

# Sun ONE Architecture Guide

---

*Delivering Services on Demand*



Sun Microsystems, Inc.  
901 San Antonio Road  
Palo Alto, CA 94303  
1 (800) 786.7638  
1.512.434.1511

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Java, Java HotSpot, J2SE, Forte, iPlanet, NetBeans, and Solaris are trademarks, registered trademarks, or service marks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

**RESTRICTED RIGHTS:** Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

---

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, Californie 94303 Etats-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, Java, Java HotSpot, J2SE, Forte, iPlanet, NetBeans, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Please  
Recycle



Adobe PostScript

# Contents

---

<b>Preface: About This Book</b> .....	<b>i</b>
What's in This Book .....	i
Who Should Read This Book .....	iv
What This Book Does Not Include .....	v
<b>Part 1. Introduction</b> .....	<b>1</b>
Chapter 1: Delivering Services on Demand .....	3
Evolution of Networked Computing .....	4
Scope of Services on Demand .....	6
The Web Application Model .....	7
The Web Services Model .....	8
Services and Registries .....	10
ebXML: Enabling Integrated e-Commerce .....	11
The Java Web Client Model .....	13
Beyond Formal Standards .....	13
Moving Forward .....	13
Chapter 2: The Sun™ Open Net Environment (Sun ONE) Architecture .....	15
Service Stack .....	15
Standards Associated with the Sun ONE Platform .....	17
Product Mappings to the Service Stack .....	18
Sun ONE Architecture Integration and Interoperability .....	19
Interoperability with Existing Applications .....	20
Interoperation with Microsoft .NET .....	20
Phases of Adoption .....	21
<b>Part 2. Service Containers</b> .....	<b>25</b>
Chapter 3: J2EE™ Components and Containers .....	27
Components .....	28
Java Servlet API .....	29
JavaServer Pages™ Technology .....	29
Enterprise JavaBeans™ .....	30
J2EE Platform Container Provided Services .....	33

Web Services . . . . .	34
XML Document Parsing API (JAXP) . . . . .	34
XML-to-Java Object Binding Facility (JAXB) . . . . .	35
SOAP RPC API (JAX-RPC) . . . . .	35
Business Registry and Repository (JAXR) . . . . .	35
SOAP Messaging API: JAXM . . . . .	35
Internet Mail API (JavaMail) . . . . .	36
Cross-platform and CORBA Interoperability . . . . .	36
Access to Database Servers . . . . .	37
Access to Name and Directory Servers . . . . .	38
Component and Container Interfaces . . . . .	38
The Application Server . . . . .	39
The iPlanet™ Application Server . . . . .	40
Deployment Options . . . . .	41
Scalability . . . . .	41
High Availability . . . . .	41
Management . . . . .	41
Tools Integration . . . . .	42
Component Life Cycle Optimizations . . . . .	42
Platform Integration . . . . .	42
Interchangeable Components . . . . .	42
Application Server Interfaces . . . . .	43
<b>Part 3. Service Integration . . . . .</b>	<b>45</b>
Chapter 4: The J2EE™ Connector Architecture and Web-Service-Based Integration . . . . .	47
Overview of the EIS Integration Facilities . . . . .	48
Overview of the Connector Architecture . . . . .	49
Advantages of the Connector Architecture . . . . .	49
Connector Architecture Contracts . . . . .	49
System-Level Contracts . . . . .	50
Application-Level Contract . . . . .	51
Packaging and Deployment . . . . .	51
Web-Service-Based Integration . . . . .	51
Support for Web Services in the Sun ONE Architecture . . . . .	52
Using Web Services from the Sun ONE Architecture . . . . .	52
Implementing Web Services with the Sun ONE Architecture . . . . .	53
J2EE Connector Architecture Interfaces . . . . .	53
Chapter 5: Asynchronous Reliable Messaging . . . . .	55
Messaging Basics . . . . .	55
Existing Messaging Systems . . . . .	56
Asynchronous Reliable Messaging Systems . . . . .	56
The Java™ Message Service Technology . . . . .	57

