can now validate that a message complies with a valid XML schema prior to transformation. a valuable capability.

  Any one of the DataPower primary services can be called upon to issue routing and transformation instructions upon receipt of inbound XML document messages. Invariably, if those instructions are based on tagged data found in the message body or header, the DataPower core XSLT engine is employed through the XSL Proxy. So XML parsing is a core, fundamental component of DataPower. In addition, with XSL 2.0 and XPATH 2.0 compliance, it has full schema validation support.

- HTML/SOAP to XML transformations

  In a gateway role, particularly in the DMZ, DataPower is well suited for transforming the lightweight protocols (HTML and SOAP) to XML utilizing the same primary services that leverage the XSL Proxy. For transformation duties, an XSL Proxy can perform any and all of the following:

  – Accept and send SOAP, raw XML, or unprocessed input.
  – Validate and transform XML documents.
  – Process large documents in the streaming mode.
  – Allow, strip, or process attachments (DIME or MIME).

- Binary transformations

  For an ESB to be useful in an enterprise environment it must be able to deal with services that have not been implemented using XML or the latest in WebService-* protocols. The majority of corporate data today exists in traditional format. For the large customers this means mainframe databases. The ability to interface with existing COBOL and RPG programs gives an ESB designer much more flexibility to provide service modernization. This increases the number of service consumers that can take advantage of these programs and data and extends the reach of the ESB further into the enterprise. In the end, this increases overall value of the ESB and can improve time-to-market of new enterprise services.

  DataPower is a device very focused on transformation and translations based on open standard technologies such as XML. But DataPower has been enhanced with WebSphere Transformation Extender (WTX) to provide support for XML-to-binary and binary-to-XML transformation on the appliance without having to write any XSLT. As in the standalone product, writing binary transformations is as simple as using the Design Studio graphical user interface to define inputs and outputs and then map between them. This is discussed in detail in 5.2.1, “WebSphere Transformation Extender (WTX)” on page 145.

- Content enrichment and filtering

  Enterprise modernization often requires that a result from service invocation be enhanced with additional information from other data sources in order to meet the requirements of industry-specific XML schemas. Content enrichment is the ability to invoke several services or database queries and combine the results as a single service response. An equally important function, content filtering, is the removal of extraneous content from a message based on rules and policies.

  The DataPower XI50 provides several avenues to support content enrichment. By itself, XSLT and binary transformation is an inline process that does not directly support content enrichment (it could be achieved but would be convoluted and difficult to build). So we need a way to augment the transformation on the fly. DataPower provides several connectivity services that can provide access to data services that could be the source of enrichment.

  In most scenarios the multi-protocol gateway service is the service selected for content enrichment, as multiple inputs and protocols are required. The connection alternatives are:

  – SQL data object: This is the typical access method used, since databases provide the most relevant data for enrichment.

  – Content enrichment is unique in that almost always *read only* is the typical data access method required. This is because the enrichment task is populating or rewriting a body or header during the inline processing of the document. Data altering functionality at the source is usually not required and not typically desired, so that one does not impede the messages in flight with latency issues associated with a commit transaction. DataPower supports this with a read-only object parameter.

  – Messaging objects, JMS, MQ, EMS: These DataPower objects are also an option, but usually there is too much potential latency in a request/response scenario on a message bus for content-enrichment requirements. Typically, only small runtime increments are allotted for content-enrichment activities. DataPower, with its high-throughput capability, can be caught in time-out scenarios in high-latency request/response scenarios, and the resulting exception processing would have to be treated as a failed request and processing of the message halted.

    See Figure 5-6 for a content-enrichment scenario.

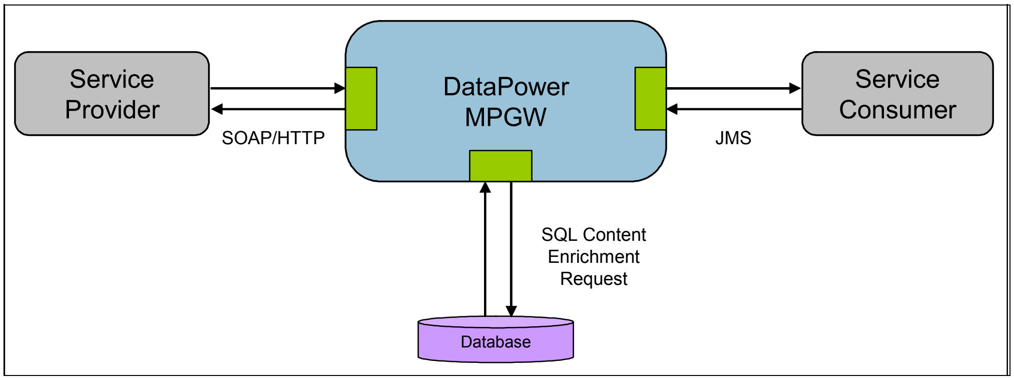


Figure 5-6 Content enrichment

5.1.7 Securing messages

Finally, another critical set of services that an ESB must provide is in securing messages. While the set of security services is quite large, there is a fairly small subset that any ESB should be able to provide:

- Support for standard WS-Security protocols as defined in the WS-I Basic Security Profile
- Support for credential mapping across multiple protocols and transports
- AAA

In many cases, the ESB will need to work with other components of the enterprise architecture in order to perform this task. For instance, if an ESB were to provide authorization, authentication, and audit (AAA) functions on services, it might need to interface with a user registry such as LDAP or to share security tokens with an access management server. Additionally, the ESB could be a logical place to enforce more advance security policies, such as XML denial-of-service protection.

This topic is covered extensively in Chapter 4, “Security” on page 69.

5.1.8 Connectivity

In this section we discuss connectivity.

Types of message requests

An ESB must support a means of connectivity that allow requestors and providers to interact with the bus. In fact, *interacting* with the ESB always happens in the form of message exchange. A requestor must be able to send a request message to the ESB, over one or more standard transport protocols. It must also be able to receive standard response messages from the ESB. Similarly, a service provider must be able to receive request messages from the ESB and send response messages back to it.

The message exchange between the ESB and service requestor and provider can follow several patterns, all of which the ESB must support. These patterns are:

- Request/response (synchronous)

  This pattern is when a client sends a request message to the ESB and blocks until the response comes back. This includes the fact that a response message must be correlated to the proper request message by the ESB, not the client.

- Request/response (asynchronous)

  In this case, the sending of a request and the reception of the respective response are decoupled, at least in terms of time. The client can continue to perform work while the request is being satisfied by the service provider. The response is either actively solicited by the client at a later time, or is sent by the ESB to an asynchronous *response listener* when it is available.

  DataPower’s implementation of request/response is the same whether it is synchronous or asynchronous, as DataPower does not have the capability, nor the need, to block for a synchronous request. It is up to the service provider to have the sufficient functional components to fulfill its handshake duties in a synchronous scenario. This allows for DataPower to focus and behave well in the mediation later with minimal resource requirements. From DataPower’s perspective, any request response is a potentially long-running asynchronous transaction. Is duty in request and response is to correlate the return message, if it ever returns, to the original request message via the correlation ID as specified in WS-Correlation.

- Fire-and-forget

  The requestor sends a message to the ESB and no response is expected. The client can immediately continue to do other work.

- Publish/subscribe

  The pub/sub pattern supports one-to-many relationships between publishers and subscribers. Subscribers indicate their interest in *topics*. Whenever a message is sent to this topic by a publisher, the message is forwarded to all subscribes. This pattern is extremely useful to address many business problems and hence should to be supported by the ESB.

  It requires an additional integration product to help DataPower participate in a pub/sub scenario. DataPower by itself does not provide built-in publish/subscribe functionality. But Datapower can leverage integration products that do have built-in pub/sub capabilities such as WMQ, JMS, and TIBCO, and all of which have client support on the Datapower device.

JMS

Java Message Services (JMS) is Sun's standard API for messaging middleware. JMS is a Java API for accessing messaging middleware. Messaging middleware acts as a broker to provide asynchronous delivery of data between applications. Messaging middleware:

- Acts as an intermediately between the producer and consumer of the message
- Ensures reliable message delivery

Messaging middleware stores, routes and manages messages for delivery. Imagine a courier service where you drop off a package for delivery. That company ensures that your package will be delivered by a certain time. If the other party is not available, it will hold the package for later delivery or pickup. This is essentially how messages are routed through messaging middleware's services such as JMS.

The application that produces the messages is known as producer, and the application receiving the messages is known as consumer (Figure 5-7).

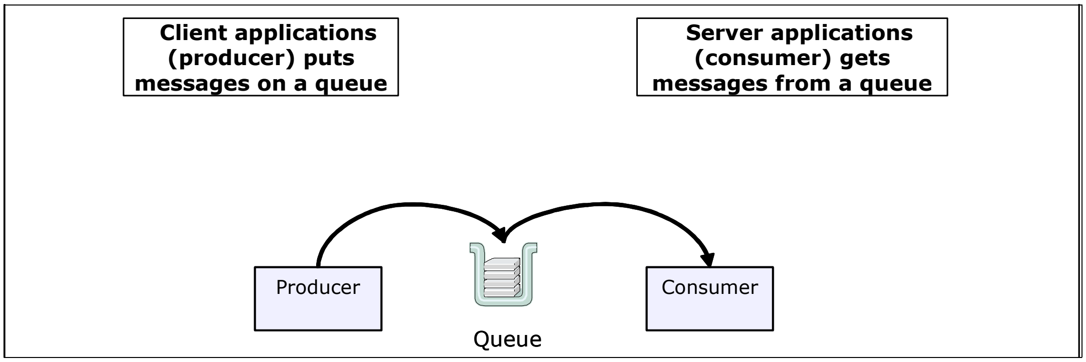

Client applications (producer) puts messages on a queue
Server applications (consumer) gets messages from a queue
Producer
Queue
Consumer

Figure 5-7 Queuing messages with middleware

DataPower's implementation of JMS

A messaging middleware that supports JMS is known as a JMS provider. Messaging middleware acts as a broker to provide asynchronous delivery of data between applications. The application that produces the messages is known as producer, and the application receiving the messages is known as consumer.

Since the DataPower appliance does not contain a JVM (usual technology container for a JMS service request), it cannot exchange messages originating from a JVM with the JMS provider in WebSphere Application Server using JMS. Instead, it uses the IBM JetStream Formats and Protocols (JFAP) to communicate with the service integration bus. JFAP is conveniently used by the default JMS provider in WebSphere Application Server. With JFAP support via a WebSphere JMS object, a multi-protocol gateway can provide default JMS capabilities both as a client-facing and a server-facing messaging service (Figure 5-8).

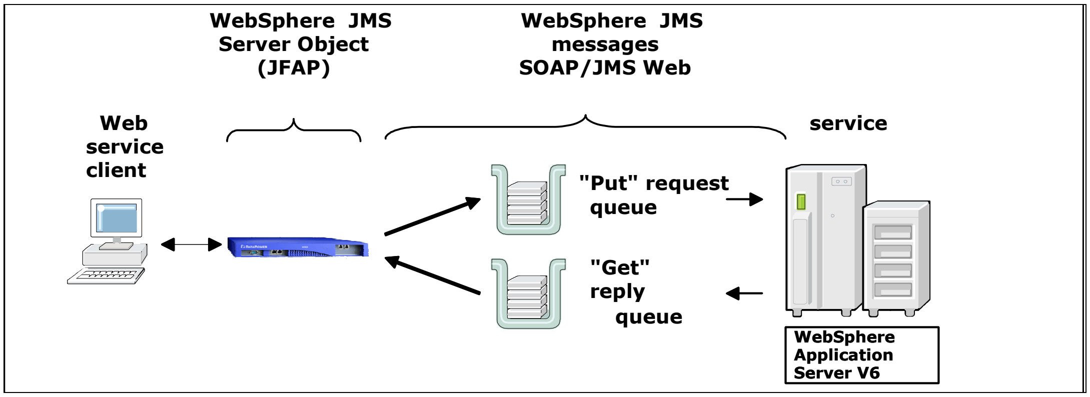


Figure 5-8 JFAP to WAS V6

WebSphere JMS server object

The DataPower WebSphere JMS server object enables the multi-protocol gateway service to connect to the default messaging provider in WebSphere Application Server (WAS) V6.

Execute the following required steps to enable JMS messaging from DataPower to the default messaging provider in WebSphere Application Server V6:

1. Define the WebSphere JMS endpoint. connect to the default messaging provider in WebSphere Application Server V6.
2. Enter the name of the service integration bus.

   A service integration bus (SIB) supports applications using message-based and service-oriented architectures. A bus is a group of interconnected servers and clusters that have been added as members of the bus. Applications connect to a bus at one of the messaging engines associated with its bus members.

   A messaging engine is a component, running inside a server, that manages messaging resources for a bus member. Applications are connected to a messaging engine when accessing a SIB.

   Applications (such as the WebSphere JMS object) running outside the WAS environment cannot locate directly a suitable messaging engine to connect to the target bus. In such cases the remote clients or servers must access the bus through a bootstrap server that is a member of the target bus.

A bootstrap server is an application server running the SIB process, but need not be running any message engines. Rather, the bootstrap server selects a messaging engine that is running in an application server that supports the bootstrap protocol requested by the remote device (Figure 5-9).

To connect to a messaging engine, the remote application first connects to a bootstrap server. The bootstrap server selects a messaging engine and then tells the client application to connect to that message engine to gain bus access.

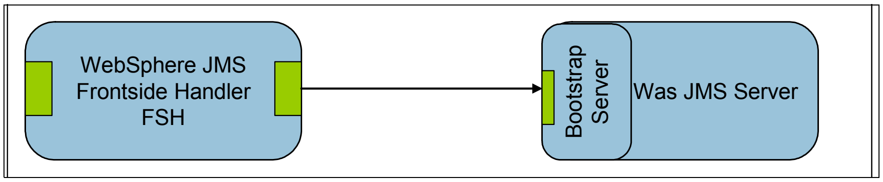


Figure 5-9 Outside a WAS environment

3. Optionally, configure security (SSL, user name/password), transactionality, connection, and memory settings.

SQL data source

A DataPower SQL data source object provides the necessary configuration values to establish a direct connection to a data server. When configured, it is possible to dynamically perform database operations (such as SELECT and INSERT) on the remote database instance. This instance is identified over the network by IP/domain and port. The information retrieved (SELECT) can then be used in a processing policy, or conversely, information obtained during the processing of a message can be stored (INSERT/UPDATE/DELETE) in the configured database instance.

Stored procedure calls are supported. The driver technology is ODBC, not JDBC™. This is because, for security reasons, Java technology is not allowed on any DataPower device. Since both ODBC and JDBC support the same SQL query syntax, implementation is the same.

So why is an SQL data source needed for message handling? A SQL data source was added to XI50 to do (at a minimum) the following:

- Support content enrichment, where a mediation of an in-flight message requires enhancement to the document in order to make decisions such as routing determination, or to improve the contents of the document before passing it on to the consumer.
- Act as a proxy to a final database destination. DataPower can therefore fulfill its connectivity duties in the ESB and act as the adapter to an endpoint, ending the transaction after transforming from, for example, a lightweight protocol SOAP instruction.
- Submit a query request to a database provider and respond back with database results to the requestor. For example, an order processor can submit a request to a Web Proxy service and the service can transform and submit the request to a database provider via a SQL data source object, and return the database response in HTML back to the requestor.

Database sources supported are:

- DB2®
- Oracle
- MS-SQL Server
- Sybase

DataPower and DB2 pureXML

Data enrichment is a classic scenario for a simple use case for leveraging the SQL data source. Actually, the client in this scenario acts as a bridge between two disparate technologies: XML objects and relational data persistence. As long as our use cases are relatively simple, this client interface works. But we are in a transition phase where relational data persistence is at odds with the open standards XML messaging infrastructure now being introduced by the Web, Web services, and ESB. *Shredding* XML data (mapping the elements and attributes of an XML document to columns in one or more tables) loses its gains when XML documents contain heavily nested parent/child relationships and irregular structures.

IBM introduced the pureXML™ feature in DB2 9 that allows XML data to be stored natively in the database. It makes XML a first-class data type in DB2, and a column of type XML can store any well-formed XML in its native hierarchical form within the database. Once the data is stored in the database as XML, the data can be queried very efficiently with XQuery, SQL, or several other DB2-supplied application development interfaces that DB2 offers.

To leverage the capabilities of DB2 pureXML, we need a different architecture approach that allows us to merge the capabilities provided by DataPower and DB2 pureXML in a single solution. In this scenario, we would like to go beyond the simple use case of data enrichment and instead receive complex XML data from a requester and submit it to DB2 9 in its native XML form. With this scenario, WebSphere DataPower can complement DB2 pureXML solutions by offloading XML validation and transformation.

A major benefit of this scenario is that the WebSphere DataPower Appliance performs all validation steps with appropriate error handling. It also oversees the insertion of the XML document into the DB2 pureXML database, off-loading the validation steps from the database processor. The insertion is performed only if the document has successfully passed all validation steps.

What we would also like to do is validate against industry standard formats, such as Transaction Workflow Innovation Standards Team (TWIST) or Universal Financial Industry message scheme (UNIFI)—ISO 20022—using predefined XML stylesheets. Industry formats are an important part of standardized information exchange between different information systems across the industry including health care, insurance, and finance. These formats are based on XML. XML Schema defines the structure of documents, to which all derived documents must comply. Even though DB2 pureXML is capable of XML Schema registration, XML document validation, and XML stylesheet transformations, this is the domain where DataPower excels with its dedicated hardware-based XML parsing engine. Another nice feature is that since DataPower is intersecting the transaction in a gateway pattern, it can perform routing and security at the same time, greatly optimizing the solution by accomplishing all of these functions at the same time and at near wirespeed.

To facilitate the schema validation process, we introduce Schematron, a declarative validation language that enables the checking and cross-checking of XML content through the specification of rules in XPath, and of custom error messages should the rules fail. This language allows us to dynamically create intermediate XSL stylesheets based on simple rule language constructs. After Schematron rules are defined in an XML format, the *rules document* is transformed into an intermediate XSL stylesheet using the Schematron XSL stylesheet. The resulting new XSL stylesheet is then applied to each XML document and will, if the content is not as expected, produce custom messages in the exception process. Figure 5-10 shows the architecture design for this scenario.

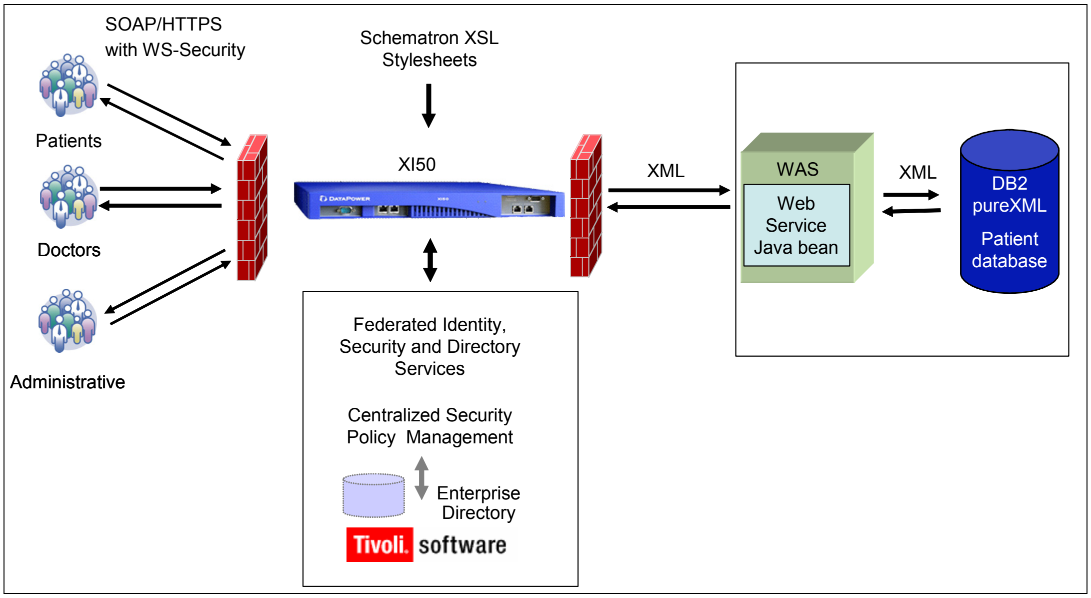


Figure 5-10 DataPower to DB2 pureXML

Step 1: XML schema, XML documents, and Schematron

The steps are:

1. Define an industry format for the solution. This can be any industry format based on XML. The industry bundles for pureXML, published on alphaWorks®, illustrate access to content stored on IBM DB2 in XML, through small script or Java-based applications. They are focused on populating, validating, and querying XML content that is pertinent to a specific industry.
2. Download the industry-specific schemas from IBM alphaWorks and identify the desired schemas.
3. Enhance the industry specific schemas validation rules with Schematron. Create declarative-based rules, checking and cross-checking XML content through the specification of rules in XPath, and developing custom error messages should the rules fail. See Example 5-1.

Example 5-1 Schematron implementation (simple.sch)

```
<?xml version="1.0" encoding="utf-8"?>
<schema xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>Simple Schematron Validation Example</title>
  <pattern name="Personal Information">
    <rule context="/person/name/first">
      <report test="text() = 'gerard'">
        First name must not be 'gerard'!
      </report>
    </rule>
  </pattern>
</schema>
```

The above example looks up the first name in an XML document, checks whether the first name equals *gerard*, and prints a validation failure message if it does. In case of validation failure, the message would be `First name must not be 'gerard'!`.

4. Sample XML documents are created that are valid and invalid with respect to validation against the XML schema and test the Schematron rule defined above.

Step 2: DB2 pureXML database and Data Web services

The steps are:

1. Create a database, schema, and a table that has a XML record type.
2. Create a stored procedure that inserts an xmlRecord.
3. Create a data Web service that exposes the stored procedure. The Web service can have either a SOAP-based or REST-based interface.

   This scenario generates a Web service using a simple Java class to insert and retrieve XML data into and from DB2 9 using the pureXML feature. Rational Application Developer (RAD) is the used design tool. This Web service can be consumed by any Web services client that can make SOAP over HTTP Web service calls (in this case DataPower).
4. For this scenario, a bottom-up approach is used to create a Java bean that contains data access and manipulation methods, for manipulating data in an XML column of the DB2 9 sample database. The Java methods are then exposed in the Java bean as Web services using the RAD tooling.

Step 3: WebSphere DataPower SOA Appliance

The steps are:

1. Create a DataPower XML firewall service that is configured for XML schema validation. XML documents being sent to this policy on the DataPower Appliance are validated against the XML schemas created in the previous steps.
2. Apply the Schematron XSL stylesheet to the incoming request XML document. If the Schematron stylesheet action produced an error message, the error message needs to be sent back to the client that initially sent the request XML document.
3. If there was no error, the DataPower Appliance should forward the XML document to the DB2 pureXML Data Web Service, which will then insert the valid XML document into the database.
4. It is also important to know whether the insert into the DB2 pureXML database through the Data Web Service was successful. Therefore, the DataPower XML firewall will forward the response message from the DB2 Data Web Service to the client that initially sent the request XML document to the DataPower Appliance, indicating whether the insertion operation was successful.

Summary

This section showed how DB2 pureXML and the WebSphere DataPower SOA Appliance can compliment each other to realize powerful applications, where the WebSphere DataPower appliance performs XML validation, and the DB2 pureXML database manages the XML storage, indexing, and querying. Both XML structure validation (through XML schema) and content validation (through Schematron) have been described. The combination of the two products, WebSphere DataPower and DB2 pureXML, provides flexible and speedy access to validated XML documents.

Note: This scenario is based on work found in the article “XML schema and content validation using DataPower and DB2 pureXML,” which provides a detailed overview on how the scenario is set up, including a sample XML schema, sample XML documents, a Schematron example, a DB2 pureXML database, a data Web service, and the configuration of a WebSphere DataPower SOA Appliance.

http://www.ibm.com/developerworks/db2/library/techarticle/dm-0805malaika3/index.html?ca=drs-#resources

Resources

For further reading see:

- XML schema and content validation using DataPower and DB2 pureXML:

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0805malaika3/index.html?ca=drs-#resources

- *DB2 9: pureXML Overview and Fast Start*, SG24-7298:

  http://www.redbooks;.ibm.com/abstracts/SG247298.html?Open

- Schematron Web site:

  http://www.schematron.com/overview.html

- Generate Web services for DB2 9 pureXML:

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0706bommireddipalli/

- Get started with Industry Formats and Services with pureXML:

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0705malaika/

- Industry Formats and Services with pureXML:

  http://www.alphaworks.ibm.com/tech/purexml

TIBCO EMS

The TIBCO EMS server is analogous to a WebSphere MQ queue manager in that it manages PUT and GET queues. The TIBCO EMS server provides messaging services for communicating applications by monitoring queues, by ensuring that sent messages are directed to the correct receive queue, or that messages are routed to another queue manager.

With the TIBCO EMS connector object, DataPower can act as a gateway to another bus technology that resides in the ESB. There is no rule that says that any ESB must use only one bus technology. In reality, larger enterprises will quite possibly possess more than one bus technology that has been procured in different budget cycles or that was obtained in an acquisition with the resulting merging of IT infrastructures. With gateway capability to TIBCO EMS as well as to WebSphere MQ, DataPower is particularly strong when acting as a proxy to multiple ESBs. This is known as a Federated ESB topology.

TIBCO’s EMS is JMS compliant. This simplifies the task of DataPower in a federated gateway role, particularly as the XI50 has a JMS connector object. For more information see “JMS” on page 137.

Note that TIBCO Rendezvous (RV) is not supported. TIBCO RV is an older bus technology that is still used today in heritage solutions. The supported architecture for this scenario is to connect a TIBCO EMS provider with an RV request, and the DataPower EMS client would connect to that EMS provider.

For all of these scenarios, refer to Figure 5-11. Note that the XI50 can also reside in the DMZ.