Objectives of the JMS Technology . . . . .	57
Java Message Service Technology Provider (“JMS Provider”) . . . . .	57
Java Message Service Technology Clients (“JMS Clients”) . . . . .	58
Java Message Service Technology Messages (“JMS Messages”) . . . . .	58
Java Message Service Technology Domains (“JMS Domains”) . . . . .	58
Portability. . . . .	58
Java Message Service Does Not Include . . . . .	58
Java Message Service Requirements. . . . .	59
Requirements Beyond Java Message Service. . . . .	59
Multiple Queue Delivery Styles . . . . .	60
Multiple Protocol Support . . . . .	60
Security. . . . .	60
Object Management. . . . .	60
Pluggable Persistence. . . . .	61
Distributed Transaction Support. . . . .	61
iPlanet™ Message Queue for Java . . . . .	61
Administered Objects . . . . .	62
Client Runtime . . . . .	63
Message Production . . . . .	63
Message Consumption . . . . .	63
ConnectionFactory Administered Objects . . . . .	64
Destination Administered Objects . . . . .	65
The Message Service. . . . .	65
Broker Components and Functions. . . . .	65
Connection Services . . . . .	67
Message Router . . . . .	68
Basic Delivery Mechanisms . . . . .	68
Reliable Delivery . . . . .	69
Persistence Manager . . . . .	70
Security Manager. . . . .	70
Authentication . . . . .	71
Authorization . . . . .	71
Encryption . . . . .	71
Logger . . . . .	71
Multi-Broker Configurations (Clusters) . . . . .	72
Multi-Broker Architecture. . . . .	72
Physical Destinations . . . . .	73
Queue Destinations . . . . .	74
Topic Destinations . . . . .	75
Auto-Created (vs. Admin-Created) Destinations . . . . .	75
Chapter 6: Business Process Integration . . . . .	77

The Integration Challenge . . . . .	78
Setting 1: Business Document Exchange . . . . .	78
Setting 2: Connecting Internal Applications . . . . .	78
Setting 3: Establishing New Partnerships and Businesses . . . . .	79
Existing Styles of Integration . . . . .	79
B2B Integration . . . . .	80
EAI Integration . . . . .	81
e-Commerce Integration . . . . .	82
ebXML . . . . .	82
ebXML Objectives and Architecture . . . . .	83
ebXML Messaging . . . . .	83
ebXML Collaboration Elements . . . . .	84
Collaboration Protocol Profiles . . . . .	84
Collaboration Protocol Agreements . . . . .	85
Document Exchange Processes . . . . .	85
Business Process Schema Specification . . . . .	86
Registry Repository . . . . .	86
ebXML Core Components Project . . . . .	86
ebXML Functional Overview . . . . .	87
Reliable Electronic Business Exchange . . . . .	88
Advantages over Fax Messaging . . . . .	88
Differentiation from Web Browser Messaging . . . . .	88
Importance of Quality of Service . . . . .	89
Authentication and Audit . . . . .	89
Profiles and Agreements in Practice . . . . .	89
Exchange Processes . . . . .	90
iPlanet™ Integration Server . . . . .	91
Controller/Coordination Layer . . . . .	92
Private Business Process Engine . . . . .	93
Message Routing Table . . . . .	94
Document Exchange Process Engine . . . . .	94
Data Transformation and Translation Layer . . . . .	94
Messaging Interface Layer . . . . .	95
Future Directions for Messaging . . . . .	96
Process Integration Interfaces . . . . .	96
<b>Part 4. Service Creation, Assembly, and Deployment . . . . .</b>	<b>99</b>
Chapter 7: Development Tools . . . . .	101
Sun™ Open Net Environment (Sun ONE) Platform Tool Suite Requirements . . . . .	102
NetBeans™ Software IDE . . . . .	102
NetBeans Software Core and APIs . . . . .	103