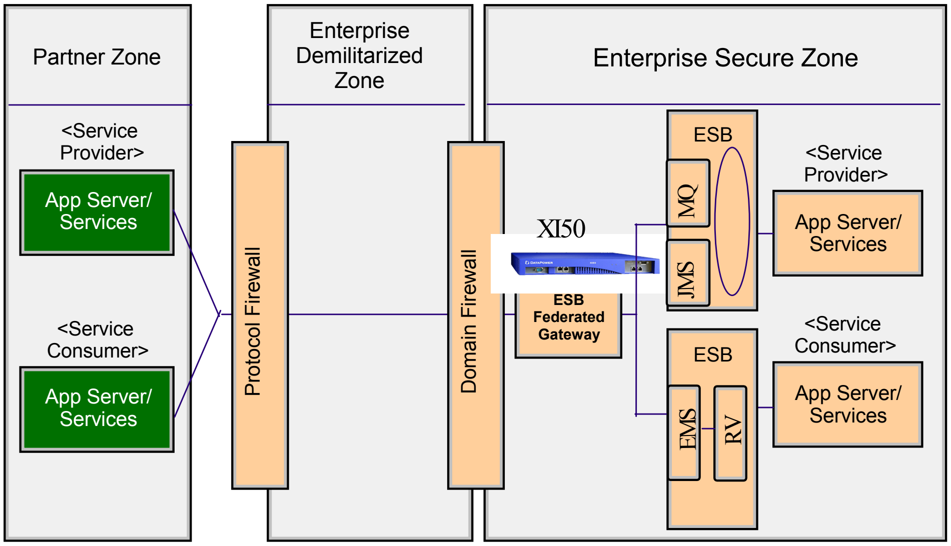


Figure 5-11 TIBCO EMS/RV/JMS federated gateway

5.1.9 Summary

Enterprise service bus is an architectural concept that is well supported by the IBM product portfolio including DataPower. These products will continue to be enhanced to meet the evolving requirements of the ESB. Using an ESB will become a critical system resource for many that will provide the availability, reliability, manageability, and security necessary for enterprise integration systems.

DataPower possesses all the major services that are required in order to implement an ESB:

- Transformation
- Routing
- Validation
- Authentication
- Logging
- Protocol bridging

When some connectivity is required outside of WMQ, JMS, EMS, or SQL, then the solution can be augmented with such products as WESB and WMB that contain full suites of IBM connector technology. As an appliance, hardware and software are contained in one solution. When the performance capabilities are considered, the price performance of DataPower as an ESB solution is hard to beat.

5.1.10 Resources

For further information see:

- “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus”

  http://www.ibm.com/developerworks/library/ws-soa-eda-esb/

- “Enterprise Service Bus implementation patterns”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html

- *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346

  http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf

- “Security Patterns within a Service-Oriented Architecture”

  http://www.ebizq.net/topics/soa/features/6554.html?&pp=1

- “Make SOA real with IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower SOA Appliance”

  http://www.ibm.com/developerworks/webservices/library/ws-soa-real1/

- “BPEL or ESB: Which should you use?”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0803_fasbinder2/0803_fasbinder2.html

- Exploring the Enterprise Service Bus, Part 2: Why the ESB is a fundamental part of SOA

  http://www.ibm.com/developerworks/library/ar-esbpat2/

- “Make SOA real with IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower SOA Appliances” series

  http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search_by=Make+SOA+real+with+IBM+WebSphere+Enterprise+Service+Bus+and+IBM+WebSphere+DataPower+SOA+Appliances

- eWeb Services ArchitectureW3C Working Group Note 11, February 2004

  http://www.w3.org/TR/ws-arch/#service_oriented_architecture

5.2 Integration to the heritage applications

In this section we discuss traditional applications that have run on mainframes for years and will run for many more years. These are also known as traditional, heritage, or mainframe applications.

5.2.1 WebSphere Transformation Extender (WTX)

IBM WebSphere Transformation Extender is the IBM universal transformation engine. In a graphical environment, called the WebSphere Transformation Extender Design Studio, you can easily describe your data and map it between multiple inputs and outputs. Once the maps are defined, you can run them in a variety of different methods, including batch mode. APIs are available in a variety of languages, via a Web Service call, on z/OS®, and in many other ways. Additionally, the WebSphere Transformation Extender runtime engine is tightly integrated with a variety of other IBM products, such as WebSphere Message Broker, WebSphere Enterprise Service Bus, and WebSphere Process Server.

WebSphere DataPower appliances provide an array of functionality in purpose-built and easily deployable devices. Several versions of the DataPower Appliances provide different levels of functionality. DataPower XML Accelerator XA 35 provides XML, schemas, XPath, and XSLT processing. DataPower XML Security Gateway XS40 adds firewall, encryption, and access control. DataPower Integration Appliance XI50 further adds ESB functionality, including message routing and transformation.

Using WebSphere Transformation Extender Design Studio to develop transformations for DataPower Appliances is a natural extension of the platform. DataPower appliances support high-performance XML-to-XML transformation on the device using XSLT. WebSphere Transformation Extender provides support for XML-to-binary and binary-to-XML transformation on the appliance without having to write any XSLT. As in the standalone product, writing transformations is as simple as using the Design Studio graphical user interface to define inputs and outputs and then map between them.

Any-to-any transformation

DataPower is a device very focused on transformation and translations based on open standard technologies such as XML. But there are several scenarios, including heritage and B2B solutions, where XML is not the standard data format. Over time this will change as the landscape evolves and accepts open standard protocols, which are beginning to accelerate. However, even today most B2B integration today is done over non-XML protocols, such as EDI.

To address this traditional requirement and others, the XI50 has been augmented with tight integration to WebSphere Transformation Extender (WTX). With WTX, XI50 can parse and transform arbitrary binary, flat text, and other market-based protocols. WTX is a powerful, transaction-oriented, data integration solution that automates the transformation of high-volume, complex transactions without the need for hand coding. This provides enterprises with a quick return on investment. This product supports EDI, XML, SWIFT, HIPAA, and other standards-based B2B integration. It also supports the real-time integration of data from multiple applications, databases, messaging middleware, and communications technologies across the enterprise. WTX, in the context of DataPower:

- Enables integration developers to visualize and transform complex data without coding.
- Handles large, semi-structured, hierarchical, and nested data such as EDI documents.
- Provides a library of ready-to-use, frequently required functions. It supports a wide variety of industry-standard exchange formats featuring financial services and health care.
- Integrates with IBM WebSphere Enterprise Service Bus offerings, DataPower, WESB, and WMB.
- Connects with a range of file-based and message-based transport protocols including WebSphere MQ.
- Runs stand-alone in message and event-driven IT architectures, or embedded with applications and J2EE servers.
- Runs on the IBM z/OS platform and works as an excellent companion to WebSphere MQ, DataPower, WebSphere Message Broker, and WebSphere Enterprise Service Bus on that platform.
- Provides a universal transformation engine for SOA.
- Provides a unified experience with a transformation and mapping IDE for DataPower, WBM, and WESB. It needs to learn only one tool for mapping and transformation.

The DataPower xformbin policy action transforms a non-XML document, such as binary or flat text, using processing control files.

WTX is composed of several components. Of interest to DataPower are:

- Design Studio: Design time tool that models and tests data mappings
- Type Designer: Defines data structures (type trees)
- Map Designer: Defines and tests data mappings between structures
- Launcher: runtime Engine that supports multiple deployment environments

DataPower uses a patented technology called DataGlue that enables the use of externally generated mappings to perform a binary transformation onboard DataPower (Figure 5-12).

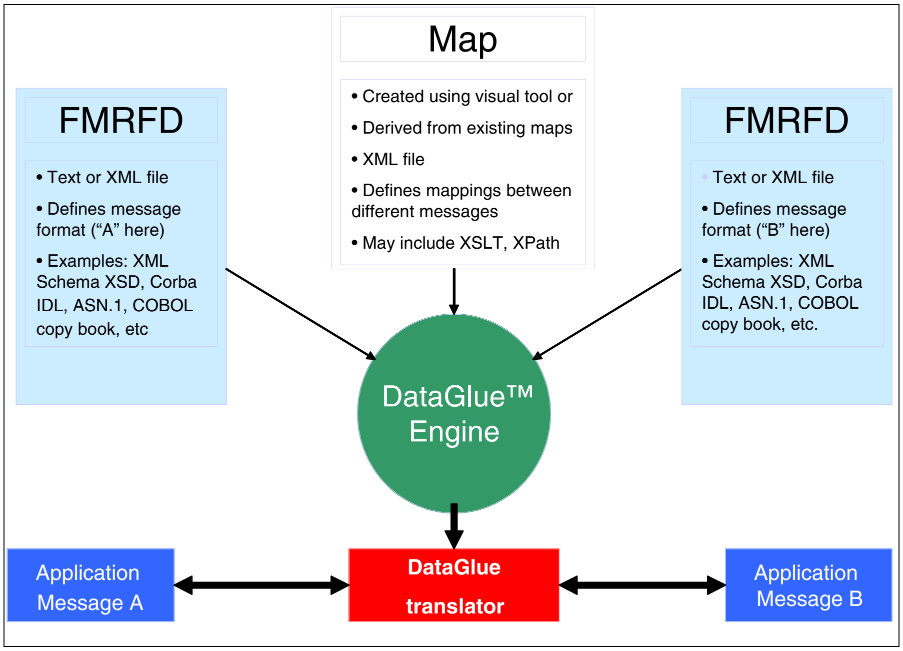

Map
• Created using visual tool or
• Derived from existing maps
• XML file
• Defines mappings between different messages
• May include XSLT, XPath
FMRFD
• Text or XML file
• Defines message format ("A" here)
• Examples: XML Schema XSD, Corba IDL, ASN.1, COBOL copy book, etc
FMRFD
• Text or XML file
• Defines message format ("B" here)
• Examples: XML Schema XSD, Corba IDL, ASN.1, COBOL copy book, etc.
DataGlue™ Engine
Application Message A
DataGlue translator
Application Message B

Figure 5-12 DataGlue format descriptors

WebSphere Transformation Extender Design Studio

WTX Design Studio now has the capability to generate data mapping and translation files meant for DataGlue engine consumption. Creating transformations using WebSphere Transformation Extender Design Studio is a simple process. A primary benefit of using this product is that understanding the process of creating transformations in the standalone version of WebSphere Transformation Extender lets you leverage those transformations to create transformations targeted for DataPower.

Creating type trees

The first step in creating a transformation is to define a type tree, which is a WebSphere Transformation Extender artifact that is the data dictionary for what you want to transform. It is a schema-like representation of the data that enables WebSphere Transformation Extender to perform validation on inputs and outputs.

There are various ways to create type trees, depending on what your data is. WebSphere Transformation Extender provides a set of importers that automate this process, as explained below. These importers include XML schemas, DTDs, COBOL copybooks, CORBA IDL files, and others. If you want to define your input data yourself, you can use the Type Designer graphical tool, which lets you specify what fields your data contains, how it is delimited, and any type of validation that you want to perform.

Type trees are not tied to an input or output direction and can be reused extensively. They simply tell WebSphere Transformation Extender what type of data to expect, to ensure that everything that you transform is valid.

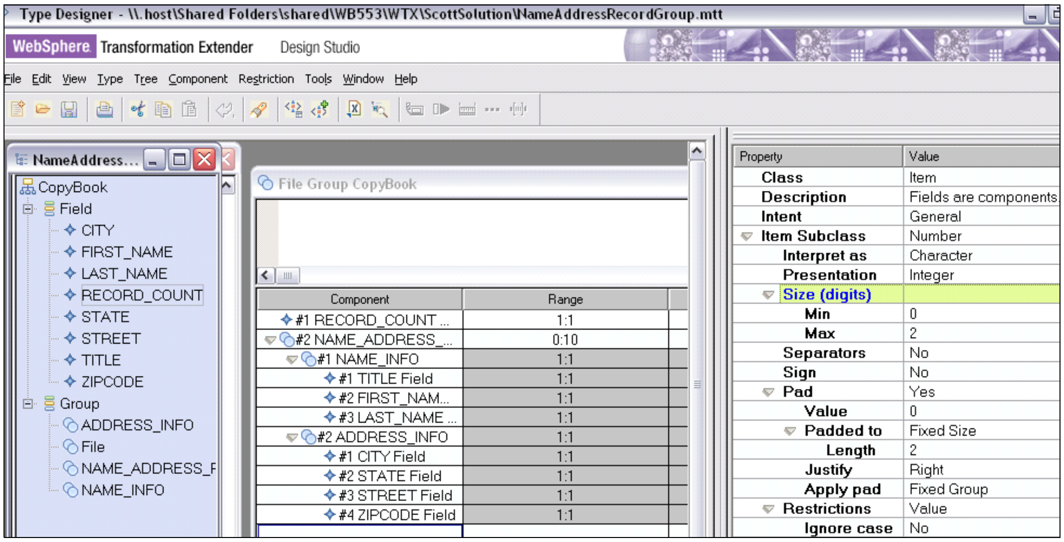
Creating maps

Once you have defined all of the type trees that you are transforming, it is time to start mapping. WebSphere Transformation Extender Design Studio provides a graphical tool called Map Designer that makes this process simple. First, create a map source file with a.mms extension, which is a physical artifact that contains all the mappings that you might perform. Next create an individual map. Map source files can contain many maps, but in this chapter we limit it to one.

Each map has input and output cards. WebSphere Transformation Extender allows multiple inputs and outputs, so you can read from various sources and write out to various places within the context of a single map. When creating an input card, the most important setting is the card name. After you define your input and output cards, mapping is as easy as dragging from an input to an output field. To perform more complex functions, WebSphere Transformation Extender provides built-in functions for tasks such as converting your data, performing mathematical operations on it, or manipulating strings. When you have mapped all the output fields, you are ready to build your map and run it, which you do from within the Map Designer tool.

For the following items refer to Figure 5-13.

- The panel on the far left shows the individual fields and any groupings definitions of a type tree.
- The center panel shows the sequenced structure of a type tree. At the highest level is the RECORD_COUNT and a record. The RECORD_COUNT specifies how many occurrences of record exists. The record can be composed of sub-structures composed of individual fields.
- Translation rules are also stored.
- Type trees can be analyzed for logic and structure consistency.
- Type trees are in MTT files, which are in a binary format.
- Type tree scripts are in MTS files. An MTS file is a XML-formatted representation of the MTT file, and is the format that DataPower understands.


Type Designer - \\.host\Shared Folders\shared\WB553\WTX\ScottSolution\NameAddressRecordGroup.mtt
WebSphere Transformation Extender
Design Studio
File
Edit
View
Type
Tree
Component
Restriction
Tools
Window
Help
NameAddress...
File Group CopyBook

CopyBook
- Field
  - CITY
  - FIRST_NAME
  - LAST_NAME
  - RECORD_COUNT
  - STATE
  - STREET
  - TITLE
  - ZIPCODE
- Group
  - ADDRESS_INFO
  - File
  - NAME_ADDRESS_F
  - NAME_INFO

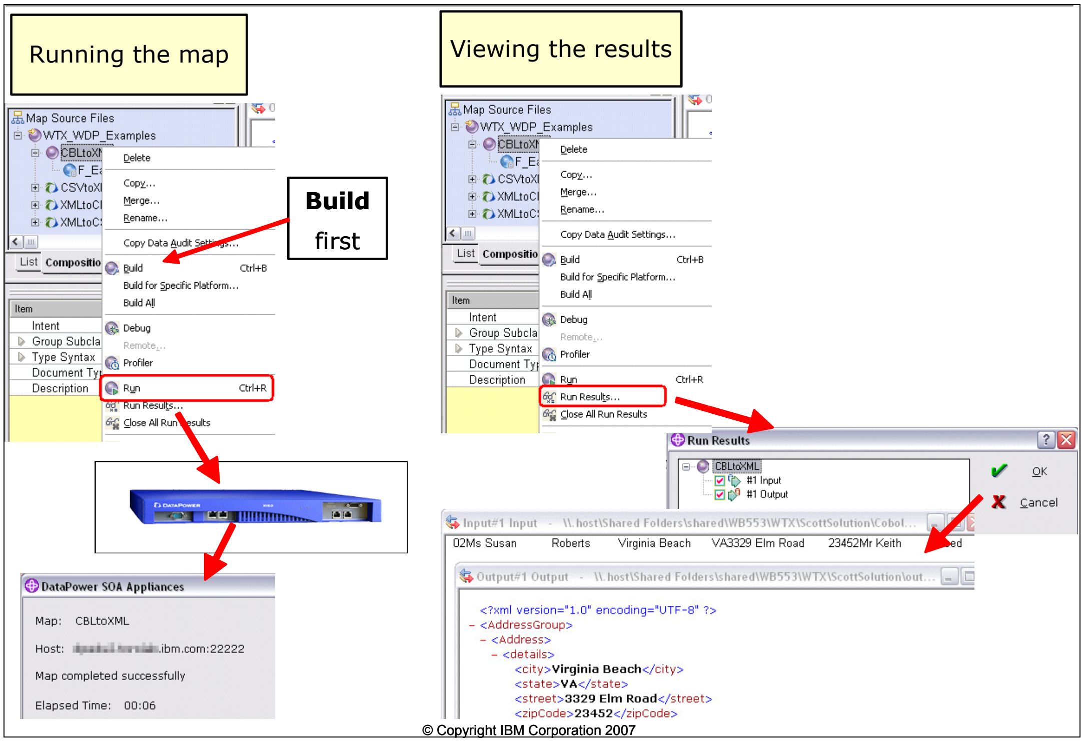
Component	Range
#1 RECORD_COUNT ...	1:1
#2 NAME_ADDRESS_...	0:10
#1 NAME_INFO	1:1
#1 TITLE Field	1:1
#2 FIRST_NAM...	1:1
#3 LAST_NAME ...	1:1
#2 ADDRESS_INFO	1:1
#1 CITY Field	1:1
#2 STATE Field	1:1
#3 STREET Field	1:1
#4 ZIPCODE Field	1:1

Property	Value
Class	Item
Description	Fields are components
Intent	General
Item Subclass	Number
Interpret as	Character
Presentation	Integer
Size (digits)	
Min	0
Max	2
Separators	No
Sign	No
Pad	Yes
Value	0
Padded to	Fixed Size
Length	2
Justify	Right
Apply pad	Fixed Group
Restrictions	Value
Ignore case	No

Figure 5-13 WTX Design Studio: Type Designer

For the following items, refer to Figure 5-14.

1. A *build* of the map must occur before the run is executed. For a DataPower run time, this produces the MTS files (XML-formatted files) needed by DataPower from the MTT files (binary files) used by WTX.
2. To view the results of the test, select **Run Results**. It then prompts for which input/output cards you actually want to see. In the area near the in and out cards, the selected input and output cards appear. In this case, the input shows the COBOL stream, and the output shows the resulting XML structure.



Map: CBLtoXML

Host: [redacted].ibm.com:22222

Map completed successfully

Elapsed Time: 00:06

```
02Ms Susan        Roberts      Virginia Beach    VA3329 Elm Road      23452Mr Keith        ed
```

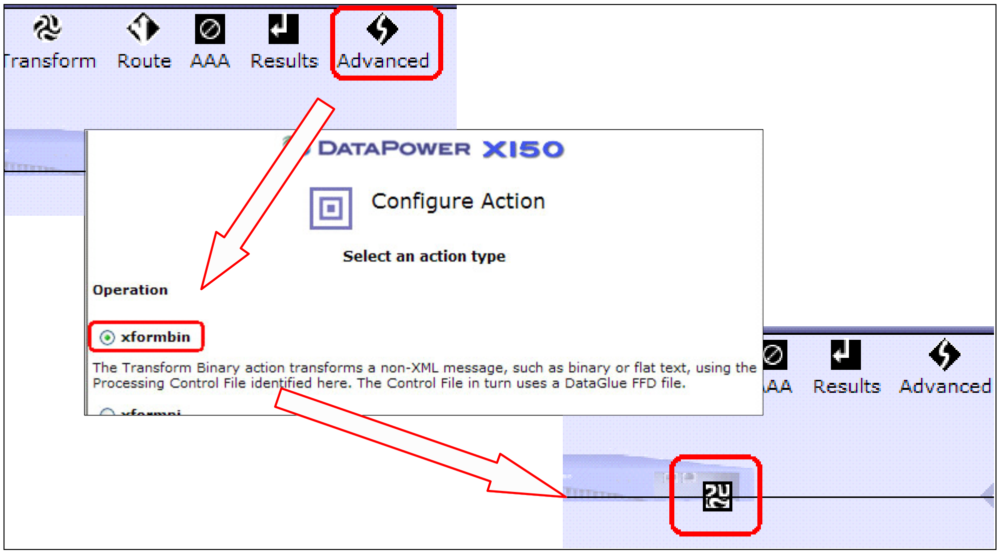
```
  <?xml version="1.0" encoding="UTF-8" ?>
- <AddressGroup>
  - <Address>
    - <details>
        <city>Virginia Beach</city>
        <state>VA</state>
        <street>3329 Elm Road</street>
        <zipCode>23452</zipCode>
```

Figure 5-14 WTX Design Studio testing in Map Designer

In summary, the files needed for input into DataPower and generated by the above steps are:

- MyMapName.xml: Containing the mappings and created by the *build* of the map that is used in the binary transform action specification
- MyInputTypeTree.mts: XML format of the MTT file
- MyOutputTypeTree.mts

DataPower provides an advanced operation to allow for processing of binary transformations. This option, xformbin, is an action (xformbin) in a policy rule (Figure 5-15).



Configure Action

Select an action type

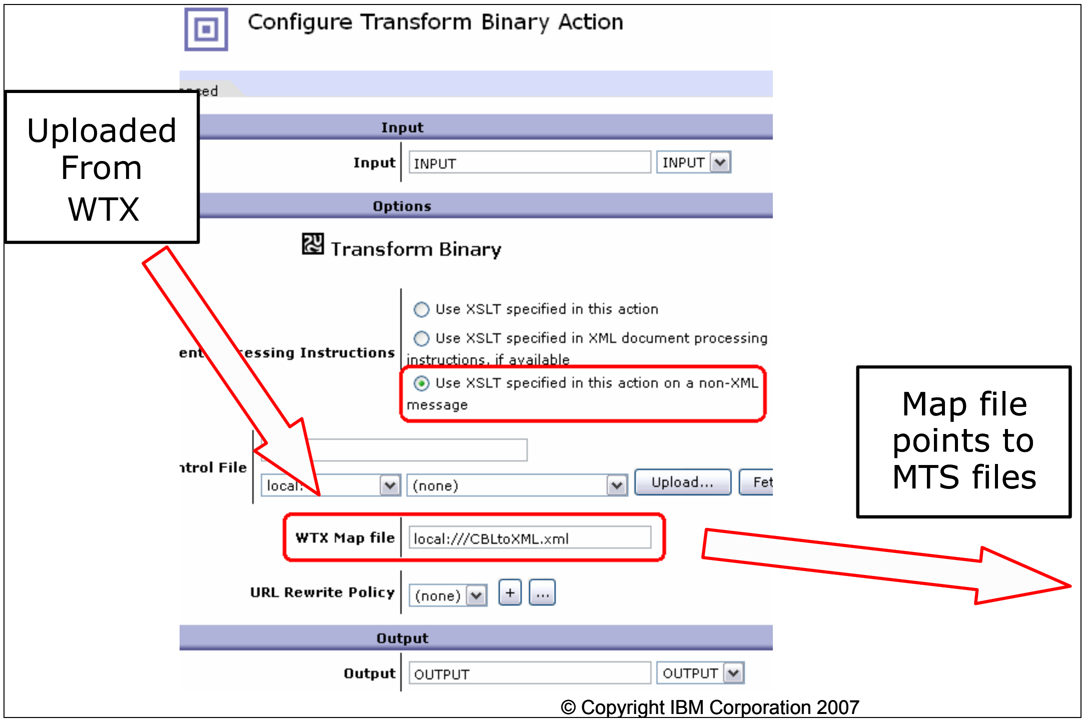
Operation

● xformbin

The Transform Binary action transforms a non-XML message, such as binary or flat text, using the Processing Control File identified here. The Control File in turn uses a DataGlue FFD file.

Figure 5-15 xformbin policy action

For configuration within DataPower, refer to Figure 5-16.



Configure Transform Binary Action

Input

Input: INPUT | INPUT

Options

Transform Binary

Processing Instructions:
○ Use XSLT specified in this action
○ Use XSLT specified in XML document processing instructions, if available
● Use XSLT specified in this action on a non-XML message

Control File: local: | (none)

WTX Map file: local:///CBLtoXML.xml

URL Rewrite Policy: (none)

Output

Output: OUTPUT | OUTPUT

© Copyright IBM Corporation 2007

Figure 5-16 Configuring a WTX-based xformbin action

The steps required after the type trees and mappings are defined and the MTS and XML files are generated are:

1. Upload MTS and XML files to the appliance.
2. Configure a document processing policy that uses a binary transform (xformbin) action.

In the DataPower run time, the map XML file does the work, with help from the MTS files.

Usage: WTX Design Studio can build type trees by importing XSD, DTD, and COBOL copybooks. So WTX can serve as the de facto mapping tool for DataPower for open standards XML objects as well binary and other non-xml objects such as EDI. This product combination extends DataPower's capabilities in the ESB.

Use WTX/DataPower for B2B, B2C scenarios where EDI and other market-based protocols are required, pre-existing WTX libraries manage the inbound proprietary protocols, and DataPower does protocol translation for downstream consumption. This shields the internal enterprise from proprietary data formats.

Scenarios

Refer to the WTX scenario found in *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327.

Refer to the WTX scenario found in "Getting started with WebSphere Transformation Extender Design Studio on DataPower" in developerWorks at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0712_vila/0712_vila.html

Information about WebSphere Transformation Extender can be obtained from:

http://www-306.ibm.com/software/integration/wdatastagetx/library/index.html

5.2.2 DataPower and WebSphere MQ: Quality of service to the traditional

Secure and reliable messaging provided by WebSphere MQ is fundamental to the SOA connectivity and strengthens the connectivity of DataPower to the enterprise. WebSphere MQ offers the necessary qualities of service required for your mission-critical business and provides a messaging backbone that can be used by all IBM ESB offerings.

WebSphere MQ for DataPower

IBM WebSphere MQ allows asynchronous message communication across a network. Whereas HTTP communication is analogous to telephone calls, message delivery over WebSphere MQ is analogous to a courier service. For point-to-point communications, messages are deposited in a queue and consumed by a service at a later time. The Queue Manager maintains a set of queues in one node in the network. Separate queues store and forward request and response messages. Queues mainly work in a first-in-first-out fashion unless a special weighting for messages is implemented.

The DataPower XI50 device can exchange messages with WebSphere MQ systems by acting as an MQ client node. This capability allows the DataPower device to bridge different messaging and transport protocols, such as HTTP or TIBCO EMS to WebSphere MQ. Messages originating within or outside of an MQ messaging bus can flow easily to and from another MQ messaging bus or other messaging system, such as HTTP or Tibco EMS. It is the multi-protocol gateway service running on the XI50 that makes this possible.

In addition to the messaging system bridging, the multi-protocol gateway service running on the XI50 can also apply the full range of transformation, security, authorization, routing, logging, and customization services available to the messages flowing through the XI50 to and from the WebSphere MQ system.

A multi-protocol gateway service, along with the other configuration objects employed on the DataPower device to implement interoperability with WebSphere MQ messaging systems, offers a great deal of flexibility for tuning the optimal interconnection. This section discusses these configuration options in the context of an enterprise architecture.

IBM WebSphere MQ performs all network communications over a channel. More specifically, the software program that allows network communication between a MQ client and the MQ queue manager is known as a client channel. The channel is a program that runs on the same host as the MQ queue manager that provides network connectivity, rather than the connection itself. If a client application is local to the queue manager, then a channel is not necessary, but is allowed. Since the DataPower appliance is always remote to the queue manager, communication always takes place over a channel.

Why WebSphere MQ for z/OS

To meet stringent business-continuity requirements, IBM WebSphere MQ for z/OS helps ensure reliable, proven message delivery, where messages are delivered exactly once and transactional (unit-of-work) support helps ensure the integrity of critical transactions. Engineered natively for z/OS, WebSphere MQ for z/OS takes full advantage of the unique features of the platform to enable its tremendous quality of service and dynamic workload management. Features include IBM Resource Access Control Facility (RACF®), automatic restart manager (ARM), IBM Workload Manager (WLM), Parallel Sysplex® with WebSphere MQ shared-queue support, DB2 data sharing, and Resource Recovery Services (RRS) global transaction coordination. WebSphere MQ for z/OS provides a specialized bridge for CICS and IMS transactions. For these reasons, a WebSphere MQ client is now on the DataPower XI50, which allows the device to be a powerful gateway for the z/OS platform.

Basic MQ architecture

The following illustrations represent the basic architecture created when a DataPower XI50 is used to connect an HTTP-based messaging system (typical of a Web services architecture) to a WebSphere MQ- based messaging bus inside the enterprise. Figure 5-17 on page 153 includes the primary configuration objects created on the DataPower device as well as the configuration of the MQ queue manager to which the DataPower device connects and exchanges messages.

In both of these architectures, the DataPower device acts as an MQ client only. It does not act as a queue manager to provide guarantees of message delivery or full rollback capability. The DataPower device does implement some transactionality, as discussed below.

HTTP to MQ

The Front Side Handler object implements HTTP transport connectivity on the client (front) side of the device. The multi-protocol gateway employs MQ-based URLs to determine the MQ queue to which requests are forwarded, and also from which replies are pulled.

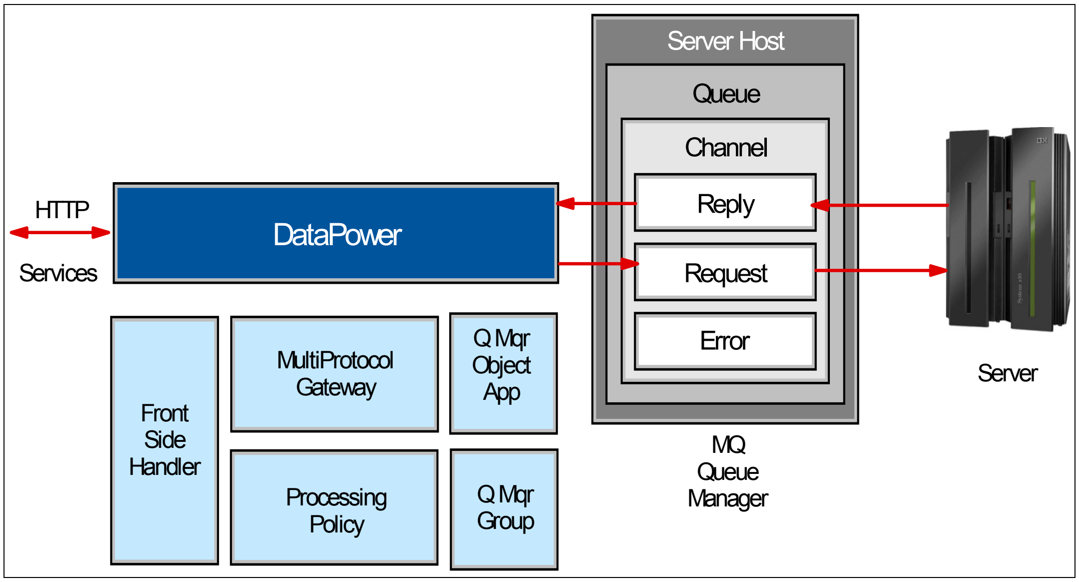


Figure 5-17 HTTP to MQ

In Figure 5-17, HTTP to MQ, messages flow to and from the DataPower device and work is performed by the DataPower device in the following sequence:

1. The HTTP client sends an HTTP-based request (typically an HTTP post containing a SOAP XML document, but may contain binary data) to the DataPower device. A Front Side Protocol Handler listens on an assigned port for incoming requests.
2. The Front Side Handler passes the message to the multi-protocol gateway service object. The MPGW then applies any and all relevant processing policy actions on the message.
3. The multi-protocol gateway may dynamically determine the appropriate destination to which to route the message, or may route all messages statically to a particular destination. In either case, in this architecture, the destination is a particular queue managed by a particular MQ queue manager. The DataPower MQ queue manager object contains the necessary information to establish a connection to the MQ queue manager.
4. The message is placed on the destination queue. If the network connection to a queue manager fails for any reason, the DataPower device can automatically retry the connection (as determined by the automatic retry property of the queue manager object).

   If the remote queue manager becomes unavailable, the DataPower device can optionally attempt to place the message on an alternate, or failover, remote queue manager, determined by a queue manager group.

   The DataPower device will automatically retry placing a message on a queue that can be reached once, in response to a 2009 MQ error code. Any further retries must be handled by the error handling configuration of the multi-protocol gateway.

   If the PUT attempt ultimately fails, an error is returned to the front side of the transaction (HTTP 500 in this case).
5. The DataPower device polls the reply-to queue to find a correlated response message. The gateway examines the correlation ID value in the MQ header of messages on the reply-to queue. When this ID is the same as the message ID assigned to the request, the

gateway takes the message as the response. If such a message is found, the multi-protocol gateway may again apply any configured processing policy actions to the response and returns the reply to the requesting HTTP client. This includes error responses from the back-end application server. If no response is found, the MPGW generates an error to the front -client.

The multi-protocol gateway can be configured to retrieve and process any message found on the reply-to queue, rather than only the correlated message. This can be done by using the Setvar processing action, or by using the set-variable() extension function. Using either method, set the extension header variable X-MQMD-Get to a blank or null value.

The multi-protocol gateway can be configured to ignore any response, including no response, by setting the gateway's response type property to pass-thru. In this case, the gateway will continue to poll the designated reply-to queue for a correlated response message. If one is found, it is returned without change to the front-side client. If none is found, nothing is returned to the front side by default. The gateway does not treat the lack of a response message on the reply-to queue as an error.

The URL used to open the connection to the back-end request queue can omit designating a reply-to queue. In this case, the multi-protocol gateway will not wait for a correlated response message, terminating the transaction. No response is sent to the front side HTTP client.

MQ to HTTP

Conversely, Figure 5-18 is an illustration of the basic architecture created with a DataPower XI50 that is used to extend a WebSphere MQ-based messaging system out to a Web services architecture.

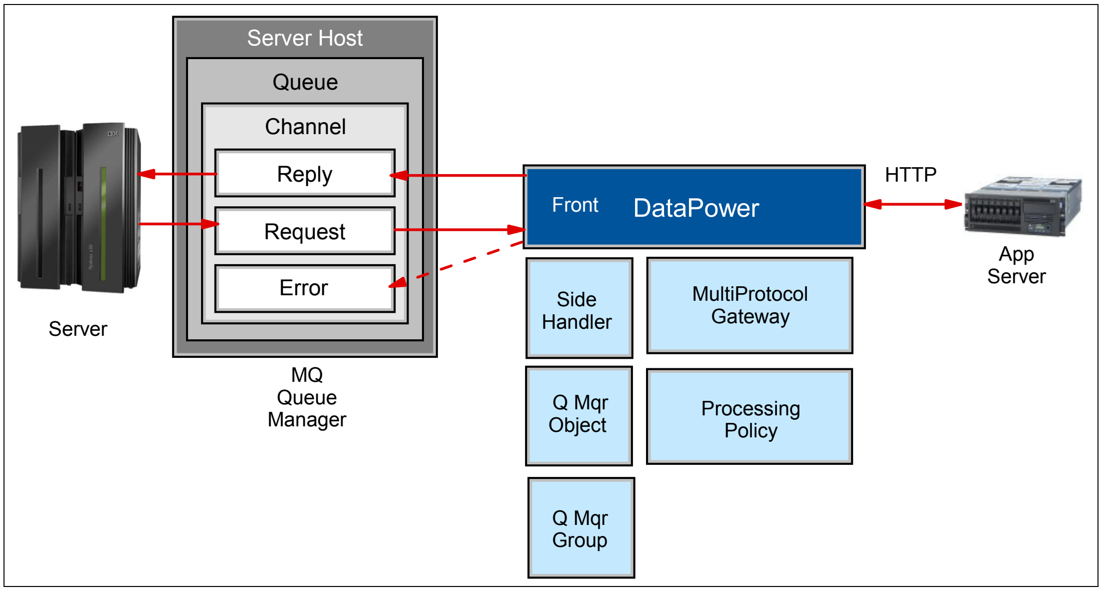


Figure 5-18 MQ to HTTP

Here, the Front Side Handler polls a particular MQ queue for request messages, and places replies from the back-end services on another MQ queue. The front-side queue manager object may optionally place messages in an error queue on the MQ queue manager. A standard HTTP URL is used to determine the destination to which requests are forwarded, and from which answers are received in accordance with the HTTP specification.

Messages flow to and from the DataPower device and work is performed by the DataPower device in the following sequence:

1. An MQ Front Side Protocol Handler polls the configured request queue, managed by the referenced MQ queue manager, for incoming requests. All messages found on the queue are copied from the queue.

   To implement redundancy, the multi-protocol gateway can be configured to use more than one Front Side Protocol Handler polling a range of queues. It is then up to the MQ system, independent of the DataPower device, to route messages to an active queue.

   By default, all request messages are immediately removed from the request queue. The device can be configured to leave the request message on the request queue until processing by the gateway is complete with no errors by setting the units of work property of the MQ queue manager object in use by the Front Side Protocol Handler to 1.

2. The Front Side Handler passes the message to the multi-protocol gateway service object. The MPGW then applies any and all relevant processing policy actions on the message. The multi-protocol gateway may dynamically determine the appropriate destination to which to route the message, or may route all messages statically to a particular destination. In this scenario, the back end employs HTTP. The gateway opens an HTTP connection and typically posts the request.

3. The multi-protocol gateway gets the response from the back-end application server. In this scenario, the HTTP protocol requires a positive response (which may contain an error message) or the gateway will register an error. The gateway applies any applicable processing policy actions to the response. Any errors will terminate the transaction.

   If the response type property of the gateway is set to pass-thru, no processing is performed on the response. The response message, if any, is returned unchanged to the front-side response queue.

4. The response message may be placed on the reply-to queue specified in the Front Side Protocol Handler. Unlike HTTP, the MQ protocol does not require a response.

   If the gateway response type has been set to pass-thru and the gateway has encountered an error, then no message is placed on the reply-to queue. If the queue manager units of work has been set to 1, the transaction is marked complete and the message is removed from the request queue even if an error has occurred.

   The gateway can also remove a message that results in an error and place the offending error on an alternate queue available through the same MQ queue manager. To enable this behavior, set the units of work to 1 and automatic backout to on in the MQ queue manager object. You may then set the number of times the messages will be reprocessed, and designate an alternate queue for placement.

Transactionality

In general, a transaction is a sequence of operations that either commit or roll back their work. A transaction rolls back if any one of the operations in the transaction fails. A transaction commits if all the operations in the transaction succeed.

The MQ protocol includes the concept of transactionality—that is, the ability to remove messages from a queue only when the agent that removed the message has successfully processed the message. This concept also includes the ability to roll back transactions, much like a database system. The MQ system implements transactionality in part through the use of units of work. The terms *transaction* and *unit of work* are interchangeable.

Resource managers such as WebSphere MQ queue manager can participate in a global unit of work, which involves the processing of resources from multiple resource managers. A transaction manager is required to coordinate such a transaction. It uses the two-phase commit protocol, a prepare and commit phase. The prepare phase ensures that all resources marked for commitment can be recoverable. The commit phase sends a request to all resource managers to commit their work.

The MQ system allows either:

- Local units of work between an MQ client and an MQ queue manager. A local unit of work is when only the queue manager resources are being updated.
- Global units of work, which may involve a series of clients and managers. A global unit of work is when resources of other resource managers are also being updated.

The DataPower device does *not* participate in globally managed transactions and cannot communicate with a transaction manager. The DataPower device can only participate in local units of work. Furthermore, a unit of work can apply only to one message.

Therefore, by default, the DataPower device does not enforce transactionality. Rather, messages are put on queues with no expectation of a response. Messages are removed from queues immediately, without regard to success of processing.

However, with MQ augmenting an ESB infrastructure, it can partner with DataPower by assuming the transactionality responsibilities by correctly setting the units of work property of the MQ queue manager object. The DataPower device can provide message accountability and guarantee that messages are not lost between DataPower and MQ. This architecture thus allows for DataPower to inherit the global transactionality that MQ is participating in at the time.

In the event of failure during a unit of work, or if the application determines that it cannot complete the unit of work for any reason, changes to resources that have already been made are backed out, or rolled back. If DataPower does not participate in the global unit of work, then how do we fail back through DataPower?

A good design pattern is to let the MQ queue manager inject a message back to DataPower representing the failed event. DataPower itself does not need to know the intent as long as there is a correlation ID correctly set. DataPower can therefore return the response to the client and the client will manage the rollback message. Because DataPower is loosely connected in the flow, it readily participates as a first class citizen in long-running transactions that have XA global signatures.

To further enhance the quality of service with either local or global transactionality, you can have DataPower interact with the MQ infrastructure in a particular way. The best design for DataPower/MQ interaction is to have DataPower communicate with one and only one MQ queue manager for any particular message flow. The reason for this is that DataPower uses the correlation identifier (CorrelID) as the correlation key for any request response scenarios. This identifier is unique to that particular queue manager.

It is important to understand that DataPower will propagate a transaction between two different WebSphere MQ queues managers. But to maintain integrity with request and reply messages between DataPower and WebSphere MQ, a handshake with a single queue manager is required because of the correlation identifier aspect. Thus, in Figure 5-19 the following two scenarios are accurate, but the *bus scenario* is preferred. This is necessary for request and reply correlation, particularly with transactional or guaranteed messaging requirements.

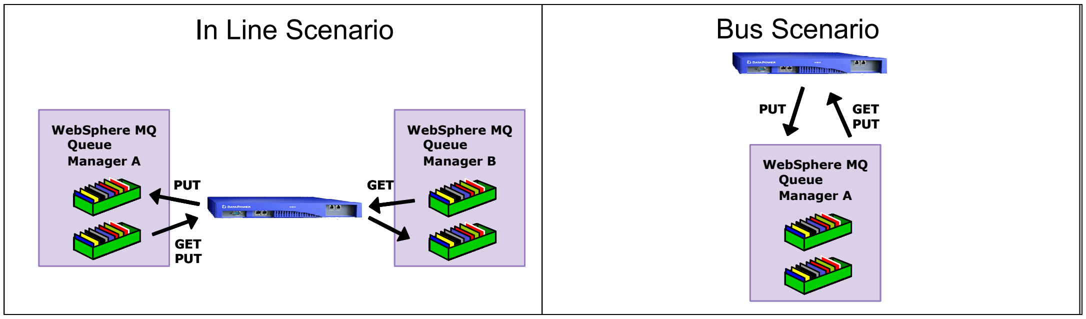

In Line Scenario
WebSphere MQ Queue Manager A
PUT
GET PUT
GET
WebSphere MQ Queue Manager B
Bus Scenario
PUT
GET PUT
WebSphere MQ Queue Manager A

Figure 5-19 MQ scenarios

Back-side processing

When the destination URL used by a gateway includes Sync=true, then the MPGW will commit the message to the request queue only when the entire message has been successfully put to the queue, thus making it available for other programs, such as the back-end application, to get the message.

The multi-protocol gateway can be configured to retrieve and process any message found on the reply-to queue, rather than only the correlated message. This can be done by using the Setvar processing action, or by using the set-variable() extension function. Using either method, set the extension header variable X-MQMD-Get to a blank, or null, value. For example:

```
setvar INPUT 'var://context/INPUT/_extension/header/X-MQMD-Get' '<MQMD/>'
```

Front side processing

Transactionality will cause the DataPower device to copy the message from the remote request queue, but not commit the get (that is, permanently remove the message from the queue) until the transaction is complete. The gateway places the message on hold, making it unavailable to any other application while the transaction is in process.

Typically, a transaction is complete when the gateway places a response on the response queue designated in the configuration of the Front Side Protocol Handler. However, unlike HTTP, responses are not required in the MQ protocol, and in many cases, might not be expected. In such a case, the multi-protocol gateway must be configured to complete the transaction through some other mechanism. When the gateway considers the transaction complete, it will then commit the fetch of the message and it is removed from the request queue.

If the multi-protocol gateway encounters an error and units of work is set to 1, the gateway will not remove the message from the request queue. The message does become available to other applications for removal. The gateway will continue to reprocess the message until the message is removed from the queue.

The gateway can optionally remove a message that results in an error and place the offending error on an alternate queue available through the same MQ queue manager. To enable this behavior, set the automatic backout to on in the MQ queue manager object. You may then set the number of times that the messages will be reprocessed and designate an alternate queue for placement.

Routing

A multi-protocol gateway can route messages to and from MQ queues, just as messages are routed to and from HTTP destinations using the same basic mechanisms. The gateway may use a static back end or dynamically determine a route.

Routing inputs

A multi-protocol gateway can examine the following inputs to help determine the desired message routing:

- Message content: The gateway can dynamically determine the destination queue by employing an Xpath routing map to examine the content of the message, or by using a custom stylesheet.
- Protocol header: The gateway can dynamically determine the destination queue by examining the value of the MQ protocol headers. As of firmware version 3.6.0.0, these values are easily obtained using a processing metadata object. A processing metadata object returns an XML nodeset containing the selected headers and values.

Routing destinations

The gateway can route all messages to the static back-end URL determined by the gateway configuration. The gateway may also determine the back-end destination dynamically. If the gateway uses dynamic determinations, then the gateway processing policy must set a back-end target at some point, using either the set-target() or xset-target() DataPower Extension Function calls.

Messages may also be sent or routed to one or more alternative destinations using the route (or route async) processing actions, just as with HTTP messages. For example, a single request message may contain a number of attachments. These attachments may be separated from the original request and routed individually to a particular destination (that may or may not be an MQ queue). The processing policy of the gateway may collect the responses and construct a reply message, may ignore the responses, or may send a response message that does nothing more than acknowledge receipt of the original request.

All destination determinations can employ an MQ URL to express the destination.

Authentication and Authorization

A multi-protocol gateway can use all of the standard AAA methods available through an AAA policy object. For example, you can examine a WS-Security header to obtain an identity that can be used to authenticate and authorize against an external authority. In addition to the standard methods, you can also use values taken from the MQ header to effect transport-independent AAA. This is done using a processing metadata object.

A processing metadata object contains a list of the header name-value pairs. The AAA extract identity phase can be set to use a processing metadata object. When the AAA action executes, the system extracts this list of pairs and forwards the results as a nodeset to the AAA authentication phase. The AAA authentication phase must then use a custom stylesheet to create an authentication query to some authority and pass on the results to the next phase of AAA.

Consider a scenario in which all messages must contain a user identity authenticated by a local LDAP authority. In this case, the processing metadata object could contain just the UserIdentity node of the MQ header, which would then be returned to the custom authentication phase stylesheet.

Error handling

A multi-protocol gateway can employ all of the standard error-handling capabilities available for HTTP traffic. This includes processing policy error rules, the on-error action, automatic fault generation when a request or response message does not pass validation checks, and custom error message generation.

The MQ protocol allows requests that have no response. The gateway may be configured to return no response message when an error is encountered during processing. As mentioned above, the Gateway response type must be set to pass-thru to implement no response.

The device provides a number of MQ-specific event (error) codes that can be read and used for error processing.

Monitoring, logging, and status

All of the standard capabilities of the multi-protocol gateway can be applied to monitoring and logging of MQ transactions. These include log targets that send selected log messages to remote facilities off the device, rotating log files, the log action that delivers a copy of an entire message to a remote facility, and the standard count and duration monitors.

Because MQ runs on a multi-protocol gateway, the service level monitor capability is also available. This allows you to notify, shape, and throttle traffic in accordance with policy rules that can filter by credentials (who is asking) and by resources (to do what). The resource class of an SLM policy offers the ability to select MQ-specific attributes, such as reply queue name and request queue name. You implement SLM by including an SLM action in the processing policy of the gateway.

Note that in the case of MQ, it is also possible to log messages by sending a copy of the message to a particular remote queue using a Results action. This queue can then simply be archived.

Status information can also be gathered using all of the standard means offered by a gateway, such as SNMP.

Using MQ with a Web services proxy

The Web services proxy service can also interoperate with the MQ messaging system. Some of the methods available to create such a configuration are:

- The proxy may employ an MQ Front Side Protocol Handler to get SOAP request messages from an MQ queue, and optionally place any response on a corresponding reply queue.
- The proxy may employ a statically defined back end that uses MQ queues to put requests and get responses. As most WSDL files indicate an HTTP endpoint, you may need to manually change the proxy type property to static back end from the typical default of static-from-wsdl. You then need to specify the back-end URL using the MQ URL syntax described above.
- The proxy may employ a results (or results async) action in the processing policy that sends messages to an MQ queue.

MQ and JMS

It is possible to transport JMS messages over an MQ system. In such a case, some architectures allow for the selection of MQ messages based on values contained in the JMS message headers. As of Version 3.6.0.0, the DataPower device does not natively recognize the JMS headers as header values in the MQ message. The DataPower device cannot select and GET messages based on JMS header values.

It is possible to use the DataPower device to read and select messages based on JMS header values by using a binary transform operation on the MQ message body to extract the JMS header values into an XML nodeset. This extracted nodeset can then be examined for the desired attributes.

Traditional MQ objects

The device includes a number of MQ-related services that predate the multi-protocol gateway. These are the MQ gateway, the MQ host, and the MQ proxy. These objects should no longer be used for MQ interoperability. The multi-protocol gateway (or, alternately, a Web services proxy) provides all of the functionality offered by these services, in addition to many new features and capabilities.

The device provides a number of traditional system variables containing information that might be useful for routing messages.   An example is:

```
var://service/correlation-identifier
```

R/W examines or sets the value contained in the MQMD (Message Descriptor) Correlation Identifier header, used for MQ host and proxy services.

These variables are not available to the multi-protocol gateway or Web services proxy and should not be used.

Resources

For further information see:

- “WebSphere DataPower WebSphere MQ Interoperability Release 3.6.0.0,” developerWorks

  http://www-1.ibm.com/support/docview.wss?uid=swg21255199
- “Integrating WebSphere DataPower SOA Appliances with WebSphere MQ,” developerWorks

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html?S_TACT=105AGX78&S_CMP=ART
- WebSphere MQ product home

  http://www.ibm.com/software/integration/wmq/
- IBM WebSphere MQ V6 Fundamentals Redbooks publication

  http://www.redbooks.ibm.com/abstracts/sg247128.html

5.2.3 Service-enable CICS and IMS traditional applications with DataPower

For System z, the challenge is not to build a new application using SOA principles, but rather to incorporate existing traditional applications into a service-oriented architecture. The same is true for the IT infrastructure in general. How do we build new infrastructure by utilizing SOA principles rather than converting a fully operational infrastructure into an SOA-enabled one? This section focuses on a CICS and IMS architecture featuring DataPower as an enabler in the middleware.

Service interface patterns

In this section, you will learn about service integration techniques that you can use without any code modification on the back-end application. There are architectural patterns for service integration. Interfaces for service integration are basically done in four different ways, as discussed in this section.

Native service interface (N)

The native service interface solution enables a core system for SOA by using features that are native to the transaction server, database manager, or whichever infrastructure component hosts the application component.

CICS Web services is an example of native service interfaces for mainframe connectivity. (An example of an adapter-provided service interface for mainframe connectivity is the IMS Simple Object Access Protocol (SOAP) gateway, which is discussed later.) Figure 5-20 shows that the service interface is provided by the container in which the application runs.

The key for abbreviations in the following figures are:

- P = presentation logic
- B = business logic
- D = data access logic

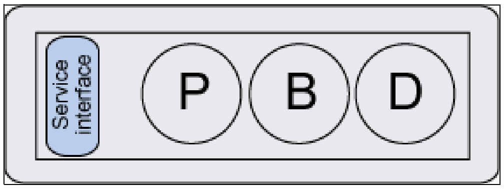


Figure 5-20 Native service interface

Adapter-provided service interface (A)

An adapter may be employed, if available, to translate between a proprietary interface to the system and SOA-compliant protocols (Figure 5-21).

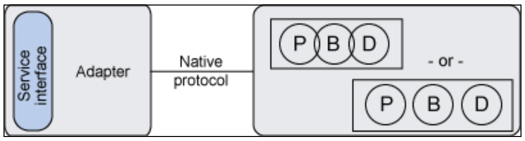


Figure 5-21 Adapter-provided service interface

The adapter usually resides outside the infrastructure component that hosts the application component that is to be service enabled. Adapters have been employed in the past to provide integration with middleware such as Web application servers.

Brokered or mediated service interface (B)

It is often easier to employ an intermediary enterprise service bus (ESB), or broker, to provide the service interface to the transitioned core application. The ESB can insulate the core application from the need to comply with new protocols, and it can transform the message content so that it can be processed by other services that connect through the ESB.

The DataPower appliance is an example of a brokered or mediated service interface, and is discussed later in this book (Figure 5-22).

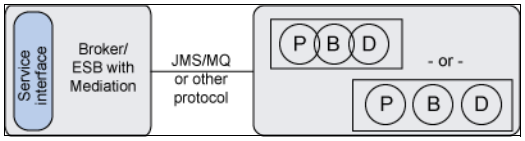


Figure 5-22 Brokered or mediated service interface

Redeveloped code with native service interface

The redeveloped code with a native service interface pattern involves taking existing core system code and rewriting it to conform to SOA-compliant structure and protocols. This often involves refactoring the source code so that it has a more SOA-friendly service structure (Figure 5-23).

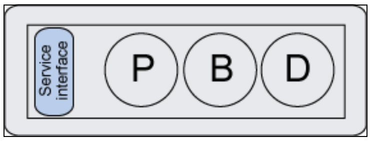


Figure 5-23 Redeveloped code with native service interface

Transition approaches

Select the appropriate transition approach depending on the level of service integration and SOA standardization that already exists, and the level that is to be achieved. There are three transition approaches available:

- Improve:
  - Focuses on service-enabling the assets that already exist.
  - Service enablement means that assets get a new type of interface that opens them up for service consumers.
  - This approach should not require any recoding or redevelopment of applications.
- Adapt:
  - Focuses on service integration.
  - The vehicle to achieve flexible service integration is the ESB.
- Innovate:
  - Results in a restructured, rewritten application that natively conforms to SOA-compliant standards and protocols using an ESB.
  - An innovated application is fully modularized so it can be reused more easily in the future, and should have a service interface.
  - Focuses on SOA enablement of existing assets, without writing any new code, and does not discuss the innovate approach.