Other Key Modules for Tool Developers. . . . .	.106
API Support Module . . . . .	.106
Form Editor . . . . .	.107
The Metadata Repository . . . . .	.108
Other Tool-Related NetBeans Software Modules . . . . .	.109
NetBeans Software Interfaces. . . . .	.109
Forte™ IDE . . . . .	.110
Primary Components . . . . .	.110
Partner and Third-Party Tools . . . . .	.113
Services Development in the Forte IDE . . . . .	.113
Services-Centric Functionality. . . . .	.113
Integrated Architecture Capabilities. . . . .	.114
Extensible Architecture Design . . . . .	.115
Service Creation . . . . .	.115
Service Assembly . . . . .	.116
Service Deployment . . . . .	.117
Forte IDE Interfaces. . . . .	.117
<b>Part 5. Service Delivery . . . . .</b>	<b>.119</b>
Chapter 8: Presentation Frameworks. . . . .	.121
Overview of Presentation Frameworks . . . . .	.121
The Model-View-Controller . . . . .	.122
The MVC Design Model . . . . .	.122
MVC1. . . . .	.123
MVC2. . . . .	.124
MVC2 and the Presentation Framework . . . . .	.124
Development Issues . . . . .	.125
Template and Non-Template-based MVC Architectures. . . . .	.126
Template Engines . . . . .	.126
DOM Manipulation . . . . .	.127
Advantages of DOM and Template Approaches . . . . .	.128
Java Specification Request™ (JSR) 127 Architecture . . . . .	.128
JSR 127 Design Goals. . . . .	.129
Overview of the iPlanet™ Application Framework . . . . .	.130
The iPlanet Application Framework's Implementation of MVC2. . . . .	.131
The iPlanet Application Framework and JSR 127 . . . . .	.132
The iPlanet Application Framework's Use of Design Patterns. . . . .	.133
Types of iPlanet Application Framework Functionality . . . . .	.133
Technical Overview of the iPlanet Application Framework Core . . . . .	.135
iPlanet Application Framework Features . . . . .	.136
Symmetrical Display/Submit Handling. . . . .	.137

Formal Model Entity . . . . .	137
Application Events . . . . .	138
Hierarchical Views and Component Scoping . . . . .	139
Efficient Object Management . . . . .	139
Support for Parallel Content . . . . .	140
Ready-to-Use, High-Level Features . . . . .	140
Tool-Readiness . . . . .	141
Scalability . . . . .	141
Presentation Framework Interfaces . . . . .	142
Chapter 9: The Portal Server . . . . .	143
Using the Portal to Deliver Web Applications . . . . .	143
Aggregation and Presentation . . . . .	144
Specialized Providers for Aggregation and Presentation . . . . .	145
Personalization for Users and Applications . . . . .	146
Security for Users and Applications . . . . .	146
Management of Users and Applications . . . . .	147
Enhancing the Portal for Web Services . . . . .	147
Aggregation and Presentation of Web Services . . . . .	147
Delivery to End Users . . . . .	148
Delivery to Web Services and Applications . . . . .	148
Personalization for Web Services . . . . .	148
Supporting the Java Web Client Model . . . . .	149
System-Level Interfaces Supporting Personalization . . . . .	149
Location SPI . . . . .	149
Presence SPI . . . . .	149
Notification SPI . . . . .	149
Usage SPI . . . . .	150
Security for Web Services . . . . .	150
Management of Web Services . . . . .	150
Specialized Web Service Provider . . . . .	150
Portal Server Interfaces . . . . .	151
Chapter 10: The Java Web Client Model . . . . .	153
Design and Deployment Considerations . . . . .	154
Protocols and Payloads . . . . .	154
Environment . . . . .	155
MVC Design for Java Web Clients . . . . .	155
Supporting Architectural Elements . . . . .	156
XML Information Services and Device Interaction . . . . .	157
Client Device . . . . .	157
Java™ Virtual Machine (JVM™) and KVM Operating Environments . . . . .	158