Solutions for service-enabling IMS and CICS transactions

A typical IMS and CICS transaction is divided into presentation, business, and data logic layers. However, there are usually relatively tight dependencies between the layers, and they may not be easily separated and broken out to be exposed as services. This is not from bad design, but simply because the applications were built when there were no requirements for supporting multiple channels and different standards.

Most IMS and CICS applications are developed for a 3270 display panel environment, using several transactions in a sequence to fulfill a use case. Panel navigation on a 24x80 display compared to service interfaces in an SOA are very different in their characteristics. Reusability is an issue, making it hard to find possible service candidates.

The main concerns when moving into SOA are service granularity and separation of concerns.

SOA ambition level and transition approach

Depending on the requirements of the target level of service integration, there are different approaches. Do you want to create multichannel access, such as phone, Internet, intranet, extranet, and so on? Or, is it about integration with a portal, or maybe a fully fledged business process engine that will execute workflows? What kind of service integration are you looking for?

- Improve.

  The improve approach focuses on service enabling the assets that already exist. Service enablement means that assets get a new type of interface that opens them up for service consumers. Usually, it also requires that open standards are introduced into the architecture.

- Adapt.

  The adapt approach sets the ambition level one step higher than the improve. Mediation, transformation, routing, and an open standards-based service registry are introduced so that services can be dynamically located and called. As a result, the back-end traditional applications do not need to comply with new protocols.

The requirements on the current back-end traditional when it comes to separation of concerns and service granularity are higher than in the improve approach.

The starting point for the improve approach usually includes plenty of old traditional applications that need to get involved in a multichannel architecture. It may not be a good business case to redevelop all the functions using new techniques. Service-enabling and opening up some of the transactions for a multichannel architecture is good enough.

Quality of service and security are extremely important when it comes to service enabling back-end transactions. Sometimes you do not know who will call the service, from where, and through which channel. Reliability and security of the underlying integration platform become extremely important.

Solution techniques

The technique that you use in your solution should include technologies and products that help the solution meet the required level of service integration. Figure 5-24 shows the relationship between service interface patterns and transition approaches.

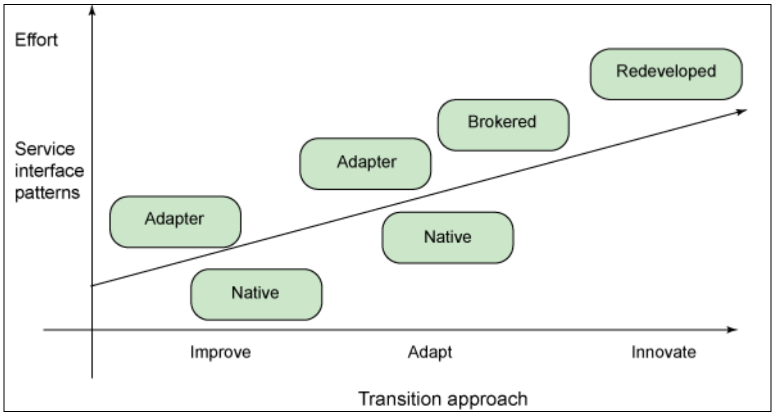


Figure 5-24 Patterns and transition approaches

This section discusses the IBM DataPower appliance approach only. However, before choosing a solution technique, do a thorough requirements analysis. Some questions to consider are:

- What is the scope of the solution? Is it a one-time solution, or will it be deployed enterprise wide?
- How long is the solution expected to live? Is it a temporary tactic solution, or a long-term strategic choice?
- Does the solution need to interoperate with an ESB? How about a service registry?
- What is the knowledge level or maturity of the developers? Are there development standards in place for building services in an SOA?
- What are the nonfunctional requirements? What are the volumes? Response time? Security? Transactions? Do the services need to be part of a distributed unit of work (UoW)?
- Systems management? Requirements on tools and routines for monitoring the solution?

DataPower appliance WTX IMS scenario

The following scenario uses the XI50 product, which has the following features:

- WebSphere Transformation Extender (WTX), any-to-any transformation engine (based on descriptors and mappings). Transformation can be done between binary or flat text to XML, XML to binary or flat text, binary to and from binary, and XML to and from XML.
- Content-based transaction routing.
- Protocol bridging (HTTP, MQ, and so on).
- Lightweight message brokering.
- Message-level security (WS-Security, XML Digital Signature, XML Encryption).

DataPower with IMS

Using Figure 5-25, let us walk through this from the back end to the front end:

1. Integration with IMS is done through WebSphere MQ, either with a native MQI interface in the IMS transaction or through the IMS OTMA bridge. In the latter case, back-end application code can run without code modification. Note that with the DataPower 3.6.1 release, the XI50 has native IMSConnect support. Although MQ can still be used, it is not required.
2. DataPower uses the built-in MQ client to connect to a queue manager, which provides the connectivity with IMS.
3. DataPower publishes the Web services interface and transforms the SOAP XML message into an MQ message.
4. Web services standards and security are maintained in DataPower, without any need for the back-end IMS application to deal with anything in that area.

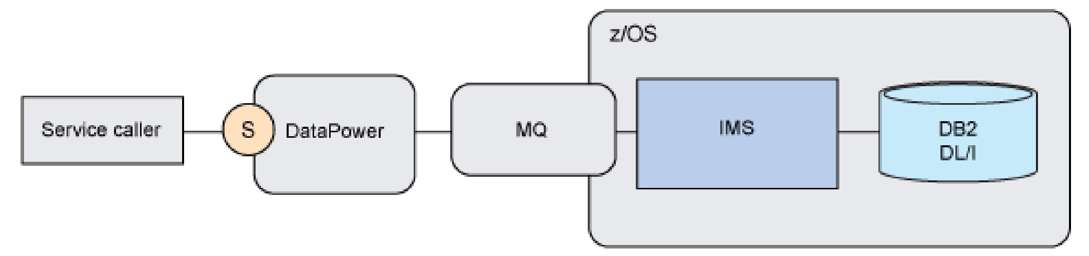


Figure 5-25 DataPower with IMS

DataPower with CICS

DataPower with CICS is very similar to the example with IMS, except that Open Transaction Manager Access (OTMA) is not available for CICS. The CICS transaction needs to be MQ enabled.

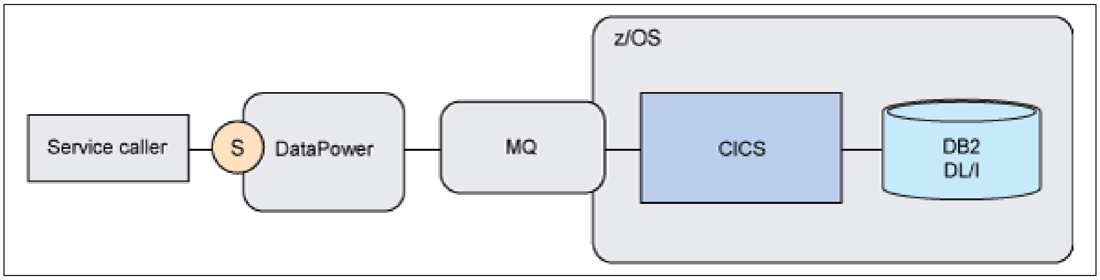


Figure 5-26 DataPower with CICS

DataPower with IMS and CICS

In a mixed IMS and CICS environment, DataPower can be used as a front end for both. Some of the benefits of this solution are:

- Loosely coupled interfaces between the front-end and back-end interfaces using MQ. Since both IMS and CICS are callable by MQ, there is strong support for other clients, platforms, and programming environments.
- Web services standards and definitions are kept and maintained in one place, in the DataPower box. Web Service Definition Language (WSDL) definitions are independent of the back-end system.
- XML processing is done separately from the back-end applications, which offloads the host from CPU-intensive processing.
- Rules and definitions for message and XML transformation are done in DataPower.
- The security model is externalized from the IMS and CICS applications and defined in DataPower.

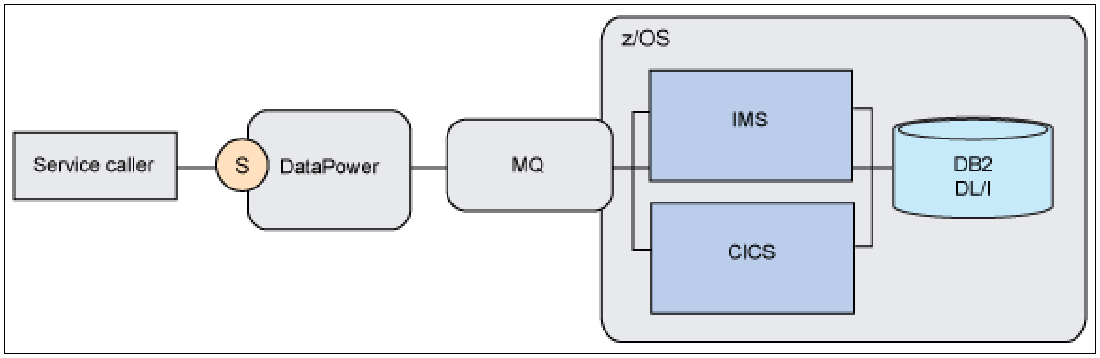


Figure 5-27 DataPower with IMS and CICS

Summary

In this section we explored scenarios for service-enabling CICS and IMS applications using DataPower. You learned important aspects to consider for service-enabling traditional applications, including determining the ambition level and whether the traditional code can be reused and exposed as a service using a native service interface, adapter, or a broker, or whether it needs to be completely redeveloped.

You discovered how IBM WebSphere DataPower Integration Appliance can help you service-enable your applications. In cases where back-end (MQ-enabled) mainframe applications are to be Web-service-enabled, the DataPower Integration Appliance enables you to:

- Keep all Web services standards, security, and definitions in one place.
- Avoid traditional programming because message transformations are done using schemas and configurations.
- Perform high-speed transformations, offloading XML handling from the mainframe.
- Have a non-intrusive solution that plugs easily into the existing architecture.

Resources

For further information see:

- "Service-enable CICS and IMS traditional applications using the IBM WebSphere DataPower SOA Appliance"

  http://www.ibm.com/developerworks/library/ar-datapow/

- *SOA Transition Scenarios for the IBM z/OS Platform*, SG24-7331

  http://www.redbooks.ibm.com/abstracts/sg247331.html?Open

- "Integrating WebSphere DataPower SOA Appliances with WebSphere MQ"

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html?S_TACT=105AGX78&S_CMP=ART

5.2.4 Building message flows in WebSphere Message Broker

IBM WebSphere Message Broker (WMB) provides an integrated ESB that can connect just about any application or service to any other application or service, which can be particularly beneficial for large heterogeneous environments. WebSphere Message Broker handles a broad range of transports and protocols, including WebSphere MQ, Java Message Service (JMS) Version 1.1, Hypertext Transfer Protocol over Secure Socket Layer (HTTPS), Web services, file, and user-defined protocols. It handles practically any data format, including C and COBOL data structures, XML, and industry formats, such as Society for Worldwide Interbank Financial Telecommunication (SWIFT), electronic data interchange (EDI), and the Health Information, Portability and Accountability Act (HIPAA). Advanced data transformation and validation for complex and variable data formats can be achieved in combination with WebSphere Transformation Extender for z/OS.

WMB with DataPower

To add scalable security processing when the security volumes for Web services increase, you can use the seamless integration between WebSphere Message Broker and IBM WebSphere DataPower Integration Appliance XI50. This integration enables you to configure the WebSphere Integration Appliance DataPower XI50 device using the WebSphere Message Broker operational console. It enables you to deploy WebSphere Message Broker flow end points and security profiles to the WebSphere DataPower Integration Appliance XI50 device using common descriptions and tools for security processing.

DataPower WMB scenario

An optimum way for DataPower to communicate with the z/OS is via WebSphere Message Broker (WMB), when WMB is already an implemented mediation server brokering messages to and from the mainframe over WebSphere MQ as the bus. In a design scenario, DataPower sends a request to a WebSphere MQ request queue, which is monitored by WMB for incoming requests. WMB transforms the incoming request and sends it to CICS for customer information. CICS processes the request and sends the data back to WMB. WMB transforms the data, builds the reply message, and sends it to DataPower via WebSphere MQ, which in turn sends it to the original requester.

Message Broker security integration with DataPower

An increasingly popular ESB processing pattern combines the strengths of WebSphere Message Broker and the DataPower XI50 or XS40 appliance, with the DataPower appliance serving as a security gateway to Message Broker. This pattern is especially valuable when processing messages over HTTP. This pattern leverages DataPower's strong security capabilities and brings significant value to this processing pattern. Refer to Chapter 4, "Security" on page 69, for further discussion on DataPower security patterns.

To implement this proxy pattern, DataPower takes on the security duties for Message Broker by performing WS-Security processing. This is done by creating a firewall within it, which requires administration and information from both the Message Broker message flow and the DataPower appliance. You can perform this administration manually on the DataPower appliance, or you can leverage the DataPower security wizard found in the Message Broker Explorer.

You can also use the wizard with a use case in which SOAP messages are received over HTTP and processed by a message flow. The body of the SOAP message has been secured using WS-Security processing and therefore must be decrypted before it can be processed within the message flow. A DataPower appliance is used as a front-end security gateway to decrypt the body of the SOAP message on the way into the message flow. The DataPower appliance is also used to encrypt the output message from the message flow before the reply is sent to the requesting application.

Resources

For further information see the following resources:

- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*, REDP-4365
- "Integrating DataPower with WebSphere Message Broker using the Broker Explorer"

  http://www.ibm.com/developerworks/websphere/library/techarticles/0707_storey/0707_storey.html

5.3 Web 2.0

This section discusses support for Web 2.0 technologies using the DataPower appliances. The features offered by Web 2.0 applications are becoming more widely adopted by individuals and enterprises alike. This section demonstrates how existing Web services can be used to provide content for Web 2.0 applications. The section covers the following:

- Overview of Web 2.0 technology
- Description of an example Web 2.0 integration
- DataPower configuration to support the integration

5.3.1 Web 2.0 overview

The phrase Web 2.0 has come to represent any Internet-based application that goes beyond merely serving dynamic content to clients. Web 2.0 applications or Web sites allow and encourage users to upload content to share with other Internet users. This ability sets them apart from other more traditional Web sites that are essentially read-only.

While the underlying technology that supports Web 2.0-enabled sites has not changed, the way in which the Internet is used has. The Internet is no longer a one-way street where content is delivered to the user's browser for consumption. Web 2.0 technology allows users to easily submit their own content for others to see or use a Web-based application as though it were part of their own desktop.

Web 2.0 technologies

Examples of Web 2.0 technologies include social networking, Web logs (blogs), wikis, and podcasting. This section focuses on content distribution using the Atom Syndication Format. Atom is similar to Really Simple Syndication (RSS), and was developed to resolve compatibility issues with the RSS format. Information distributed using the Atom or RSS format is referred to as a *feed*.

Atom defines an XML-based format for providing summarized informational updates for a particular subject. The maintainer of the feed can add new information or updates by modifying the Atom file for the feed. Users use feed readers or aggregators to collect and filter the new or updated information. An example use case is a news feed where breaking news headlines can be added to the Atom file by the feed's maintainer. A subscriber to the feed can then view the updated information with their reader when they refresh the feed.

Web 2.0 and DataPower

While support for Web 2.0 technology in the DataPower appliances is not yet widely used, the devices can be used to support Web 2.0-based applications. DataPower appliances can be used to mediate Atom feeds because they are XML based.

5.3.2 Sample Web 2.0 integration

To demonstrate DataPower's ability to inter-operate with and support Web 2.0-based technologies, we use DataPower to provide content from a SOAP-based Web service in the Atom Syndication Format.

SOAP Web Service

The example SOAP Web service used in this discussion provides a list of customers. Requests to the service simply define the number of customers that should be returned. An example request to the Web service is shown in Example 5-2.

Example 5-2 Web Service sample request

```
<env:Envelope xmlns:query-string="http://www.datapower.com/param/query"
xmlns:q0="http://sample10.datapower.ibm.com/crafted/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<env:Body>
<q0:CustomerListRequest>
<length>1</length></q0:CustomerListRequest>
</env:Body>
</env:Envelope>
```

The Web service responds with a list of customers, including their names and IDs. A sample response is shown in Example 5-3.

Example 5-3 Sample Web Service response

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<soapenv:Body>
<CustomerList xmlns="http://sample10.datapower.ibm.com/crafted/">
<Customer xmlns="">
<CustomerID>0</CustomerID>
<CustomerName>customer0</CustomerName>
</Customer>
</Customer>
</CustomerList>
</soapenv:Body>
</soapenv:Envelope>
```

Atom Feed

As described in “Web 2.0 technologies” on page 168, Atom feeds are XML based. An Atom document is shown in Example 5-4.

Example 5-4 Sample Atom document

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
<id>johndoe@my.test.feed.org</id>
<title>This is an example feed</title>
<link href="http://my.test.feed.org/feed/" rel="self"/>
   <link href="http://my.test.feed.org/"/>
<updated>2007-10-05T18:30:02Z</updated>
<author>
<name>John Doe</name>
<email>johndoe@my.test.feed.org</email>
</author>
<entry>
<title>DataPower Test Atom Feed</title>
<updated>2007-10-04T18:30:02Z</updated>
<summary>This is a test of the DataPower ATOM feed.</summary>
</entry>
</feed>
```

The feed element describes the title, ID, and update time stamp of the feed, as well as other metadata such as the author’s name and relevant Internet links. The entry elements define the actual informational content of the feed. Every feed and entry element must contain a title, ID, and time stamp element.

To represent the Web service as an Atom feed, the list of customers returned by the Web service needs to be transformed into individual Atom entries.

XSL transformations

The previous section briefly described the format of the Atom Syndication Format. To display the SOAP responses from the SampleRead Web service in this format, some XSL transformations must be performed by DataPower. This involves the following:

- Transforming the empty GET request from the Atom client to a SOAP request to the Web service
- Converting the list of customers returned by the Web service to Atom entry elements
- Adding metadata that describes the feed and its entries

These operations are performed using two XSLT stylesheets. The first generates a SOAP request and sends it to the SampleRead Web service. The response from the Web service is passed to the second stylesheet for processing. This stylesheet extracts the customer elements of the CustomerList element and converts them to Atom Syndication Format entry elements. It also inserts the required Atom metadata element's ID, title, and time stamp for both the entry elements and the root feed element.

DataPower configuration

This section describes the steps to configure the DataPower appliance to enable the SampleRead Web service as an Atom feed.

Two XML firewalls are created to support the SampleRead Web service as a Atom feed. The architecture is shown in Figure 5-28.

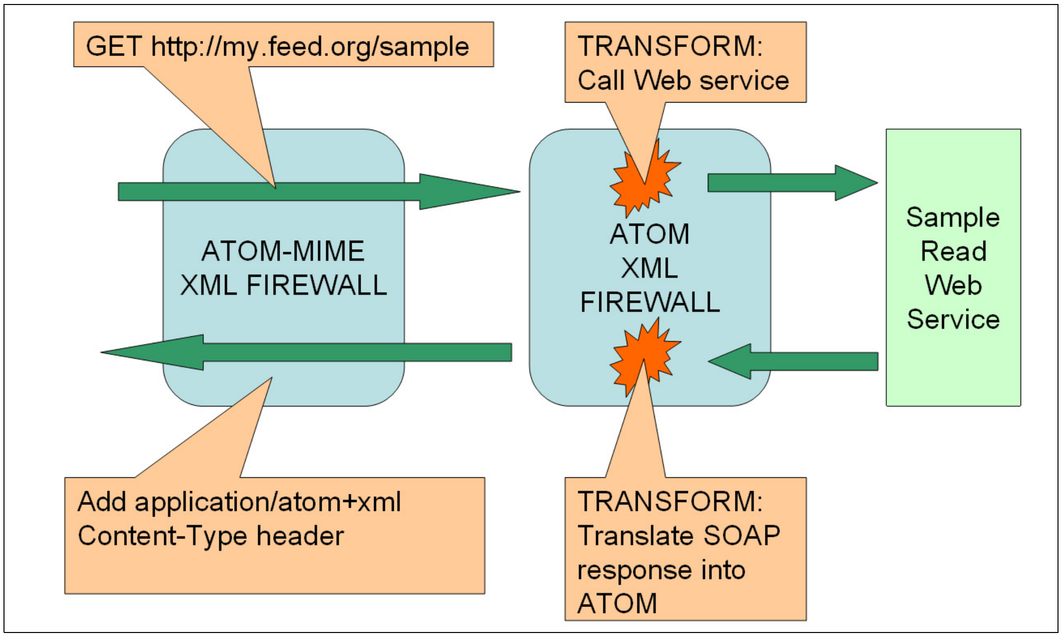

GET http://my.feed.org/sample
TRANSFORM: Call Web service
ATOM-MIME XML FIREWALL
ATOM XML FIREWALL
Sample Read Web Service
Add application/atom+xml Content-Type header
TRANSFORM: Translate SOAP response into ATOM

Figure 5-28 Logical architecture of DataPower XML firewall

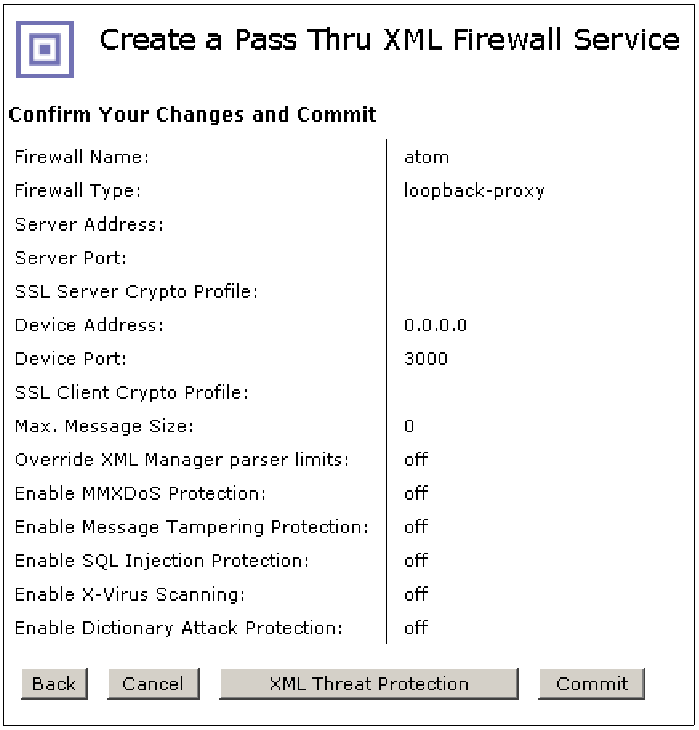
The first firewall receives the request from the feed reader and forwards the request unchanged to a second loopback firewall. The second (loopback) firewall has a processing policy that consists of two XSL transforms. The first of these transforms generates a SOAP request to the SampleRead Web service to obtain the list of customers. The second transform processes the Web service's response, and translates it into the Atom format. The translated output then becomes the response input of the first firewall. The first firewall's response processing policy adds the Atom Content-Type. Note that the reason that two firewalls are needed is because it is not possible to set response headers in a loopback firewall.

The Content-Type header for Atom feeds should be *application/atom+xml*, whereas it is *application/text+xml* for Web services.

Create the Web service facing XML firewall

To configure the DataPower appliance, perform the following steps. We configure the second firewall first.

1. At the Control Panel choose the button for the service **XML Firewall**.
2. In the panel Configure XML Firewall click **Add Wizard** to create a new XML firewall aided by a wizard.
3. Choose **Pass Through**. We only configure basic properties with the wizard and customize the XML firewall later.
4. Use the name `atom` for the firewall.
5. Select **loopback-proxy** for the firewall type.
6. For the clients to connect to our XML firewall, we set the device address to 0.0.0.0, which means that the firewall is listening on all active interfaces. We also define a device TCP port, in this case 3000.
7. The basic configuration with the wizard is finished. Press **Commit** to save the firewall.

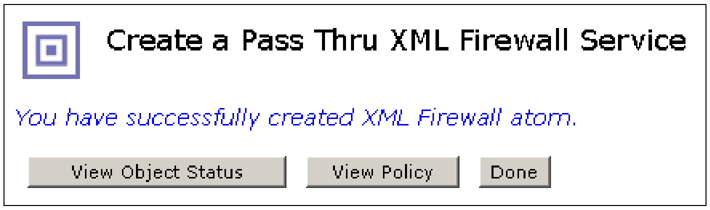


Create a Pass Thru XML Firewall Service

Confirm Your Changes and Commit

Firewall Name:	atom
Firewall Type:	loopback-proxy
Server Address:	
Server Port:	
SSL Server Crypto Profile:	
Device Address:	0.0.0.0
Device Port:	3000
SSL Client Crypto Profile:	
Max. Message Size:	0
Override XML Manager parser limits:	off
Enable MMXDoS Protection:	off
Enable Message Tampering Protection:	off
Enable SQL Injection Protection:	off
Enable X-Virus Scanning:	off
Enable Dictionary Attack Protection:	off

Figure 5-29 The atom XML firewall service

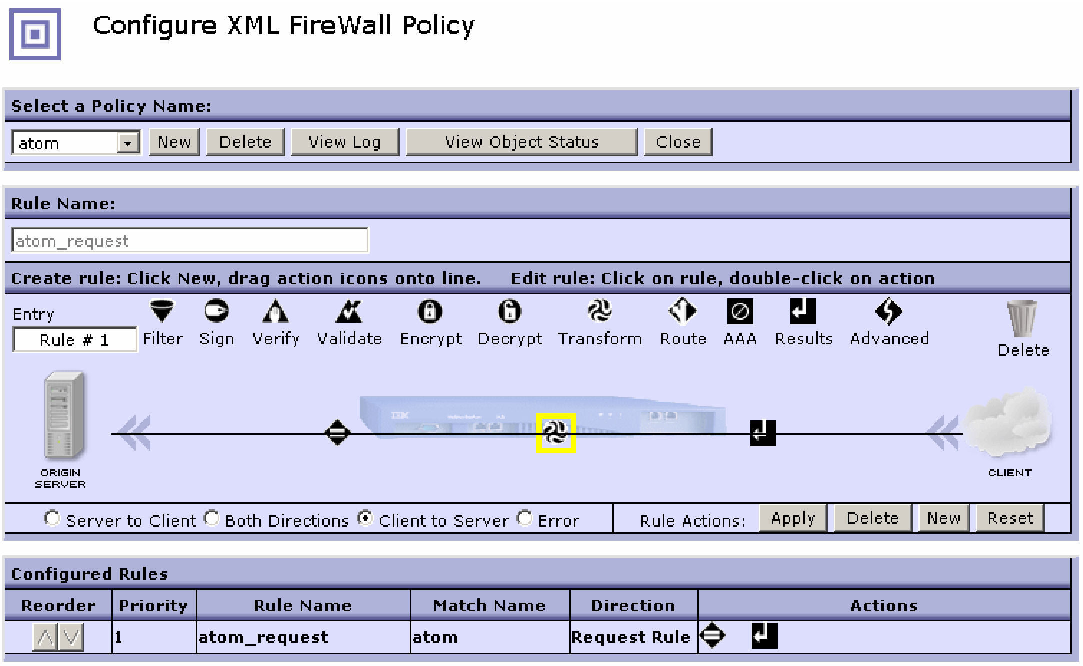


Create a Pass Thru XML Firewall Service

You have successfully created XML Firewall atom.

Figure 5-30 Finishing the wizard

8. In the main XML firewall panel (also shown when entered through the control panel/XML firewall) we select the newly created firewall. First we have to change the request type on the bottom right-hand side of the panel from SOAP to Non-XML. This is because the initial request from the feed reader is not in an XML format.
9. Select the ellipsis button (...) beside the atom firewall policy to customize it.
10. Drag and drop a **Transform** action onto the processing rule diagram.



Configure XML FireWall Policy

Select a Policy Name: atom

Rule Name: atom_request

Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Entry Rule # 1

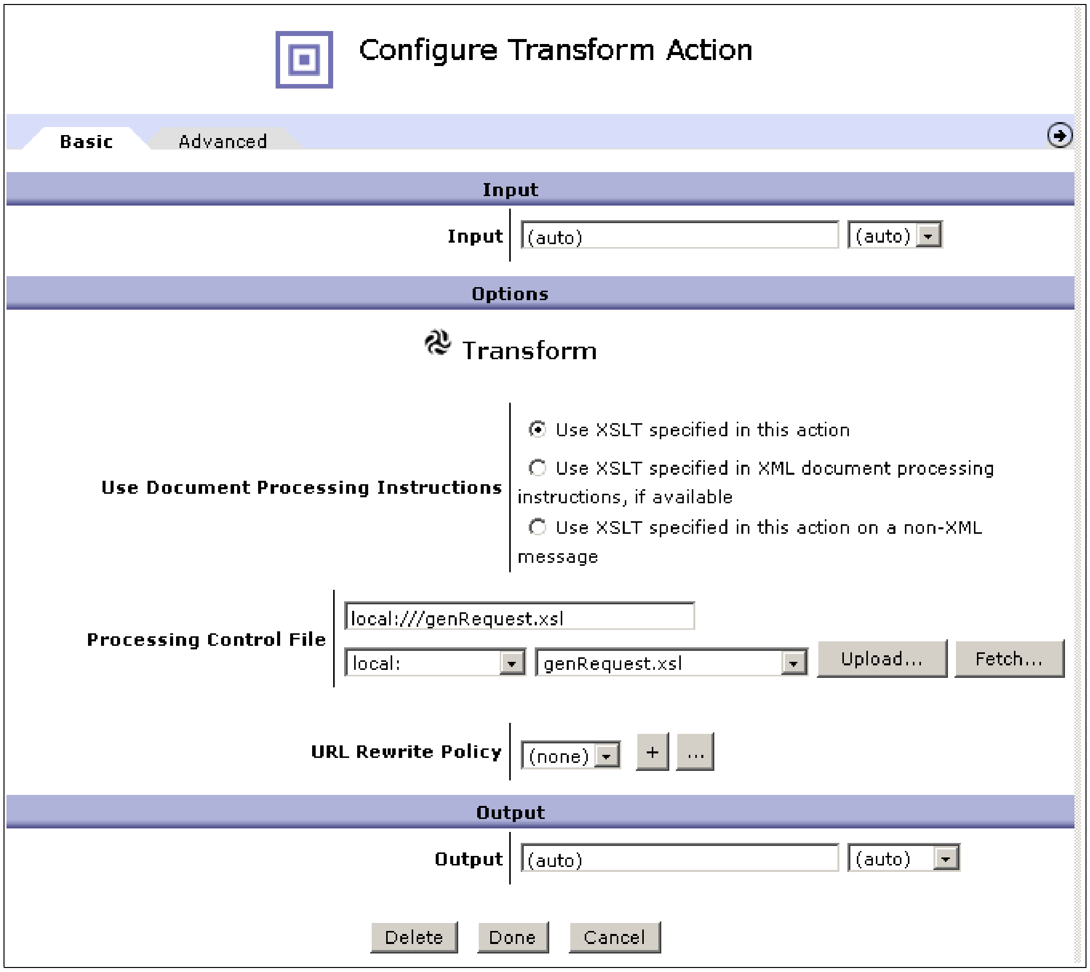
○ Server to Client ○ Both Directions ● Client to Server ○ Error

Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
∧ ∨	1	atom_request	atom	Request Rule	

Figure 5-31 Configuring the XML firewall policy

11. Double-click the new Transform action icon to configure this action. We have to upload the genRequest XSLT file we want to use. This is the XSLT that sends an out of band SOAP request to a Web service and receives a SOAP response with the requested list of items.



Configure Transform Action

Input

Input: (auto) (auto)

Options

Transform

Use Document Processing Instructions:
● Use XSLT specified in this action
○ Use XSLT specified in XML document processing instructions, if available
○ Use XSLT specified in this action on a non-XML message

Processing Control File: local:///genRequest.xsl
local: genRequest.xsl

URL Rewrite Policy: (none)

Output

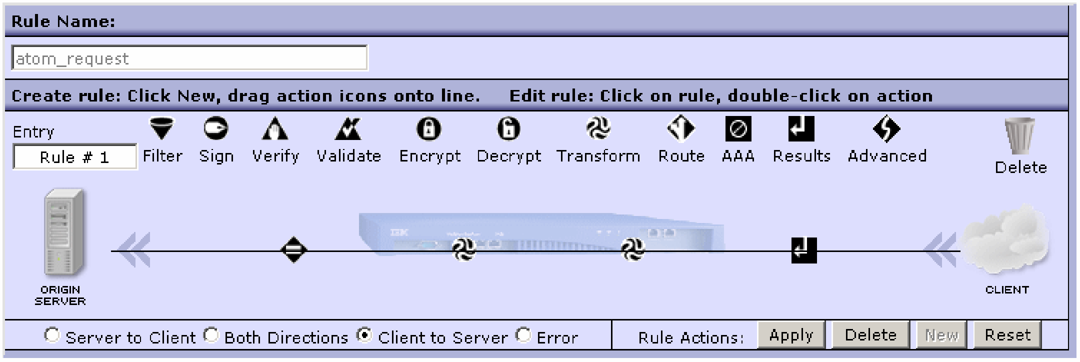
Output: (auto) (auto)

Figure 5-32 Configuring the Transform action

12. Click **Done** on the Configure Transform Action panel.

13. On the policy diagram insert another Transform action after (that is, to the right of) the existing Transform action.

14. Double-click the newly created **Transform** action and configure it to use the atomResponseMapper.xsl file. This stylesheet performs the transformation of the SOAP response from the previous step to the Atom format.



Rule Name:
atom_request

Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Entry
Rule # 1

○ Server to Client ○ Both Directions ● Client to Server ○ Error

Figure 5-33 The final rule for the atom XML firewall

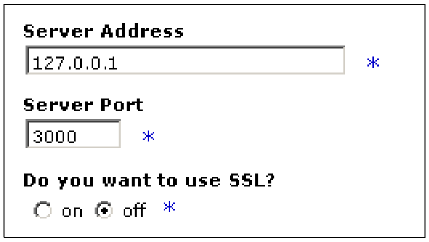
15. Apply the changes to this rule and close the policy editing window.

16. Click **Apply** and **Save Config** to save the changes.

Create the client-facing XML firewall

Now we need to create the XML firewall that receives requests from a feed reader. We use the wizard as before. Name the firewall `atom-mime` and perform the following steps:

1. Choose **static-backend** for the firewall type, as it allows us to define not only a front end but also a back-end system.
2. In the back-end configuration we enter the address and port of the XML firewall service that we created before. Use the address 127.0.0.1 to indicate a local service and the port defined above (3000).



Server Address
127.0.0.1 *

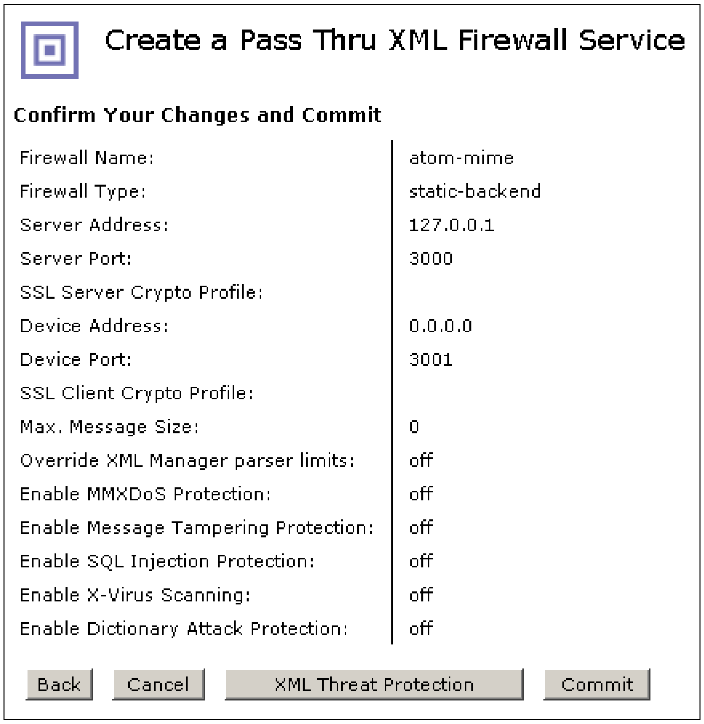
Server Port
3000 *

Do you want to use SSL?
○ on ● off *

Figure 5-34 Back-end connection configuration

3. Use port number 3001 for this firewall.

4. Commit the configuration of this firewall on the next panel (Figure 5-35).



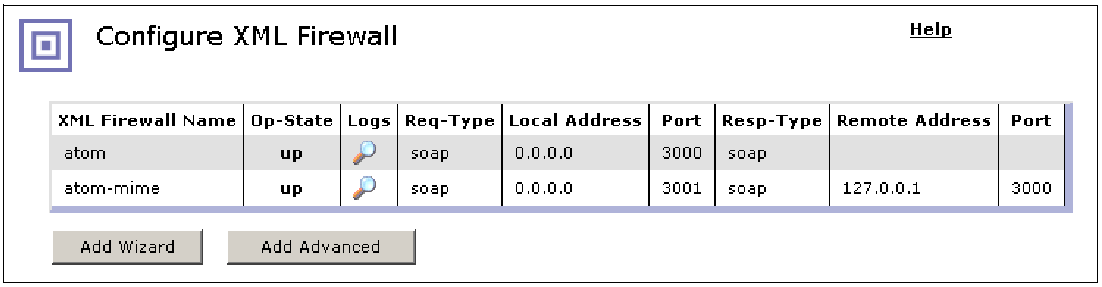
Create a Pass Thru XML Firewall Service

Confirm Your Changes and Commit

Firewall Name:	atom-mime
Firewall Type:	static-backend
Server Address:	127.0.0.1
Server Port:	3000
SSL Server Crypto Profile:	
Device Address:	0.0.0.0
Device Port:	3001
SSL Client Crypto Profile:	
Max. Message Size:	0
Override XML Manager parser limits:	off
Enable MMXDoS Protection:	off
Enable Message Tampering Protection:	off
Enable SQL Injection Protection:	off
Enable X-Virus Scanning:	off
Enable Dictionary Attack Protection:	off

Figure 5-35 XML firewall atom-mime

5. Select the newly created **atom-mime** XML firewall from the Configure XML Firewall page (Figure 5-36).



Configure XML Firewall

XML Firewall Name	Op-State	Logs	Req-Type	Local Address	Port	Resp-Type	Remote Address	Port
atom	up	🔍	soap	0.0.0.0	3000	soap		
atom-mime	up	🔍	soap	0.0.0.0	3001	soap	127.0.0.1	3000

Figure 5-36 The two created XML firewalls

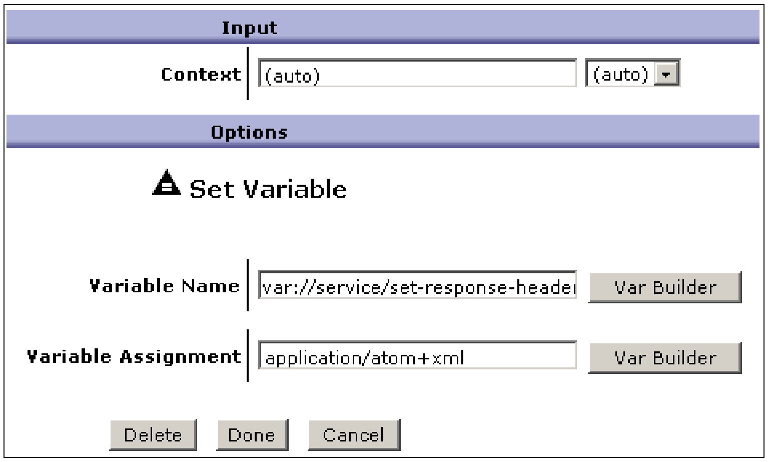
6. We set the request type and response type to Non XML.

7. Select the ellipsis button beside the atom-mime firewall policy to customize it.

8. Create a new rule action and make it a Server to Client (Response Rule).

9. Drag and drop an **Advanced** action onto the processing rule diagram. Double-click it and select the **setvar** (set variable) option.

10.Set the variable var://service/set-response-header /Content-Type to application/atom+xml (Figure 5-37).



Input

Context: (auto) (auto)

Options

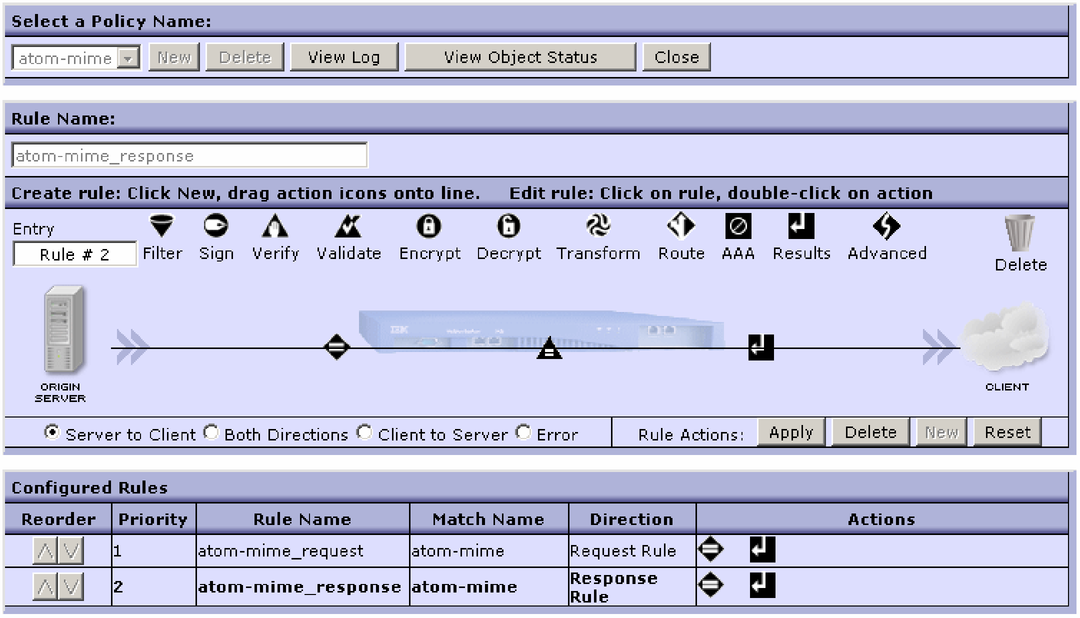
Set Variable

Variable Name: var://service/set-response-header

Variable Assignment: application/atom+xml

Figure 5-37 Setting a variable

11.Click **Done** to return to the processing policy window (Figure 5-38).



Select a Policy Name: atom-mime

Rule Name: atom-mime_response

Create rule: Click New, drag action icons onto line. Edit rule: Click on rule, double-click on action

Entry: Rule # 2

● Server to Client ○ Both Directions ○ Client to Server ○ Error

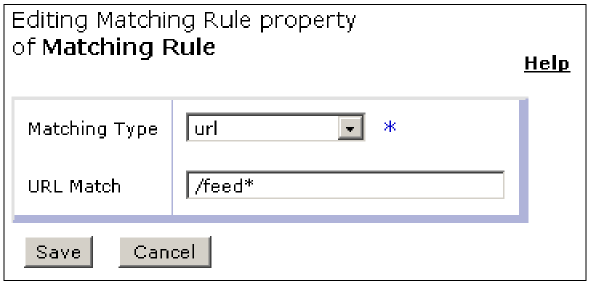
Configured Rules

Reorder	Priority	Rule Name	Match Name	Direction	Actions
Λ V	1	atom-mime_request	atom-mime	Request Rule	◆ ↲
Λ V	2	atom-mime_response	atom-mime	Response Rule	◆ ↲

Figure 5-38 The atom-mime firewall policy

12.Double-click the matching rule icon (=) to open the configuration window.

13.Select the ellipsis beside the atom-mime matching rule. On the Rules tab change the URL matching type to match only /feed* URLs. Save the changes. See Figure 5-39.



Editing Matching Rule property
of **Matching Rule**

Matching Type: url *

URL Match: /feed*

Figure 5-39 Creation of a new matching rule

14.Apply the changes to the processing policy and close the policy editing window.

15.On the main configuration panel, select the tab **Advanced** and set Disallow GET (and HEAD) to ON. That is, we want to *allow* GET requests for this firewall.

16.Click **Apply** and **Save Config** to commit the changes to the atom-mime XML firewall.

5.3.3 Demonstration

By turning on the probe for the firewalls, we can view the data as it traverses the appliance.

To test out the configuration, we direct a feed reader to the client-facing XML firewall called atom-mime. This firewall has the port number 3001, so the URL that the feed reader should use is http://<datapower host name>:3001/feed.

Requests sent to this firewall are sent unchanged to the atom XML firewall. Viewing the probe for the atom firewall, we can see that the first transform action sends a SOAP request to the Web service. The SOAP response is then translated by the second transform into the Atom format, as shown in Example 5-5.

Example 5-5 Translated SOAP response

```
<atom:feed xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:atom="http://www.w3.org/2005/Atom" xmlns:date="http://exslt.org/dates-and-times"
xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:dp="http://www.datapower.com/schemas/management">
<atom:id>atom.sample.com</atom:id>
<atom:title>feed demo: List of Customers</atom:title>
<atom:updated>2007-11-14T22:56:58+11:00</atom:updated>
<atom:entry>
<atom:id>0</atom:id>
<atom:title>customer0</atom:title>
<atom:updated>2007-11-14T22:56:58+11:00</atom:updated>
<atom:content>
<Customer xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<CustomerID>0</CustomerID>
<CustomerName>customer0</CustomerName>
</Customer><
/atom:content>
</atom:entry>
</atom:feed>
```

The Atom content is then returned to the atom-mime firewall, where the response-processing rule is invoked. Viewing the response probe for the atom-mime firewall, we can see that the Content-Type header has been updated by the set variable action from application/text+xml to application/atom+xml.

5.3.4 Summary

This section demonstrated how the DataPower appliances can be used to enable existing applications to support Web 2.0.technologies. Web 2.0 entities such as blogs, wikis, and social networking are becoming increasingly popular as a means for both individuals and companies to interact with one another.

Using a simple XSL transformation, we have shown how SOAP-enabled Web services can provide content in the Atom Syndication Format. This format is widely used to provide syndicated news and other informational content to users. The XSL transformation performed by the DataPower appliance converted the Web service's SOAP-based response into the Atom format by adding the necessary Atom metadata around the Web service response.

5.4 DataPower scenarios

In this section we examine four different scenarios for using DataPower as part of an ESB. These scenarios illuminate common cases for using DataPower and show both how the scenario is implemented with the DataPower technology and why it is the best choice in each case.

1. DataPower in a complex ESB with WESB—ABC Hotel
2. DataPower in a complex ESB with WMB—XYZ Insurance
3. DataPower as a Web services Gateway—food products
4. DataPower as a simple standalone ESB—brokerage firm

5.4.1 ESB scenario: WESB, WSRR, Tivoli for ABC Hotel

ABC Hotel is a global hotel and leisure company. It offers luxury and upscale full-service hotels under several brand names. This includes over 900 owned, managed, or franchised hotels with over 230,000 rooms in inventory.

ABC initiated a large, multi-year IT project to replace their large, monolithic mainframe-based central reservations system (CRS) with one implemented as a set of distributed J2EE services. Drivers for this project were cost reduction (mainframe replacement) and increased business agility. The project was conceived largely as one that would employ concepts and doctrine of SOA. Services are to be deployed in two geographically distributed datacenters. Multiple instances of individual services will be deployed to facilitate system throughput and availability.

Notably, the CRS project was originally conceived without a general purpose ESB framework. As the project matured, the complexity of service integration rose to unacceptable levels. It imposed a large burden on architects, developers, and operations and was an inhibitor to successful completion of the CRS project. Thus, a decision was made to refactor to incorporate an ESB. ABC undertook an evaluation effort to select the proper course of action.

Summary of ESB requirements presented to IBM is presented below:

- Request routing and workload management: Multiple copies of individual services were deployed within the infrastructure to facilitate reliability, serviceability, and availability. The ESB should route client requests to appropriate versions and instances of requested services. The ESB should also make some provisions for workload management, distributing requests to identical service instances. This could be a simple load distribution algorithm, but there was strong interest in more active, endpoint-driven load workload management.
- Registry integration: ABC employed a UDDI-based service registry for a build-time catalog of services. There was no provision for a more active runtime service registry, but the requirement to support such a registry was understood.
- Mediation: ABC expected the ESB to mediate between clients and services. Service data is expressed in WSDL/XML, so there was no need to handle more esoteric data formats. Message validation (WS-I and XSD) as well as support for performance/transformation acceleration were key drivers.
- Transaction management: ABC had certain use cases that would require compensation and rollback of committed service requests (distributed transaction support in composite services). They desired support for rule-based, atomic, and process-driven transactions.
- Service choreography and workflow: Complex workflow support was a requirement for a small subset of service interactions. BPEL support was highly desirable for these cases. Support for business state machines, scheduling and timing of interactions, event correlation, and workflow involving human interaction was also preferred.
- Ability to facilitate integration with back-end systems: The ability to integrate directly with Siebel® and other back-end systems was a consideration.
- High performance/low latency: The ESB should have a minimal negative impact on overall performance and latency.
- Monitoring, managing, maintaining: Collectively, there were broad requirements for service level monitoring, collecting standard and custom metrics, fault detection, root cause analysis, reporting, and historical analysis. Automated operator alert and notification facilities were critical.
- Security: Integration with Tivoli Access Manager, LDAP, and other third-party security products was a requirement. Credential and key management capability, standards-based authentication, access-control, encryption/decryption, SSL acceleration, Sarbanes-Oxley (SOX) controls, and suitability for DMZ deployment were all considerations.
- SDLC: ABC uses Eclipse-based Rational Application Developer and Rational Software Architect. ESB development tools that fit well within their existing environment were preferred.
- Fast deployment: ABC had already invested considerably before choosing to refactor to include an ESB. Cost and speed of deployment were critical considerations.
- Reduced operational complexity: Reduction of risk and complexity was what drove ABC to refactor to an ESB-based architecture. ABC had a strong preference for a turnkey, low-complexity solution.

Fault with complex recovery scenario

In the scenario, envision a fault in a composite service interaction while fulfilling a client request from a property. Recovery from the fault is deemed significant enough to require a workflow that possibly includes human recovery tasks. The core ESB patterns do not change, although the fulfilling architecture will include additional components to enable this function.

Complex workflow with back-end integration

In this scenario, a create customer request is entered into a preferred guest tracking system. Business users want portions of this information entered in Siebel as part of this mediation. The ESB fulfills standard mediation of existing services while also including an update to an ISV back-end system. The transform-monitor-route ESB aggregate pattern applies here, but the fulfilling architecture utilizes multiple IT components to fulfill the service request.

Solution architecture overview

IBM proposed a hybrid ESB architecture consisting of both hardware and software ESB implementations. Each has different strengths in ESB functionality.

- Appliance
  - Optimized for very high performance, very low latency processing
  - Simplest configuration, deployment, and operation
  - Secure, tamper-proof
  - Offloads expensive processing from general-purpose platforms
- Software
  - Support application hosting, workflow support, and complex mediation
  - Application integration via adapters
  - Enables business activity monitoring

  IBM expects many customers to deploy hybrid architectures that leverage strengths of both approaches. The hybrid architecture is augmented with SOA service registry and management capabilities.
- Solution components
  - DataPower XI50 (hardware ESB)
  - WebSphere Process Server (includes WebSphere ESB)
  - WebSphere Business Integration Adapters (integration with back-end systems)
  - IBM Tivoli Composite Application Management for SOA (service monitoring and management)
  - WebSphere Service Registry and Repository (service registry)

Figure 5-40 illustrates the interaction of these components.