Mobile Devices . . . . .	.158
Extended Services . . . . .	.158
Telephony Access Mechanisms . . . . .	.159
Server-Side Provisioning . . . . .	.160
Java Web Client Model Interfaces . . . . .	.160
<b>Part 6. Fundamental Services . . . . .</b>	<b>.163</b>
Chapter 11: Identity and Policy Services . . . . .	.165
Identity, Roles, and Security . . . . .	.166
iPlanet™ Directory Server Products . . . . .	.168
Identity: Authentication . . . . .	.168
Identity: Web Single Sign-On . . . . .	.170
Identity: Cross-Domain Single Sign-On . . . . .	.171
Identity Management: User Account Management and Provisioning . . . . .	.171
User Provisioning and Self-Registration . . . . .	.171
Profile API . . . . .	.172
User Organization and Integration . . . . .	.172
Delegated Management . . . . .	.172
Policy Management and Evaluation . . . . .	.172
Policy Framework . . . . .	.173
Plug-in SPIs . . . . .	.174
Management and Evaluation APIs . . . . .	.174
Privacy . . . . .	.175
Difference Between Policy and Privacy . . . . .	.175
Security . . . . .	.176
Public Key Infrastructure . . . . .	.176
Kerberos . . . . .	.177
UDDI . . . . .	.177
Logging and Audit . . . . .	.178
Identity and Policy Services Interfaces . . . . .	.178
Federated Identity Systems . . . . .	.180
Liberty Alliance Project . . . . .	.180
Management Services . . . . .	.181
The Sun ONE Platform Management Architecture . . . . .	.181
Sun ONE Platform Management Information and CIM Models . . . . .	.183
Use of the CIM Model for All Sun ONE Platform Components . . . . .	.183
Use of Standard Schema to Describe All Sun ONE Platform Components . . . . .	.184
Provision of CIM-Based APIs . . . . .	.184
Integration of Existing Management Schemes . . . . .	.185
Interoperation via Multiple, Pluggable Protocol Adapters . . . . .	.185
Use of Standard CIM Mappings to Existing Schema . . . . .	.186
SNMP . . . . .	.186

Java™ Management Extensions (JMX™) .....	186
J2EE Platform Management .....	187
Management Interfaces .....	187
Chapter 12: Platform Services .....	189
Hardware Platform and Resource Management .....	190
Scaling and RAS .....	190
Interfaces to Exploit Cluster Configurations .....	191
Service and Application Failover .....	191
High-speed Communication, Synchronization, and Checkpointing .....	192
Solaris™ Resource Manager .....	192
Networking .....	193
Networking in the Solaris Operating Environment .....	193
Storage, Filing, and Data Access .....	194
Solaris Operating Environment Storage, Filing, and Data Access .....	194
Name Service, Directory, and Registry Functions .....	194
Naming, Registry, and Directory Services in the Solaris Operating Environment ...	195
Security .....	195
Authentication .....	195
Secure Internet Transport .....	196
Strong Random Numbers .....	196
Interfaces .....	197
<b>Part 7. Core Web Services .....</b>	<b>201</b>
Chapter 13: Core Web Services .....	203
Portal-based Web Services .....	203
Location Web Service .....	204
Presence Web Service .....	204
Notification Web Service .....	204
Usage Web Service .....	205
Search Web Service .....	205
File Web Service .....	205
Communications Applications Web Services .....	205
Mail Web Service .....	206
Calendar Web Service .....	206
Contacts Web Service .....	206
Conferencing Web Services .....	207
<b>Glossary .....</b>	<b>209</b>
<b>Bibliography .....</b>	<b>217</b>
<b>Index .....</b>	<b>225</b>

# Preface: About This Book

---

Sun™ Open Net Environment (Sun ONE) is Sun Microsystems' standards-based software vision, architecture, platform, and expertise for building and deploying Services on Demand.

## What's in This Book

This book provides developers with an understanding of the components and functionality of the Sun ONE architecture. It also includes descriptions of the Sun technologies and tools that map to each portion of that architecture, along with a listing of the interfaces that are consumed or presented by each component.

This book contains the following information, presented in seven parts:

### Part 1, Introduction

**Chapter 1, *Delivering Services on Demand***, summarizes the history of networked computing. After discussing the ways in which the dynamic concepts of Web services and Services on Demand are transforming today's networked computing environment, it explains how the Sun ONE architecture is designed to implement those concepts.

**Chapter 2, *The Sun ONE Architecture***, describes the various layers of the Sun ONE architecture, along with their associated standards and integrated products. Taken together, these elements provide a basis for the scalable, reliable, open-standards-based Web services of today and for the Services on Demand of tomorrow.

## Part 2, Service Containers

**Chapter 3, Java™ 2 Platform, Enterprise Edition (J2EE™ platform) Components and Containers**, explains how the J2EE platform defines the Service Container for the Sun ONE architecture. In addition to the J2EE technology standard, the Service Container includes a de facto standard based on the iPlanet™ Presentation Framework, along with various Web-services-oriented APIs, tools, and technologies. This chapter concludes with a discussion of iPlanet Application Server, a fifth-generation architecture that provides high levels of performance, scalability, and reliability in a J2EE technology-conformant application server.