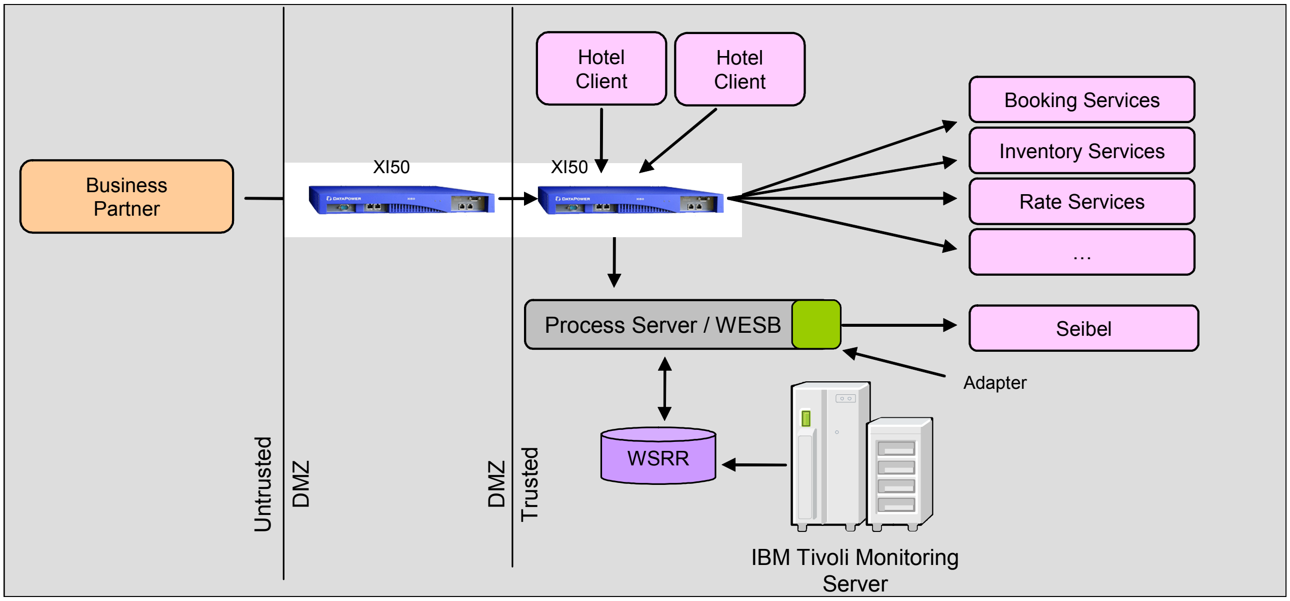


Figure 5-40 Complex integration scenario

The DataPower XI50 fulfills two roles in this architecture: Service gateway (in the DMZ) and general hardware ESB (in the trusted zone). Tasks that require a software ESB implementation (such as using a Siebel Adapter or complex error recovery with human interaction) are off-loaded to WebSphere Process Server includes as a core component WebSphere ESB.

DataPower fulfills the majority of ESB traffic requests at ABC, enabling very high performance. WebSphere Process Server and WebSphere ESB augment DataPower to allow maximum flexibility in the architecture.

IBM Tivoli Composite Application Management products enable monitoring and management of the ESB and components that interact with it. Information collected by monitoring agents can enable complex ESB enrichment and routing patterns, such as dynamic service selection (selecting a target service based on information collected by monitoring agents, such as hosting server load).

WebSphere Service Registry and Repository enables full life-cycle governance of services and plays a role in enabling dynamic service selection in the ESB. As noted above, UDDI alone is inadequate for these tasks.

This architecture is intended to strike a balance between performance and flexibility.

5.4.2 ESB scenario 2: XYZ Insurance

XYZ Insurance is a medical insurance provider. The company insures millions of customers and must interact with the customers, providers, third-party agencies, and others. This example is a composite of a number of real-world insurance projects.

XYZ decided to embark on an SOA project as a way to modernize their information technology while leveraging a substantial investment in mainframe (z/OS) applications written in COBOL, Assembler, and PL/I. These systems use DB2 and IMS databases and much of their communication with users is through Customer Information Control System (CICS). IBM

WebSphere MQ provides much of the intra-application and inter-application messaging backbone. There is a main data center connected to several satellite data centers and to third-party systems.

XYZ decided to use a service-oriented architecture to build a new call center environment (CCE) and to build a new claims processing application (CPA). Many other applications and changes took place, but we focus on these two major systems.

The overall project requirements that IBM and XYZ used to define the architecture included:

- Leverage traditional applications without requiring re-write of code.
- Leverage mainframe systems while creating new services on the platform of choice for that application.
- Support multiple communications protocols (SOAP/HTTM, WebSphere MQ, and so on).
- Securely support SOA interactions with outside organizations.
- Operate in an environment with several data centers, geographically dispersed.
- Provide the ability for real-time monitoring and routing of transactions based upon desired level of service.

The ESB requirements are similar to the ABC case study, except that more complex mediations are required. Also, due to many regulations, security and auditability must be more robust than in the ABC case study.

Fraud detection based on service call patterns

Fraud detection based on service call patterns is a complex case requiring monitoring of not only the number of calls, but also of what entity made the calls, and in what time frame. In this case, the information about a particular user making an abnormally high number of calls involving one customer or provider temporarily disables the user's access and provides notification so that XYZ can investigate the abnormal behavior. This monitoring, logging, correlation, and security system integration presents a significant challenge, but can be met with existing ESB technology, along with other SOA technologies.

Solution architecture overview

IBM and XYZ worked together to create a hybrid ESB architecture consisting of hardware and software ESB components. All three IBM ESB technologies were brought into play to provide the required functionality. The need for traditional integration, service choreography, and basic workflow drove the architecture decisions in using the technology set. Figure 5-41 illustrates the interaction of these components.

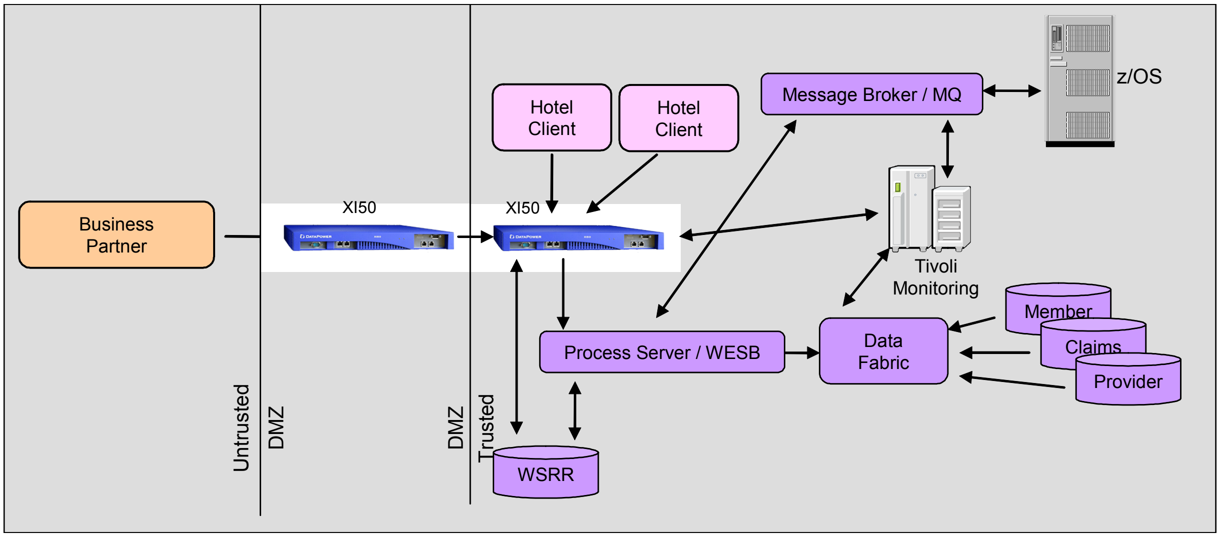


Figure 5-41 Complex ESB scenario with Message Broker

As in the ABC example, the DataPower XI50 plays two roles:

- Security firewall in the DMZ
- Hardware ESB in the trusted zone

WebSphere Process Server/ESB is used to choreograph complex service interactions, taking data from the data fabric and combining it into more complex business service replies. WebSphere Message Broker is used to handling complex message transformation needed to interact with traditional systems. This example shows the value of using all three IBM ESB solutions in one design, as each is leveraged to take advantage of its most significant capabilities.

IBM realizes that many organizations have similar complex needs to XYZ. These requirements will drive the use of several technologies in order to provide the most appropriate capabilities and meet availability, performance, and security requirements.

5.4.3 Food products corporation

In this scenario, a large international food products conglomerate was examining options for new marketing campaigns on the Web. The company had determined that pursuing a buy-over-build strategy (allowing them to outsource back-end functions) would be the most cost-effective. Thus, they chose to use business partners to provide services in support of their campaigns rather than providing them internally.

The goal of this effort was to reduce both the cost and time to market for new marketing campaigns. To accomplish these ends, they would need a way for new service providers to be easily *plugged* into the architecture, thereby enabling new features to be added with reduced development time. To support this, they would need simplified systems management in order to facilitate the production support process by providing visibility into complex integration points.

In evaluating this situation, the IBM team determined that there were several crucial elements that they had to meet in order to make the company successful. First, they had to provide a unified security infrastructure so that the interactions with their business partners would secure private customer information as well as analytic information about the campaign that was proprietary to the company. Second, they needed to be able to assess the success of the campaign and the ability of their business partners to support the campaign. Also, they needed to be able to quickly identify and fix problems as they occurred. This meant that the company would need a unified logging and usage analysis solution. Finally, they needed to be able to view the information about use statistics and performance in a common dashboard to get a view of the current status of the campaign at a glance.

Detailed description

The company has made an architectural decision that WS-Security is to be a critical part of the enterprise security strategy. They see the need for digital signatures, message-level encryption, and exchange of identity credentials in a standard way to be important in meeting the needs for customer privacy and protection of their own proprietary data. All new internal services will be deployed with WS-Security, and all new internal clients will be WS-Security enabled also. They would like to enforce the WS-Security basic profile as a common standard for all business partners as well.

However, not all of their business partners are yet able to fully take advantage of WS-Security. Existing services have often been deployed using channel-level encryption via HTTPs and using identity token exchange through HTTP BASIC-AUTH. Likewise, not all of their internal client programs have been brought up to speed on the newly adopted standards, either. Therefore, one need that the company has is to be able to bridge between channel-level encryption and credential exchange mechanisms and WS-Security.

Figure 5-42 shows the architectural needs that this client has and how they will be implemented in their infrastructure.

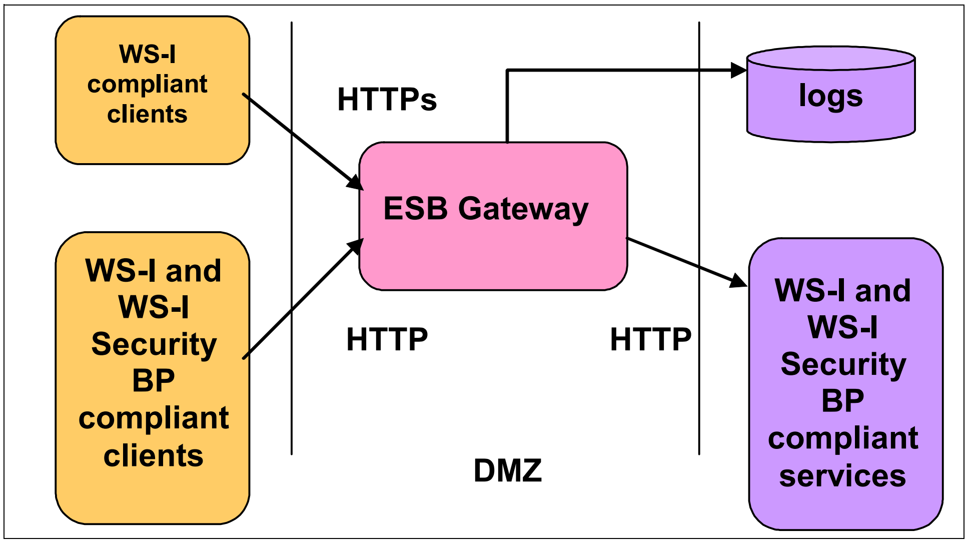

WS-I compliant clients
HTTPs
logs
ESB Gateway
WS-I and WS-I Security BP compliant clients
HTTP
HTTP
WS-I and WS-I Security BP compliant services
DMZ

Figure 5-42 Architectural elements

ESB functional elements

The functional capabilities of an ESB that applied to this particular scenario are:

- Security: The common security infrastructure deployed for this application uses several of the features described in the section on ESB features. The solution involves credential mapping, encryption, and decryption. AAA is not needed in this instance, as this is handled at the application-server level.
- Protocol translation: In this scenario there are only minimal requirements for protocol translation since all communications occur over HTTP and HTTPs. The only case to consider is switching from an incoming HTTPs transport to straight HTTP inside the firewall when we convert to using WS-Security.
- Message monitoring and logging: The common logging facility developed for this application is an excellent example of a message logging feature. Here, all messages should be logged for type and time spent in transit.
- Message transformation: We do not have any requirements for message transformation in this scenario.
- Routing: In this case there are no routing features needed, although this is a potential area of expansion. If several partners were to offer similar services, it would be possible to choose from those partners based on QoS and cost information.

Proposed DataPower implementation

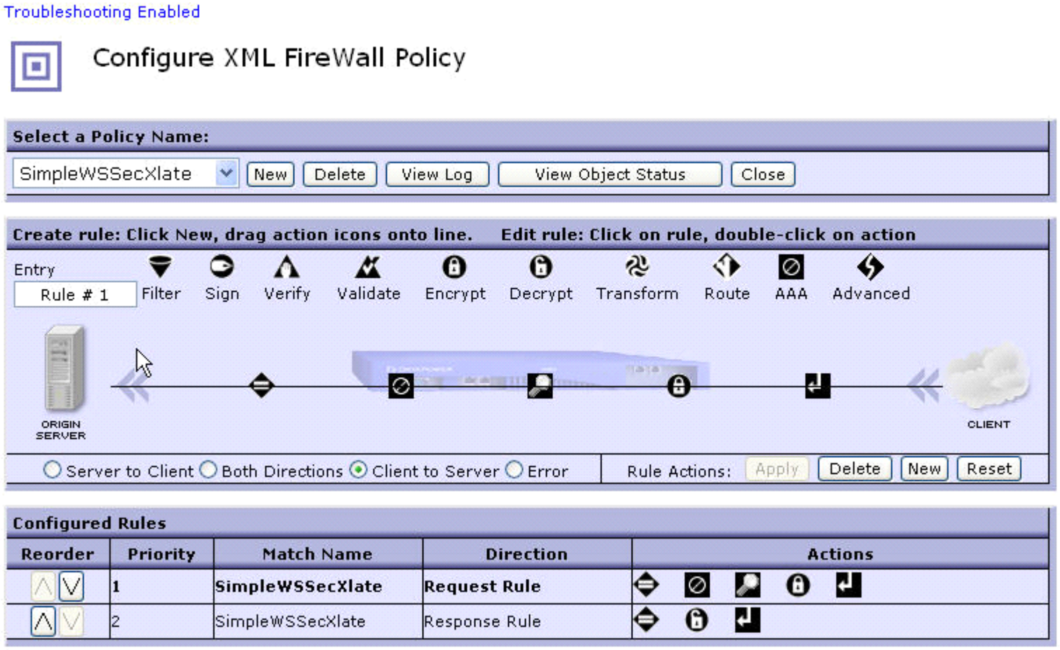
We have three major requirements:

- Support for WS-Security encryption and decryption
- Support for translation of WS-Security UserName tokens into HTTP BASIC-AUTH headers (and vice versa)
- Logging

It should come as no surprise that when we examine the implementation of each sub-scenario, they contain actions that correspond to these particular requirements. In particular, there are three cases that require support:

- Converting from an HTTP BASIC-AUTH client (communicating over SSL) to a server that uses WS-Security UserName tokens and WS-Security encryption
- Converting from a client that uses WS-Security UserName tokens and WS-Security encryption to a server that uses HTTP BASIC-AUTH headers and SSL encryption
- Simply logging the messages in the case where the client and server match up

We examine the first in detail, and the other two are derivative of the first case. We begin by examining the firewall processing policy definition for the first case, shown in Figure 5-43.



Troubleshooting Enabled

Configure XML FireWall Policy

Select a Policy Name: SimpleWSSecXlate

Rule # 1

○ Server to Client ○ Both Directions ● Client to Server ○ Error

Configured Rules

Reorder	Priority	Match Name	Direction	Actions
∧ ∨	1	**SimpleWSSecXlate**	**Request Rule**	
∧ ∨	2	SimpleWSSecXlate	Response Rule	

Figure 5-43 Firewall processing policy

The highlighted rule (the rule for processing requests) has five steps, or actions:

- Action 1 is the match action. This allows the integration engineer to define the cases under which this processing policy will match requests that enter the XML firewall. The default case (which we have taken) is to match all URLs that arrive at this port. This allows this XML firewall to act as a proxy for multiple services represented by different URLs.
- Action 2 is the authenticate, authorize, and audit (AAA) rule. It is used by DataPower to handle cases where authentication, authorization, or credential mapping is needed. In our case, this action is set up to pull the user credentials off of the HTTP BASIC-AUTH headers, authenticate them against LDAP, and then map the credentials into a WS-Security UserName token for use by the server.
- Action 3 is the log action. It takes the current context (the message received) and logs it to a predefined destination. There are a number of different choices for log destinations, including common destinations like a UNIX syslogd, but also supporting destinations like the common event infrastructure.
- Action 4 is the encryption action. It can encrypt all or part of the message using basic XML Encryption or (as in our case) WS-Security encryption.
- Action 5 is the return action, which allows the processing policy to return the newly changed message to its destination.

On the response side, there are only three actions:

- A match rule
- A decryption action (which decrypts the response message)
- A return rule to return the now decrypted response to the sending service requestor

5.4.4 Brokerage firm

In this scenario, a large brokerage firm has requirements that are parallel to those of the previous scenario, but that have an additional wrinkle. Not only do they require WS-Security support and logging support, but also they have a need for complex routing of requests.

Detailed description

This brokerage firm has recently adopted Web services as a middleware standard for internal server-to-server communications, replacing an internally developed set of protocols. However, the adoption of Web services standards is proceeding slowly, and some aspects of the traditional architecture remain in the current architecture.

In particular, this firm has the following problem. They have a set of services implemented on the mainframe as HTTP Web services. In order to take maximum advantage of their current hardware setup, they are split across multiple CICS regions. Thus, each Web service has been deployed on more than one CICS region. This means that, to take full advantage of the available hardware, requests to each Web service should be load balanced across the various regions. However, there are additional considerations. Due to the way that these services have been implemented internally, there is a need for client affinity to particular CICS regions hosting the services. Complicating this is that the set of regions is somewhat dynamic. It changes occasionally as additional regions are brought online in response to increasing demand.

So, when a new client request in a session begins, the client is assigned to one of the existing regions in order to provide load balancing. But all requests from that client should then be directed to the same CICS region in order to allow that region to maximize its performance through caching of expensive configuration data. Likewise, there are additional complications in the routing of messages to these regions. The algorithm dictates these steps:

1. Look in the routing header on the SOAP message to determine whether the destination has already been determined (True if this is not the first message from this client).
2. Look at the system context. If the system is currently in alarm mode, then turn back certain messages of low priority.
3. Otherwise, either generate a destination for a new client, or route the message to the previously determined destination.

The architectural elements of this problem are shown in Figure 5-44.

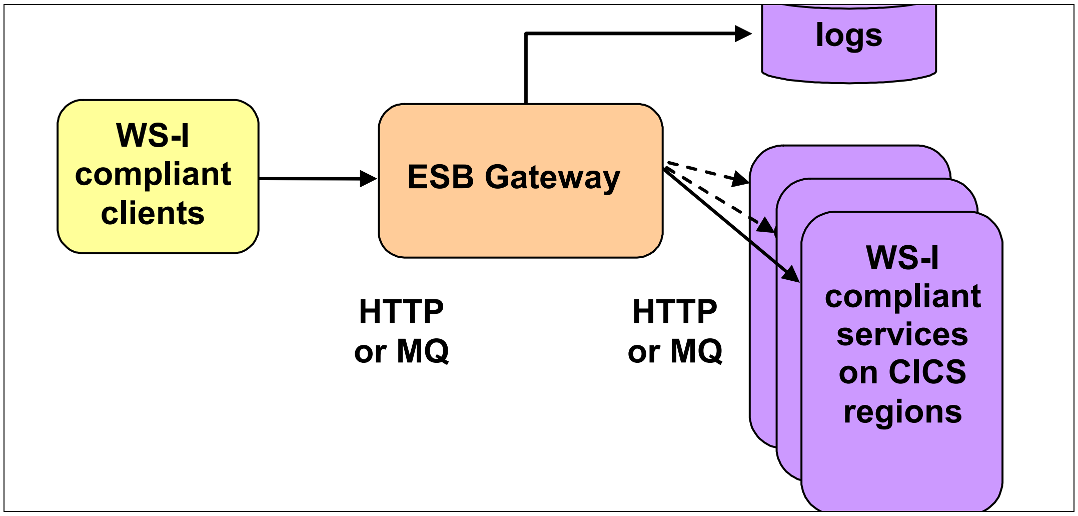


Figure 5-44 Architectural elements

ESB functional elements

The ESB functional elements are:

- Security

  The common security infrastructure deployed for this application will use credential mapping, encryption, and decryption. Just as in the previous scenario, AAA is not needed in this instance, as this is handled at the application-server level.

- Protocol translation

  In this scenario there are no requirements for protocol translation since all communications occur over HTTP and HTTPs. However, there is a future need to support SOAP over JMS as well as HTTP.

- Management and monitoring

  The common logging facility developed for this application is an excellent example of a monitoring feature. Here, all messages should be logged for type and time spent in transit.

- Message transformation

  In this case, the only message transformation features needed are in the development of the affinity token headers (see below).

- Routing

  Routing is the most important feature used in this scenario, as it is critical to meeting the performance needs of the application.

Proposed DataPower implementation

In this example, we combine the policy (flow) defined in the previous example with a new set of actions that filters the message when the system context is in alarm mode and determines the routing of the message in the other cases.

Since this set of requirements is derivative of the requirements of the previous scenario, it should come as no surprise that the solution is similar to that of the previous scenario as well. In fact, there's only one new action to introduce in this solution, the routing action.

The routing action allows us to choose the endpoint of a service dynamically. You must first set up the XML firewall to use a dynamic back end in order for this to function, but otherwise no changes are needed to the set up of the XML firewall from the previous example.

When considering how to perform the routing, there are three options to set the endpoint in the routing action:

- The routing action can set the endpoint based on the contents of a variable (for example, it will use the results of an earlier processing step).
- The routing action can set the endpoint based on an XPath Routing Map. Here, you choose from N static destinations based on evaluating a set of XPath expressions from information about the current context. This would be useful, for instance, when you would use the content of a header to choose a particular endpoint.
- The routing action can invoke an XSL Stylesheet that uses the DataPower extension function <dp:set-target> to set the endpoint.

In most cases we set the routing based on the content of a header, so the XPath Routing Map seems attractive. However, it cannot handle the case where the list of endpoints changes dynamically. Thus, the best solution would be to use the stylesheet option. In the stylesheet we could both obtain the list of possible destinations (from a context variable, or dynamically through a <dp:soapcall> invocation) and route to the destinations as needed.

Another transformation (earlier in the flow) can determine whether the destination header is set (for example, whether it is the first time that we have received a request). If not, then it can pick a destination and add the new header to the context.

5.4.5 Resources

For further information see “Enterprise Service Bus implementation patterns” at:

http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html

6

Configuration management

This chapter discusses several methods for managing the configuration of DataPower devices. We can leverage the way the system itself works to make it self-configuring. This allows for rapid swapping of boxes within a production environment, should the need arise. Topics include:

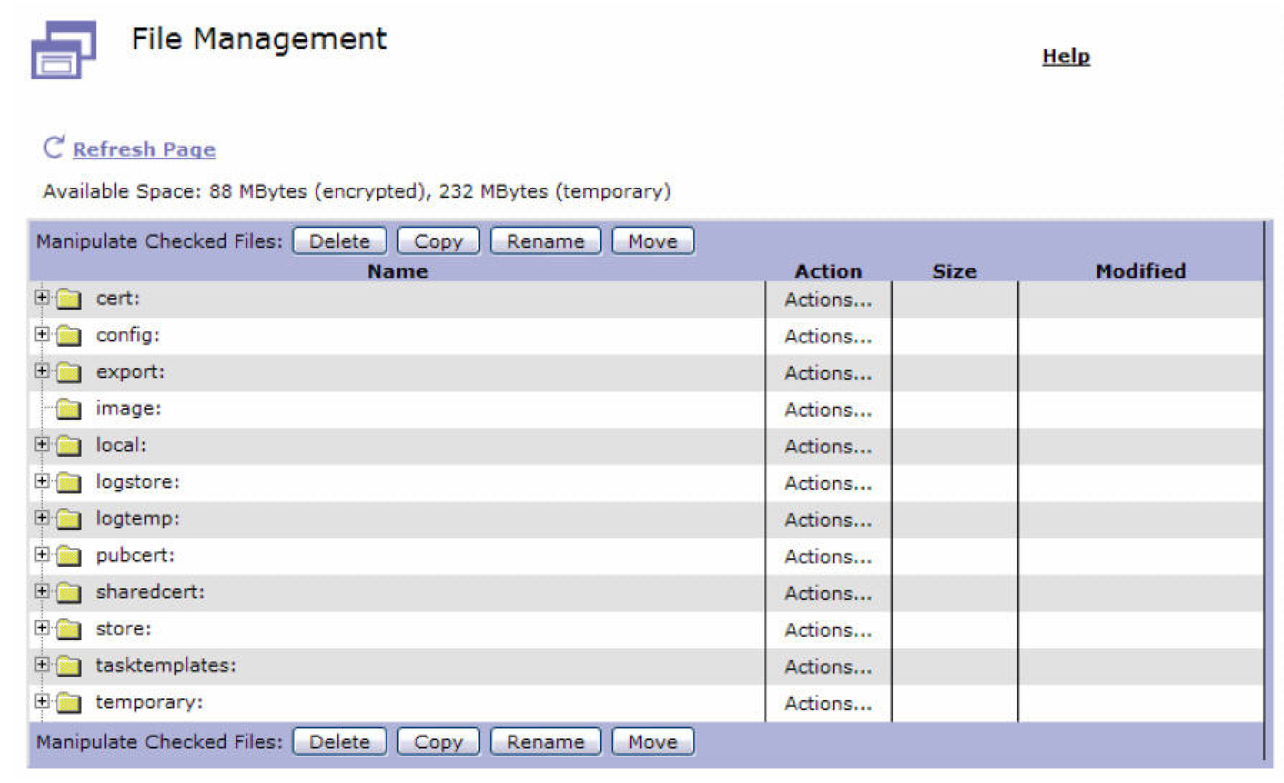
- DataPower File system directories and domains
- Devices, environments, and load balancers
- Configuration using the WebGUI, the command-line interface, and the XML Management Interface
- Package importing and exporting
- Using a repository
- Using IBM Tivoli Composite Application Manager System Edition (ITCAM SE), also known as ITCAM Systems Edition for DataPower

6.1 Introduction

Each DataPower device contains a configuration. The device is configured using objects that are hierarchically organized into services. It is the services that expose ports for the consumption of traffic over supporting protocols such as HTTP, FTP, MQ, JMS, and NFS. Services implement functionality such as authentication and authorization of Web services, acceleration of XSLT Transformations, and enterprise service bus protocol mediation.

6.1.1 File system directories and domains

The DataPower file system is an encrypted RAM data source separated into several named directories. See Figure 6-1. Directories are used to manage configuration data, store XSLT stylesheets, capture logging events, manage cryptographic certificates and keys, and control other system functions. Configuration files are stored in the config directory, while custom data maintained by the user are stored in the local directory. The device stores most of its required files in a directory called store;. For a complete description of the file system see the WebGUI Guide for your device.



Available Space: 88 MBytes (encrypted), 232 MBytes (temporary)

Name	Action	Size	Modified
cert:	Actions...		
config:	Actions...		
export:	Actions...		
image:	Actions...		
local:	Actions...		
logstore:	Actions...		
logtemp:	Actions...		
pubcert:	Actions...		
sharedcert:	Actions...		
store:	Actions...		
tasktemplates:	Actions...		
temporary:	Actions...		

Figure 6-1 DataPower file system

6.1.2 Devices, environments, and load balancers

The DataPower file system is initially booted as a single data space within a domain or partition labeled *default*. The file system may be further partitioned with the creation of application domains. A newly created domain will be provided with its own local directory, and will inherit read access to DataPower configuration rooted in the store directory of the default domain. Access to other application domains may also be granted.

Each DataPower device may participate as a member of a peer group that provides services and shares management information with other members for the group. For the purpose of this document, a shared *environment* of devices refers to the group of devices servicing software life-cycle areas such as test, development, or production. Migrating configuration information to an environment refers to the entire group of devices.

It is often the case that groups of devices, perhaps an entire environment, will reside behind a Load Balancer that acts as a façade exposing a single virtual IP address for request traffic. Common routing algorithms such as *round robin* and *least frequently used* are available. This topology has no effect on the configuration of the device. There will be no device affinity, as each transaction should be able to be processed by any of the devices in the load-balanced group. Service level data is shared by devices through peer group registration, and all transactions participate in service level monitoring.

DataPower devices may also be deployed in an active/active or active/standby configuration whereby a virtual IP address is used to control multiple devices. In this configuration the standby device monitors the health of the active device, and assumes its IP address in the event that it becomes unavailable. In an active/active configuration, multiple standby configurations are employed. Each device acts as the standby for the other, and assumes its traffic in the event of failure, without the need to maintain one box offline.

6.1.3 Boot sequence for DataPower

Before discussing the methods available for configuration management, it is important to describe what happens at boot time within a DataPower device.

When a DataPower appliance first boots, it loads a startup configuration (from config:///autoconfig.cfg, which can be modified using the **`boot config`** command) in the default domain. All the information in the configuration file is in command-line interface (CLI) format. It is executed *synchronously*, one line at a time, until initialization is complete. This is why all the box-level initialization precedes any other domain statements in this file.

Therefore, at the very least, the configuration file should define the IP addresses for each Ethernet port and DNS names for resolution. After that, domain definitions and other artifacts can be set. However, it is suggested that this file be kept small so that only critical values are placed within it to improve readability. Additional configuration files can be nested in a daisy-chain fashion as necessary to complete the configuration of the device.

Note: The lines in this file are executed in sequence, so basic properties like IP address must come before anything else. Object order is significant, too. Subordinate objects such as the match rule must be defined prior to parental objects such as the XML firewall.

Compare this to the SOMA/XML system, where the XML files may not necessarily be executed in a strict order. Using the CLI configuration files removes this dependency.

Example 6-1 shows a snippet from the configuration file of a simple XML firewall, and supporting style policy rule and action that performs XML threat protection.

Example 6-1 Example of ASCII configuration file

```
action simpleFirewall_Rule_0_Action_0
   type results
   input ‘INPUT’
   output-type default
exit
```

```
rule ‘simpleFirewall_Rule_0’
   action simpleFirewall_Rule_0_Action_0
exit
matching ‘matchAll’
   urlmatch ‘/*’
exit
stylepolicy ‘simpleFirewall’
   match ‘matchAll’ ‘simpleFirewall_Rule_0’
exit
xmlfirewall ‘simpleFirewall’
   local-address 0.0.0.0 2092
   summary ‘an example XML Firewall Service’
   query-param-namespace ‘http://www.datapower.com/param/query’
   remote-address 127.0.0.1 1001
   stylesheet-policy simpleFirewall
   response-type unprocessed
exit
```

6.2 Configuration options

There are several ways to configure the device. Those used will be dictated by solution requirements. They consist of graphical tools and automated/scripted and programmatic processes. These methods may be used in combination, and all methods result in a similar implementation on the device.

The WebGUI and the command-line interface through SSH are the two main ways of administering configurations. For every operation that can be done in the WebGUI, there is a corresponding operation in the CLI. See the Command Reference Guide for your device for details.

Regardless of the configuration method used, the end result of each operation is a modification to the onboard device configuration. This data is expressed as an ASCII file resident on the devices’ encrypted RAM file system, and contains a list of CLI commands. If the WebGUI or XML Management Interface was used, their execution is translated into the corresponding CLI commands. Upon restart of the device, the designated configuration file is loaded, and services and objects are reconstructed from its content and references to external resources. This onboard configuration file may itself be downloaded, edited, and reloaded onto the device.

Important: The WebGUI is typically used in the development phase of a project. Configurations are usually exported/imported using the other mechanisms (CLI, XML Management Interface) as one gets closer to the test phase and production staging. Migration methods are discussed further in the sections below.

6.2.1 WebGUI interface

The WebGUI interface is the simplest management interface to use. On most pages, a help link provides online help through a pop-up browser window. Most fields also provide in-line help when selected. Lastly, the two-step process for committing configuration changes provides an opportunity to discard changes. See the WebGUI Guide for your device for a complete description.

6.2.2 Command-line interface

The command-line interface provides a simple but powerful management interface. Its syntax should be familiar to terminal users on a UNIX-like environment. Unlike the WebGUI, configuration changes are immediately committed. An **undo** command allows administrators to revert to a previous configuration. All administrators should be familiar with basic CLI commands, as this management interface is the only one available on first use to enable one of the other management interfaces using the configure terminal command.

The CLI is a streamlined yet powerful system for controlling every facet of the appliance. Although it looks like a command shell, there is no compiler or interpreter to run arbitrary code. The alias function is a macro for multiple CLI commands, and the exec function executes configuration scripts limited to the device itself.

In the global configuration mode, the administrator can create, modify, or remove any DataPower service or interface that can be found in the WebGUI.

The CLI is not a true scripting facility like Perl, Tcl, or ANT. The alias CLI command does allow a simple form of defining a set of commands to execute. Automating the repetition of tasks in a scripting language is much easier than using a Web GUI. *Scriptin*g in this sense is the sequencing of commands. Example 6-2 shows a CLI shell file that imports a WS-Proxy to the demo application domain of the device.

Example 6-2 Command-line Interface shell file

```
ssh DataPowerIP<<EOF
admin
passw0rd
config
copy -f https://HTTPServerIP/HRServiceExport.zip
temporary:///Demo/HRServiceExport.zip
passw0rd
import-package HRServiceImport
auto-execute off
destination-domain Demo
source-url temporary:HRServiceExport.zip
exit
import-execute HRServiceImport
no import-package HRServiceImport
write mem
y
exit
exit
EOF
```

Tasks executed in Web GUI require the user to remember what they performed, especially when they need to undo actions. Administrators can create an undo script to ensure that changes are undone.

Many options exist for migrating changes from development to production. For example, using the Web GUI, administrators can create the domain configuration and store it on a Web server. The domain on the appliance can reference this configuration on startup. Another option for importing shared objects into a domain is to use packages. The package contains the set of shared objects used by several domains.

These methods can be used to migrate changes from development to production, but may not be preferred since they cannot be audited and the consistency in the files cannot be guaranteed.

6.2.3 XML Management Interface

The XML Management Interface provides a structured language for sending a batch of configuration commands. This interface allows for rapid automated configuration of new application domains or entire DataPower SOA Appliances. A complete Redpaper publication has been written about this interface: *WebSphere DataPower SOA Appliance: The XML Management Interface*, REDP-4446[1].

The XML Management Interface allows appliance management using different XML-based interfaces/specification. The SOAP Management URI (SOMA) interface enables management using SOAP over HTTPS. The SOMA interface provides advantages over other approaches. Because it is programmatic it can be used to build a custom management application using a programming language like Java. Also, it executes remotely without any tools while the WebGUI requires a Web browser and CLI requires an SSH client.

The SOAP interface extends its functionality to third-party Web service clients. It involves the programmatic modification of configuration data, and may be based on external conditions and events. For example, a brokerage organization may define a service level agreement that limits the number of trades that a client may execute in a given time frame. However, based on market conditions, the parameters of this SLA may change.

This demonstrates a potential use of the XML Management API. This facility allows for the authenticated real-time modification of configuration data via SOAP messages sent over a secured HTTPS interface. All the functions of the WebGUI and CLI are supported using the XML Management API. Each request is packaged as a SOAP message.

Formatting of request SOAP documents is derived via the WSDL and XSD documents available from the store directory of the device. It also provides configuration management tools. For information regarding request formats, refer to the WebGUI documentation for your device.

The SOAP request Example 6-3 shows the creation of a matching rule object via the XML Management Interface. Notice the testDomain attribute on the dp:request element.

Example 6-3 XML Management Interface request

```
<?xml version=‘1.0’ encoding=‘UTF-8’?>
<env:Envelope xmlns:env=‘http://schemas.xmlsoap.org/soap/envelope/’>
   <env:Body>
      <dp:request domain=‘testDomain’
xmlns:dp=‘http://www.datapower.com/schemas/management’>
         <dp:set-config>
            <Matching name=‘matchAny’
xmlns:dp=‘http://www.datapower.com/schemas/management’
xmlns:env=‘http://www.w3.org/2003/05/soap-envelope’>
               <MatchRules>
                  <Type>url</Type>
```

[1] http://www.redbooks.ibm.com/abstracts/redp4446.html?Open

```
                  <Url>.*</Url>
               </MatchRules>
            </Matching>
         </dp:set-config>
      </dp:request>
   </env:Body>
</env:Envelope>
```

WebSphere DataPower SOA Appliances contain a powerful management framework that is accessible through XML messages sent over HTTPs. These messages can be chained and scripted in order to perform automated configuration management and operations on the device.

In order to perform configuration management through the XML Management Interface, the interface must first be enabled through the CLI or WebGUI. After this is done, the XML commands can be sent to the specified host/port/URL. In the following two examples, we perform a substitution on the file to create a new domain with its own user and group defined.

Example 6-4   Initial XML file

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Body>
    <dp:request xmlns:dp="http://www.datapower.com/schemas/management">
      <dp:set-config>
        <Domain name="%domainName%">
          <UserSummary>%domainDesc%</UserSummary>
          <NeighborDomain class="Domain">%domainNeighbor%</NeighborDomain>
        </Domain>
        <UserGroup name="%groupName%">
          <UserSummary>%groupDesc%</UserSummary>
%groupPolicies%
        </UserGroup>
        <User name="%userName%">
          <Password>%userPass%</Password>
          <GroupName>%groupName%</GroupName>
          <AccessLevel>%userAccess%</AccessLevel>
          <UserSummary>%userDesc%</UserSummary>
        </User>
      </dp:set-config>
    </dp:request>
  </env:Body>
</env:Envelope>
```

Example 6-5 is a shell script that can make the substitutions.

Example 6-5   lShell script file

```
#s!/bin/ksh
domainName=${1:-'testDomain'}
domainDesc=${2:-'testDomain for applications'}
domainNeighbor=${3:-'default'}

userName=${4:-'testUser'}
userPass=${5:-'passw0rd'}
userDesc=${6:-'test user for this domain'}
```

```
userAccess=${7:-'group-defined'}

groupName=${8:-'testGroup'}
groupDesc=${9:-'testGroup for application domain'}
groupPolicy01=${10:-"*/$domainNeighbour/*?Access=r"}
groupPolicy02=${11:-"*/$domainName/*?Access=r+w+a+d+x"}
groupPolicies="
<AccessPolicies>$groupPolicy01</AccessPolicies>
<AccessPolicies>$groupPolicy02</AccessPolicies>
"
sed -e "s/%domainName%/$domainName/" \
-e "s/%domainDesc%/$domainDesc/" \
-e "s/%domainNeighbor%/$domainNeighbor/" \
-e "s/%userName%/$userName/" \
-e "s/%userName%/$userName/" \
-e "s/%userDesc%/$userDesc/" \
-e "s/%userAccess%/$userAccess/" \
-e "s/%groupName%/$groupName/" \
-e "s/%groupPolicies%/$grouppolicies/" \
-e "s/%groupDesc%/$groupDesc/" sampleTemplate.xml > domain.xml
```

This newly created file (domain.xml) can then be sent to the XML Management Interface using the curl utility:

```
curl --data-binary @domain.xml https://<IP>:5550/service/mgmt/current -u
admin:<pass> -k
```

Where <IP> is the address of your DataPower device and <pass> is your admin password for this device.

6.3 Package importing and exporting

The DataPower device supports a variety of methods for the exporting and importing of configuration details. While the WebGUI offers the simplest methods, the XML Management Interface and the CLI provide the basis for an automated, scripted device configuration management capability.

There are recommended methods that may be employed to minimize the need to modify configuration during migration.

> **Note:** In order to provide for a cleaner configuration and one that minimizes migration issues, we recommend that host alias be used instead of dot decimal address in services that expose external ports. Also, use an environment-specific DNS when possible rather than a dot decimal address, and use static hosts to handle DNS aberrations. In order to eliminate extra work, migrate only those objects that require migration.

For a more in-depth discussion of configuration for high availability, configuration promotion, and control, see the following article by Rasmussen:

http://www.ibm.com/developerworks/websphere/library/techarticles/0801_rasmussen/0801_rasmussen.html

6.4 Using a repository

To store configuration files in a repository external to the DataPower device, we have the option of pulling files from a Web server or pushing them to the DataPower boxes with an automated process. Pull is preferred since it is well supported by DataPower and merely requires a valid URL. Push can be accomplished in several ways:

- Manually through file upload in the Web GUI
- By SSH
- With the XML Management Interface

This last option requires converting each file to base 64 and encapsulating it in an XML body before passing it to the XML Management Interface.

Configuration data can be pulled from a repository and managed centrally to the place where boxes can be deployed. This requires a minimal autoconfig.cfg file that sets up the IP addresses and other values specific to the device and then pulls in further configuration information as required. Example 6-6 shows a simple autoconfig.cfg file.

The configurations could be generated on-the-fly by a Web server to tailor the specific details to the subnet requesting the configuration files. This is a trivial exercise, but it would also be possible to manage configuration files through CVS systems and then generate the actual deployment files on a nightly basis ready to be served.

> **Tip:** For different topologies between production, test, and development, empty configuration files can be used as placeholders so that the method remains the same throughout the entire enterprise infrastructure.

The following command-line interface statements demonstrate a very simple default domain configuration file that could be uploaded to the config directory of the device and identified as the boot config on the system control panel. It defines the Ethernet Interface, host aliases, system contact information, the WebGUI, and the XML Management Interface. Finally, the configuration executes an off-box configuration that contains configuration information shared by all devices in the environment.

Example 6-6 Autoconfig file

```
configure terminal
# This is the fixed component of the 'Default' Domain for DataPower device
192.168.0.52
interface "eth0"
  ip address 192.168.1.50/24
  arp
  mtu 1500
  ip default-gateway 192.168.0.1
  mode 1000baseTx-FD
exit
interface "eth1"
  ip address 192.168.1.51/24
  mtu 1500
  ip default-gateway 192.168.0.1
  arp
  mode 1000baseTx-FD
exit
system
  contact "itso@us.ibm.com"
  name "DataPower"
  location "DataCentre"
exit
host-alias "authenticatedTraffic"
  reset
  ip-address 192.168.1.50
exit
host-alias "boundryTraffic"
  reset
  ip-address 192.168.1.51
exit
# Web and XML-MGMT Interfaces can be restricted to a INT, so put them in Fixed
Config
web-mgmt
  admin-state enabled
  local-address 0.0.0.0 9090
  idle-timeout 6000
exit
xml-mgmt
  admin-state enabled
  local-address 0.0.0.0 5550
  mode any+soma+v2004+amp+slm
exit
# Now pull in the Variable part of the configuration, this is the same for all
devices in this environment, i.e. Dev/Test/Prod

exec http://9.33.97.230/dp/prod/defaultDomainVariable.cfg
```

The defaultDomainVariable.cfg file can itself include other configuration files, in order to completely compartmentalize (separate) each application domain. An example defaultDomainVariable.cfg file is shown in Example 6-7.

Example 6-7 DefaultDomainVariable configuration file

```
# This is the variable component of the 'Default' Domain for all DataPower
devices. It is "exec"ed into the device via the defaultDomainFixed.cfg
configuration
dns
  admin-state enabled
  search-domain "ibm.com"
  name-server 152.155.21.10 53 53 0 3
  name-server 152.155.21.60 53 53 0 3
exit
alias "reload" "flash;boot config autoconfig.cfg;shutdown reload"
network
  admin-state enabled
  icmp-disable Timestamp-Reply
exit
ntp-service
  admin-state disabled
  remote-server 152.159.20.10
  remote-server 152.159.20.60
exit
timezone EST5EDT
snmp
  admin-state enabled
  ip-address 192.168.0.52
  community "public" "default" "read-only" "0.0.0.0/0"
exit
ssh 0.0.0.0 22
save-config overwrite
domain "prodDomain"
  reset
  base-dir local:
  base-dir prodDomain:
  config-file prodDomain.cfg
  visible-domain default
  url-permissions http+https+snmp+ftp+mailto+mq
  file-permissions CopyFrom+CopyTo+Delete+Display+Exec+Subdir
  config-mode import
  import-url "http://9.33.97.230/dp/prod/prodDomain.xcfg"
  import-format XML
exit
```

6.5 IBM Tivoli Composite Application Manager System Edition

IBM Tivoli Composite Application Manager System Edition (ITCAM SE) is a graphical user interface (GUI) to manage firmware updates and multiple boxes. ITCAM SE for WebSphere DataPower is available as a download at no additional charge[2]. It provides robust multibox management capabilities for DataPower SOA Appliances by enabling appliance administrators to efficiently roll out, update, and manage configurations on multiple DataPower appliances.

ITCAM SE for WebSphere DataPower can be used to assist in the migration of configurations across devices in a shared environment. Using this tool, a configuration installed on a master device can be deployed across all dependent devices in an automated fashion. Therefore, only the master device need be maintained. The configuration management techniques described in this document still apply when using ITCAM SE for DataPower.

> **Note:** Examples of ITCAM SE can be found in *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366.

[2] Download from http://www-306.ibm.com/software/integration/datapower/itcamse.html

A

The enterprise service bus

Successfully implementing an service-oriented architecture (SOA) requires applications and an infrastructure that can support the service-oriented architecture principles. Applications can be enabled for SOA by creating service interfaces to existing or new functions. The service interfaces should be accessed using an infrastructure that can route and transport service requests to the correct service provider. As organizations expose more and more functions as services, it is vitally important that this infrastructure should support the management of SOA on an enterprise scale.

The enterprise service bus (ESB) is a middleware infrastructure component that supports the implementation of SOA within an enterprise. Figure A-1 illustrates where the enterprise service bus fits in the SOA reference architecture.

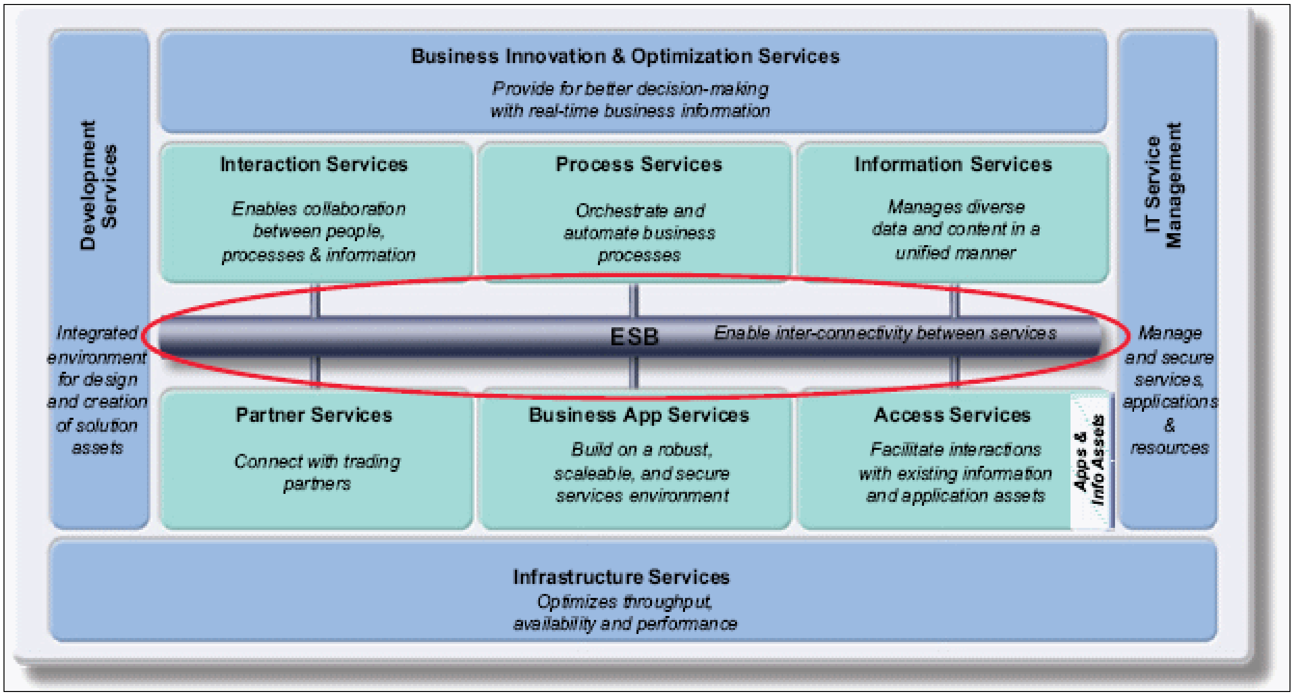

Business Innovation & Optimization Services
Provide for better decision-making with real-time business information
Development Services
Integrated environment for design and creation of solution assets
Interaction Services
Enables collaboration between people, processes & information
Process Services
Orchestrate and automate business processes
Information Services
Manages diverse data and content in a unified manner
IT Service Management
Manage and secure services, applications & resources
ESB
Enable inter-connectivity between services
Partner Services
Connect with trading partners
Business App Services
Build on a robust, scaleable, and secure services environment
Access Services
Facilitate interactions with existing information and application assets
Apps & Info Assets
Infrastructure Services
Optimizes throughput, availability and performance

Figure A-1 IBM reference architecture

Fundamentally, an enterprise service bus is a flexible connectivity infrastructure for integrating applications and services. An ESB performs the following between requestor and service:

- Converting transport protocols between requestor and service
- Handling business events from disparate sources
- Routing messages between services
- Transforming message formats between requestor and service

The need for an ESB can be seen by considering how it supports the concepts of SOA implementation by:

- Decoupling the consumer's view of a service from the actual implementation of the service
- Decoupling technical aspects of service interactions
- Integrating and managing services in the enterprise

Decoupling the consumer's view of a service from the actual implementation greatly increases the flexibility of the architecture. It allows the substitution of one service provider for another (for example, because another provider offers the same services for lower cost or with higher standards) without the consumer being aware of the change or without the need to alter the architecture to support the substitution.

This decoupling is better achieved by having the consumers and providers interact via an intermediary. Intermediaries publish services to consumers. The consumer binds to the intermediary to access the service, with no direct coupling to the actual provider of the service. The intermediary maps the request to the location of the real service implementation.

In an SOA, services are described as being loosely coupled. However, at implementation time, there is no way to loosely couple a service or any other interaction between systems.

The systems must have some common understanding to conduct an interaction. Instead, to achieve the benefits of loose coupling, consideration should be given to how to couple or decouple various aspects of service interactions, such as the platform and language in which services are implemented, the communication protocols used to invoke services, or the data formats used to exchange input and output data between service consumers and providers.

Further decoupling can be achieved by handling some of the technical aspects of transactions outside of applications. This could apply to aspects of interactions such as:

- How service interactions are secured
- How the integrity of business transactions and data is maintained (for example, through reliable messaging, the use of transaction monitors, or compensation techniques)
- How the invocation of alternative service providers is handled in the event that the default provider is unavailable

These aspects imply a need for middleware to support an SOA implementation. Some of the functions that might be provided by the middleware are:

- Map service requests from one protocol and address to another.
- Transform data formats.
- Support a variety of security and transactional models between service consumers and service providers and recognize that consumers and providers might support or require different models.
- Aggregate or disaggregate service requests and responses.
- Support communication protocols between multiple platforms with appropriate qualities of service.
- Provide messaging capabilities such as message correlation and publish/subscribe to support different messaging models such as events and asynchronous request/response.

This middleware support is the role of an ESB.

Definition of an enterprise service bus

An ESB provides an infrastructure that removes any direct connection between service consumers and providers. Consumers connect to the bus and not the provider that actually implements the service. This type of connection further decouples the consumer from the provider. A bus also implements further value add capabilities. For example, security and delivery assurance can be implemented centrally within the bus instead of having this buried within the applications.

Integrating and managing services in the enterprise outside of the actual implementation of the services in this way helps to increase the flexibility and manageability of SOA.

The primary driver for an ESB, however, is that it increases decoupling between service consumers and providers. Protocols such as Web services define a standard way of describing the interface to a service provider that allow some level of decoupling (as the actual implementation details are hidden). However, the protocols imply a direct connection between the consumer and provider.

Although it is relatively straightforward to build a direct link between a consumer and provider, these links can lead to an interaction pattern that consists of building multiple point-to-point links that perform specific interactions. With a large number of interfaces this quickly leads to the buildup of a tangle of links with multiple security and transaction models. Routing control is distributed throughout the infrastructure, and probably no consistent approach to logging, monitoring, or systems management is implemented. This environment is difficult to manage or maintain and inhibits change.

A common approach to reducing this complexity is to introduce a centralized point through which interactions are routed, as shown in Figure A-2.

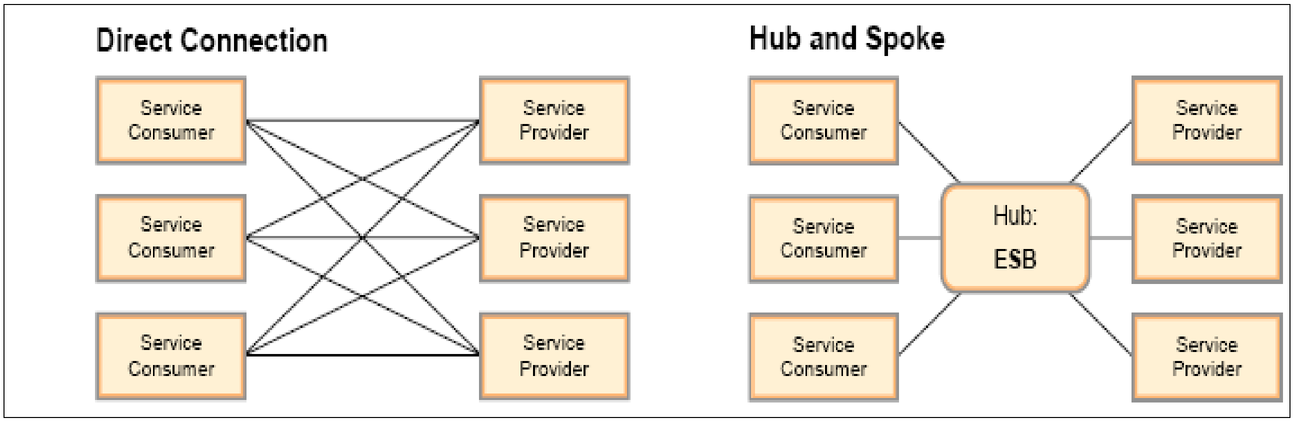


Figure A-2 Direct connection versus hub and spoke connection

A hub and spoke architecture is a common approach that is used in application integration architectures. In a hub, the distribution rules are separated from applications. The applications connect to the hub and not directly to any other application. This type of connection allows a single interaction from an application to be distributed to multiple target applications without the consumer being aware that multiple providers are involved in servicing the request. This connection can reduce the proliferation of point-to-point connections.

Note that the benefit of reducing the number of connections only truly emerges if the application interfaces and connections are genuinely reusable. For example, consider the case where one application needs to send data to three other applications. If this is implemented in a hub, the sending application must define a link to the hub, and the hub must have links that are defined to the three receiving applications, giving a total of four interfaces that need to be defined. If the same scenario was implemented using multiple point-to-point links, the sending application would need to define links to each of the three receiving applications, giving a total of just three links. A hub only offers the benefit of reduced links if another application also needs to send data to the receiving applications and can make use of the same links as those that are already defined for the first application. In this scenario, the new application only needs to define a connection between itself and the hub, which can then send the data correctly formatted to the receiving applications.

Hubs can be federated together to form what is logically a single entity that provides a single point of control but is actually a collection of physically distributed components. This is commonly termed a bus. A bus provides a consistent management and administration approach to a distributed integration infrastructure.

Enterprise requirements for an ESB

Using a bus to implement an SOA has a number of advantages. In an SOA services should, by definition, be reusable by a number of different consumers, so that the benefits of reduced connections are achieved. In addition, the ESB:

- Supports high volumes of individual interactions.
- Supports more established integration styles, such as message-oriented and event-driven integration, to extend the reach of the SOA. The ESB should allow applications to be SOA enabled either directly or through the use of adapters.
- Support centralization of enterprise-level qualities of service and manageability requirements into the hub.

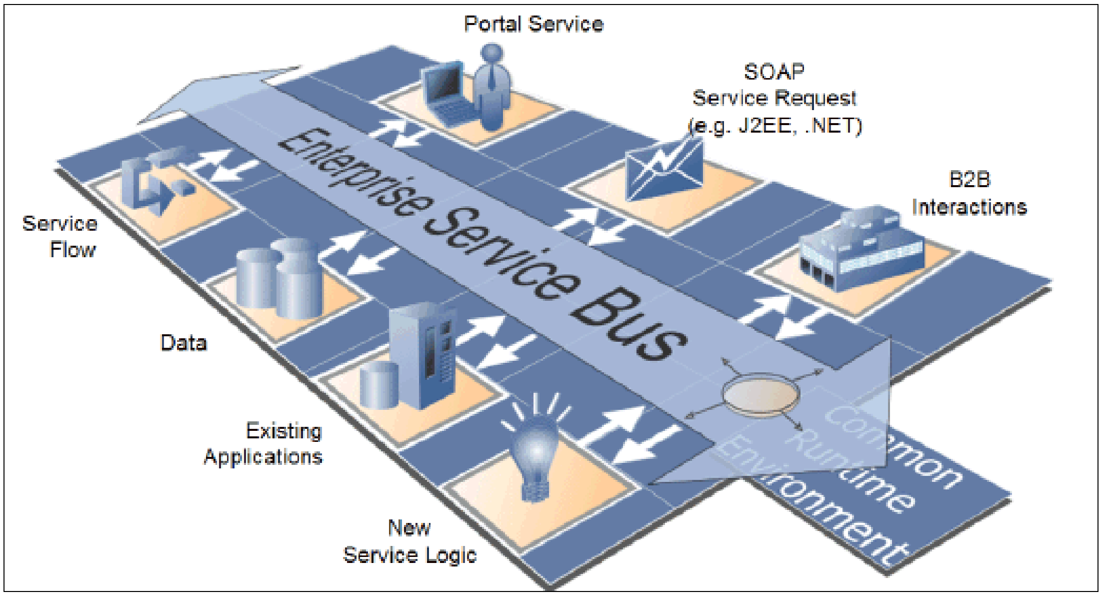


Figure A-3 High-level view of the ESB

SOA applications are built from services. Typically, a business service relies on many other services in its implementation. The ESB is the component that provides access to the services and so enables the building of SOA applications.

Mediation support

The ESB is more than just a transport layer. It must provide mediation support to facilitate service interactions (for example, to find services that provide capabilities for which a consumer is asking or to take care of interface mismatches between consumers and providers that are compatible in terms of their capabilities). It must support a variety of ways to get on and off the bus, such as adapter support for existing applications or business connections, that enable external partners in business-to-business interaction scenarios. To support these different ways to get on and off the bus, it must support service interaction with a wide variety of service endpoints. It is likely that each endpoint will have its own integration techniques, protocols, security models, and so on.

This level of complexity should be hidden from service consumers. They need to be offered a simpler model. In order to hide the complexity from the consumers, the ESB is required to mediate between the multiple interaction models that are understood by service providers and the simplified view that is provided to consumers.

Protocol independence

As shown in Figure 2-3 on page 15, services can be offered by a variety of sources. Without an ESB infrastructure, any service consumer that needs to invoke a service needs to connect directly to a service provider using the protocol, transport, and interaction pattern that is used by the provider. With an ESB, the infrastructure shields the consumer from the details of how to connect to the provider.

In an ESB, there is no direct connection between the consumer and provider. Consumers access the ESB to invoke services, and the ESB acts as an intermediary by passing the request to the provider using the appropriate protocol, transport, and interaction pattern for the provider. This intermediary connection enables the ESB to shield the consumer from the infrastructure details of how to connect to the provider. The ESB should support several integration mechanisms, all of which can be described as invoking services through specific addresses and protocols, even if in some cases the address is the name of a CICS transaction and the protocol is a J2EE resource adapter integrating with the CICS Transaction Gateway. By using the ESB, the consumers are unaware of how the service is invoked on the provider.

Because the ESB removes the direct connection between service consumer and providers, an ESB enables the substitution of one service implementation by another with no effect to the consumers of that service. Thus, an ESB allows the reach of an SOA to extend to non-SOA-enabled service providers. It can also be used to support migration of the non-SOA providers to using an SOA approach without impacting the consumers of the service.

B

XML security

XML threats

To truly harden a system using Web services, you need to perform several important security steps, including:

- Inspect messages for well-formedness.
- Validate schema.
- Verify digital signatures.
- Sign messages.
- Implement service virtualization to mask internal resources via XML transformation and routing.
- Encrypt data at the field level.

An article by Bill Hines (http://www.ibm.com/developerworks/websphere/techjournal/0603_col_hines/0603_col_hines.html) concludes with the following comments.

We consider four broad classifications of XML threats:

- XML of Service (xDOS): Slowing down or disabling a Web Service so that valid service requests are hampered or denied.
- Unauthorized Access: Gaining unauthorized access to a Web Service or its data.
- Data Integrity/Confidentiality: Attacks that strike at data integrity of Web Service responses, requests or underlying databases:
- System Compromise: Corrupting the Web Service itself or the servers that host it.

These threats may be facilitated by tricky/complex XML, virus-laden XML/SOAP attachments, and so on.

Specific types of XML attacks

These attacks can be single message or multiple message. There are several types of attacks within these four broad classifications. Several of the attack types listed below are familiar. These same types of attack occur with any remotely accessible service (for example, message tampering), but other attacks, while not really unique to XML-based services, are much more likely with such services given the nature of XML.

Single-message xDOS attacks

Single-message xDOS attacks are:

- Jumbo payloads: Sending a very large XML message to exhaust memory and CPU on the target system
- Recursive elements: XML messages that can be used to force recursive entity expansion or other repeated processing to exhaust server resources
- MegaTags: Otherwise valid XML messages with excessively long element names may lead to buffer overruns
- Coercive parsing: XML messages specially constructed to be difficult to parse to consume the resources of the machine

- Public key DoS: Utilizing the asymmetric nature of public key operations to force resource exhaustion on the recipient by transmitting a message with a large number of long-key-length, computationally expensive digital signatures

Multiple-message XDoS attacks

Multiple-message XDoS attacks are:

- XML flood: Sending thousands of otherwise benign messages per second to tie up a Web service. This attack can be combined with replay attack to bypass authentication and single-message XDoS to increase its impact.
- Resource hijack: Sending messages that lock or reserve resources on the target server as part of a never-completed transaction.

Unauthorized access attacks

Unauthorized access attacks are:

- Dictionary attack: Guessing the password of a valid user using a brute force search through dictionary words
- Falsified message: Faking that a message is from a valid user, such as by using man in the middle to gain a valid message and modifying it to send a different message
- Replay attack: Resending a previously valid message for malicious effect, possibly where only parts of the message (such as the security token) are replayed

Data integrity/confidentiality attacks

Data integrity/confidentiality attacks are attacks that strike at the data integrity of Web service response, requests, or underlying database, such as:

- Message tampering: Modifying parts of a request or response in flight. Most dangerous when undetected. Less commonly known as *message alteration*.
- Data tampering: Exploiting weakness in the access control mechanism that permits the attacker to make unauthorized calls to the Web service to alter data.
- Message snooping: A direct attack on data privacy by examining all or part of the content of a message. This can happen to messages being transmitted in the clear, transmitted encrypted but stored in the clear, or decryption of messages due to a stolen key or cryptoanalysis.
- XPath/XSLT injection: Injection of expressions into the application logic. Newer modifications include blind XPath injection, which reduces the knowledge required to mount the attack.
- SQL injection: Inserting SQL in XML to obtain data in addition to what the service was designed to return.
- WSDL enumeration: Examining the services listed in WSDL to guess and gain access to unlisted services.
- Routing detour: Using SOAP routing header to access to internal Web services.

Systems compromise attacks

Systems compromise attacks corrupt the Web service itself or the servers that host it.

- Malicious include: Causing a Web service to include invalid external data in output or return privileged files from the server file system. For example, using embedded file URLs to return UNIX password files or other privileged data to the attacker.
- Memory space breach: Accomplished via stack overflow, buffer overrun, or heap error, allows execution of arbitrary code supplied by the attacker with permissions of the host process.
- XML encapsulation: Embedding system command in the XML payload, such as through the CDATA tag.
- XML virus (X-Virus): Using SOAP with Attachments or other attachment mechanisms to transmit malicious executables such as viruses or worms.

Security services: Standards supported by DataPower

Service	Relevant standards
Identity services	SAML, IdAS
Authentication service	WS-Trust, SAML, PKI, Kerberos
Authorization service	WS-Authorization, WS-Policy, XACML
Audit service	WS-BaseNotification, CBE extension
Message protection	WS-Security, WS-SecurityPolicy, WS-SecureConversation, XKMS, PKI, SSL/TLS

WSS specifications supported by DP.

Standard title	Description
WS-Security	http://www.oasis-open.org/specs/index.php#wssv1.1
WS-Trust	
WS-SecurityPolicy	http://www.oasis-open.org/specs/index.php#wssecpolv1.2
WS-SecureConversation	http://www.oasis-open.org/specs/index.php#wssecconv1.3
WS-ReliableMessaging	http://www.oasis-open.org/specs/index.php#wsrx-rm1.1
XML-Signature syntax and processing	

When to use DataPower for security

Ease of use is a pre-dominant consideration. DataPower offers a simple drop-in installation and panel-based configuration with little or no development required:

- You are transforming between XML and XML or XML and any other format.
- Your interaction patterns are relatively simple.
- Your mediation requirements are met by the existing DP mediations, and minimal extensibility is needed.
- You are using XML-based or WS-Security extensively.
- You require use of advanced Web services standards.

- You need to minimize message latency when adding an ESB layer.
- You are doing extensive XML processing combined with high-performance requirements.
- Your ESB must be in production very quickly.
- You want integration with other IBM WebSphere and Tivoli products (that is, ESB beyond limits).

Note that:

- All XML interaction with third parties should go through DataPower XS40 for XML threat protection for the traffic.
- If your DataPower appliance has to do more than act as a security gateway (that is, protocol transformation, XML traffic acceleration, or account for ESB) use the Datapower XI50.

DataPower appliances address XML threats by delivering a robust XML firewall for the enterprise. DataPower appliances introduce sophisticated checks on the incoming XML, as illustrated in Figure 3-1 on page 33. This includes the following:

- XML/SOAP firewall, filtering based on message content, headers, or other network variables
- Incoming/outgoing data validation
- Data schema validation (XML and binary)
- XML threat protection
- Single-message XML denial of service (XDoS) protection
- Multiple-message XML denial of service protection
- Message tampering protection
- Protocol threat protection
- XML virus protection
- Dictionary attack protection

References

For more information refer to the following resources:

- WebSphere Application Server

  http://www-306.ibm.com/software/websphere/
- Java 2 Enterprise Edition (J2EE)

  http://java.sun.com/j2ee/
- Enabling cryptographic hardware for the Secure Sockets Layer

  http://www-306.ibm.com/software/webservers/httpservers/doc/v2047/manual/ibm/en_US/9aecdssl.htm
- WebSphere Web Services Gateway (WSGW)

  http://www-128.ibm.com/developerworks/websphere/techjournal/0408_alcott/0408_alcott.html
- Extensible Markup Language (XML)

  http://www.w3.org/XML/

- XML Path Language (XPath)

  http://www.w3.org/TR/xpath
- The Extensible Stylesheet Language Family (XSL)

  http://www.w3.org/Style/XSL
- Specification: Web Services Security (WS-Security)

  http://www-106.ibm.com/developerworks/webservices/library/ws-secure/
- SOAP specifications

  http://www.w3.org/TR/soap/
- Simple API for XML

  http://www.saxproject.org/
- SOA

  http://www.ibm.com/soa/
- WebSphere Enterprise Service Bus

  http://www-306.ibm.com/software/integration/wsesb//
- WebSphere Message Broker

  http://www-306.ibm.com/software/integration/wbimessagebroker/

Glossary

A

Attribute. Characteristic of a subject, resource, action, or environment that may be referenced in a predicate or target (see also *named attribute*).

Authorization decision. The result of evaluating applicable policy, returned by the PDP to the PEP. A function that evaluates to permit, deny, indeterminate, or NotApplicable, and (optionally) a set of obligations.

Authorization decision. The result of evaluating applicable policy, returned by the PDP to the PEP. A function that typically evaluates to permit or deny.

B

Bag. An unordered collection of values, in which there may be duplicate values.

C

Condition. An expression of predicates. A function that evaluates to true, false, or indeterminate.

Conjunctive sequence. A sequence of predicates combined using the logical AND operation.

Context handler. The system entity that converts decision requests in the native request format to the XACML canonical form and converts authorization decisions in the XACML canonical form to the native response format

Context. The canonical representation of a decision request and an authorization decision.

D

Decision request. The request by a PEP to a PDP to render an authorization decision.

Decision. The result of evaluating a rule, policy, or policy set.

E

Entitlement. A data structure that contains externalized security policy information. Entitlements contain policy data or capabilities that are formatted in a way that is understandable to a specific application. For example, an automotive price lookup service is required to return different pricing information based on the entitlements associated with user identity. Assume that there are three policies defined for accessing this service: {ReadMSRP, ReadDealerPrice, and ReadDistributorPrice}. Each policy evaluates to a Boolean value. An example of entitlement in this case could be expressed as the following set of name-value pairs: {ReadMSRP=true, ReadDealerPrice=true, ReadDistributorPrice=false}.

Environment. The set of attributes that are relevant to an authorization decision and are independent of a particular subject, resource, or action.

N

Named attribute. A specific instance of an attribute, determined by the attribute name and type, the identity of the attribute holder (which may be of type subject, resource, action, or environment) and (optionally) the identity of the issuing authority.

O

Obligation. An operation specified in a policy or policy set that should be performed by the PEP in conjunction with the enforcement of an authorization decision.

P

Policy administration point (PAP). The system entity that creates a policy or policy set.

Policy decision point (PDP). The system entity that evaluates applicable policy and renders an authorization decision. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Decision Function" (ADF) in [ISO10181-3].

Policy enforcement point (PEP). The system entity that performs access control, by making decision requests and enforcing authorization decisions. This term is defined in a joint effort by the IETF Policy Framework Working Group and the Distributed Management Task Force (DMTF)/Common Information Model (CIM) in [RFC3198]. This term corresponds to "Access Enforcement Function" (AEF) in [ISO10181-3].

Policy information point (PIP). The system entity that acts as a source of attribute values.

Policy set. A set of policies, other policy sets, a policy-combining algorithm, and (optionally) a set of obligations. May be a component of another policy set.

Policy. A set of rules, an identifier for the rule-combining algorithm, and (optionally) a set of obligations. May be a component of a policy set.

Policy-combining algorithm. The procedure for combining the decision and obligations from multiple policies.

Predicate. A statement about attributes whose truth can be evaluated.

Resource manager. The system entity located between request initiator and target resource responsible for implementing the requested operation when authorization is granted. A component of the resource manager is a policy enforcer that directs the request to the authorization service for processing.

Resource. Data, service, or system component.

Rule. A target, an effect, and a condition. A component of a policy.

Rule-combining algorithm. The procedure for combining decisions from multiple rules.

Security context. Protected data structure attached to a message that contains security credentials associated with current request.

Subject. An actor whose attributes may be referenced by a predicate.

Target. The set of decision requests, identified by definitions for resource, subject, and action, that a rule, policy, or policy set is intended to evaluate.

Related publications

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

IBM Redbooks publications

For information about ordering these publications, see "How to get Redbooks publications" on page 221. Note that some of the documents referenced here may be available in softcopy only. The first four documents are Redpaper publications about the DataPower appliance.

- *IBM WebSphere DataPower SOA Appliances Part I: Overview and Getting Started*, REDP-4327

  http://www.redbooks.ibm.com/abstracts/redp4327.html?Open
- *IBM WebSphere DataPower SOA Appliances Part II: Authentication and Authorization*, REDP-4364

  http://www.redbooks.ibm.com/abstracts/redp4364.html?Open
- *IBM WebSphere DataPower SOA Appliances Part III: XML Security Guide*, REDP-4365

  http://www.redbooks.ibm.com/abstracts/redp4365.html?Open
- *IBM WebSphere DataPower SOA Appliances Part IV: Management and Governance*, REDP-4366

  http://www.redbooks.ibm.com/abstracts/redp4366.html?Open
- *Patterns: Service-Oriented Architecture and Web Services*, SC24-6303

  http://www.redbooks.ibm.com/abstracts/sg246303.html?Open
- *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346

  http://www.redbooks.ibm.com/abstracts/sg246346.html?Open
- *WebSphere Service Registry and Repository Handbook*, SG24-7386
- *Understanding SOA Security Design and Implementation*, SG24-7310
- *WebSphere Service Registry and Repository Handbook*, SG24-7386
- *DB2 9: pureXML Overview and Fast Start*, SG24-7298
- *Patterns: Implementing an SOA using an Enterprise Service Bus*, SG24-6346
- *WebSphere MQ V6 Fundamentals*, SG24-7128
- *SOA Transition Scenarios for the IBM z/OS Platform*, SG24-7331

Other publications

These publications are also relevant as further information sources:

- DataPower WebGUI Guide, also available online with free registration

  http://www.ibm.com/support/docview.wss?rs=2362&uid=swg24014405

Online resources

These Web sites are also relevant as further information sources:

- Product information about IBM WebSphere DataPower SOA Appliances

  http://www-306.ibm.com/software/integration/datapower/index.html
- "Integrating Web applications with the DataPower Web application firewall service," IBM Developer Works, Ozair Sheikh, 12 Dec 2007

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_sheikh/0712_sheikh.html
- IBM Web services

  http://www.ibm.com/webservices
- IBM on demand operating environment

  http://www-3.ibm.com/software/info/openenvironment/
- IBM developerWorks: SOA and Web services zone

  http://www.ibm.com/developerworks/webservices
- "First look at the WS-I Basic Profile1.0," 2002

  http://www.ibm.com/developerworks/webservices/library/ws-basicprof.htm
- WC3, Web Services Addressing (WS-Addressing), 2004

  http://www.w3.org/Submission/ws-addressing/
- Web Services Security Policy Language (WS-SecurityPolicy)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf
- developerWorks, "developerWorks Forum," 2007

  http://www.ibm.com/developerworks/forums/thread.jspa?messageID=14056243
- developerWorks, "developerWorks Forum," 2007

  http://www.ibm.com/developerworks/library/specification/ws-rm/
- IBM, BEA, Microsoft, TIBCO, "Web Services Reliable Messaging Protocol Specification," 2005

  http://specs.xmlsoap.org/ws/2005/02/rm/ws-reliablemessaging.pdf
- Web Services Reliable Messaging

  http://www.ibm.com/developerworks/webservices/library/specification/ws-rm/
- Web Services Policy Framework

  http://www.ibm.com/developerworks/webservices/library/specification/ws-polfram/
- Web Services Policy Framework (WS-Policy)

  http://specs.xmlsoap.org/ws/2004/09/policy/ws-policy.pdf
- Web Services Security Policy Language (WS-SecurityPolicy)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-secpol/ws-secpol.pdf
- Understand WS-Policy processing

  http://www.ibm.com/developerworks/webservices/library/ws-policy.html
- Web Services Reliable Messaging

  http://www.ibm.com/developerworks/webservices/library/specification/ws-rm

- Web Services Reliable Messaging Protocol (WS-ReliableMessaging)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf
- Web Services Policy Attachment

  http://www.ibm.com/developerworks/webservices/library/specification/ws-polatt/
- Web Services Policy Attachment (WS-PolicyAttachment)

  http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-polatt/ws-polat-2006-03-01.pdf
- “Using DataPower SOA Appliances to query WebSphere Service Registry and Repository”

  http://www.ibm.com/developerworks/websphere/techjournal/0805_peterson/0805_peterson.html
- The XACML specification:

  http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml
- OASIS eXtensible Access Control Markup Language (XACML) Version 1.1

  http://www.oasis-open.org/committees/download.php/2406/oasis-xacml-1.0.pdf
- Creating XACML Based Authorization using DataPower, Case Study by Karen Punwani
- XML schema and content validation using DataPower and DB2 pureXML

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0805malaika3/index.html?ca=drs-#resources
- Schematron Web site

  http://www.schematron.com/overview.html
- Generate Web services for DB2 9 pureXML

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0706bommireddipalli/
- Get started with Industry Formats and Services with pureXML

  http://www.ibm.com/developerworks/db2/library/techarticle/dm-0705malaika/
- Industry Formats and Services with pureXML

  http://www.alphaworks.ibm.com/tech/purexml
- “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus”

  http://www.ibm.com/developerworks/library/ws-soa-eda-esb/
- “Enterprise Service Bus implementation patterns”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html
- “Security Patterns within a Service-Oriented Architecture”

  http://www.ebizq.net/topics/soa/features/6554.html?&pp=1
- “Make SOA real with IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower SOA Appliance”

  http://www.ibm.com/developerworks/webservices/library/ws-soa-real1/
- “BPEL or ESB: Which should you use?”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0803_fasbinder2/0803_fasbinder2.html

- Exploring the Enterprise Service Bus, Part 2: Why the ESB is a fundamental part of SOA

  http://www.ibm.com/developerworks/library/ar-esbpat2/

- “Make SOA real with IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower SOA Appliances” series

  http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search_by=Make+SOA+real+with+IBM+WebSphere+Enterprise+Service+Bus+and+IBM+WebSphere+DataPower+SOA+Appliances

- eWeb Services ArchitectureW3C Working Group Note 11 February 2004

  http://www.w3.org/TR/ws-arch/#service_oriented_architecture

- “Getting started with WebSphere Transformation Extender Design Studio on DataPower,” developerWorks

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_vila/0712_vila.html

- Information about WebSphere Transformation Extender

  http://www-306.ibm.com/software/integration/wdatastagetx/library/index.html

- “WebSphere DataPower WebSphere MQ Interoperability Release 3.6.0.0,” developerWorks

  http://www-1.ibm.com/support/docview.wss?uid=swg21255199

- “Integrating WebSphere DataPower SOA Appliances with WebSphere MQ,” developerWorks

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html?S_TACT=105AGX78&S_CMP=ART

- WebSphere MQ Product home

  http://www.ibm.com/software/integration/wmq/

- “Service-enable CICS and IMS traditional applications using the IBM WebSphere DataPower SOA Appliance”

  http://www.ibm.com/developerworks/library/ar-datapow/

- “Integrating WebSphere DataPower SOA Appliances with WebSphere MQ”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0703_crocker/0703_crocker.html?S_TACT=105AGX78&S_CMP=ART

- “Integrating DataPower with WebSphere Message Broker using the Broker Explorer”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0707_storey/0707_storey.html

- “Enterprise Service Bus implementation patterns”

  http://www.ibm.com/developerworks/websphere/library/techarticles/0712_grund/0712_grund.html

- Configuration for high availability, configuration promotion, and control

  http://www.ibm.com/developerworks/websphere/library/techarticles/0801_rasmussen/0801_rasmussen.html

How to get Redbooks publications

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

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

IBM Redbooks

DataPower Architectural Design Patterns: Integrating and Securing Services Across Domains

DataPower Architectural Design Patterns Integrating and Securing Services Across Domains

Introduction to DataPower Services

Integration Services

Security Services

IBM® WebSphere® DataPower® SOA Appliances are purpose-built network devices that offer a wide variety of functionality such as the securing and management of SOA Applications, enterprise service bus integration, and high speed XSL execution. A hardened appliance, DataPower provides robust security features including tamper protection of the device itself.

This IBM Redbooks publication was written for application architects and other consultants who want to include DataPower appliances in their solutions for reasons of speed, security, or ESB integration. The topics include DataPower services, Web services, security, and integration strategies.

SG24-7620-00 ISBN 0738431710