**Chapter 4, J2EE Connector Architecture and Web-Services-Based Integration**, explains how the J2EE Connector Architecture defines a standard way to extend the Service Container for the Sun ONE platform to integrate applications with an existing Enterprise Information System (EIS). It then explores the ways in which the Sun ONE architecture's native support for the Web-service distributed computing paradigm provides an efficient EIS integration mechanism in itself.

## Part 3, Service Integration

**Chapter 5, Asynchronous Reliable Messaging**, compares existing messaging technologies to asynchronous reliable messaging systems, which are essential to the conduct of e-commerce between loosely integrated business partners. It goes on to discuss the Java Message Service, a standard API for messaging that supports reliable point-to-point messaging as well as the publish-subscribe model. This chapter concludes with a description of the iPlanet Message Queue for Java software, a current example of an asynchronous reliable messaging system.

**Chapter 6, Business Process Integration**, explains the various types of business process integration, including Business-to Business, Enterprise Application Integration, and Electronic Commerce integration. It then discusses the Electronic Business eXtensible Markup Language (ebXML) collection of Web-services specifications in the context of process integration. In conclusion, the iPlanet Integration Server is described as an example of an ebXML application that enterprises can use for both collaboration and reliable messaging in the dynamic, international marketplace.

## Part 4, Service Creation, Assembly, and Deployment

**Chapter 7, *Development Tools***, describes the Sun ONE platform development tool suite for the creation, assembly, and deployment of Web services and Services on Demand. The Forte™ integrated development environment (IDE) and Netbeans™ software are discussed at length. In addition, this chapter considers other Netbeans software modules and APIs that provide value-add functionality on top of the Forte IDE, including the API Support module, the Form Editor, and the Metadata Repository (MDR) component of NetBeans software.

## Part 5, Service Delivery

**Chapter 8, *Presentation Frameworks***, discusses the frameworks that are responsible for gathering information from end users and the business layer of an application. A discussion of presentation frameworks in terms of the Model-View-Controller design model is followed by a consideration of Java Specification Request 127, which will provide the functionality on which presentation frameworks can standardize. Finally, this chapter describes the iPlanet Application Framework implementation of a presentation framework.

**Chapter 9, *The Portal Server***, outlines the mechanism that allows Web applications to be displayed within a single page or set of pages that the user can view in a browser. It then discusses how the capabilities that a portal server provides to Web applications can be extended to the Web services world.

**Chapter 10, *The Java Web Client Model***, considers the Web client model enabled by the Java technology (“Java Web client”) for delivery of Web services. Under the Web client model, applications written in the Java programming language are provisioned over the Web to a Web-enabled device—for example, a desktop computer, a PDA, or a Web-enabled cell phone. After discussing the Java Web Client architecture in terms of the MVC design model, this chapter focuses on the special challenges posed by the use of mobile devices, telephony access, and server-side provisioning.

## Part 6, Fundamental Services

**Chapter 11, *Identity and Policy Services***, defines the Sun ONE architecture’s security mechanisms, including single sign-on, account synchronization and provisioning, policy, privacy, personalization, and the identity solutions provided by the Liberty Alliance Project. These security mechanisms are included in the iPlanet Directory Server Access Management Edition and Integration Edition. This chapter also examines Management Services, which provide both the architectural framework and the collection of programmatic interfaces needed to manage Web services and system resources throughout the Sun ONE platform.

**Chapter 12, Platform Services**, describes the interface at the lowest level of the Sun ONE architecture. It explains how Platform Services provide the functions needed to allocate and manage the resources of the underlying network and hardware platform required to host the higher-level services in the Sun ONE platform. In a large part, it focuses on the Solaris™ Operating Environment, which provides hardware platform and resource management; naming, registry, and directory services; networking; storage, filing, and data access; and security.

## Part 7, Core Web Services

**Chapter 13, Core Web Services**, defines the manner in which Core Web Services will make the functionality included in the infrastructure of the Sun ONE architecture and in existing Web applications by Sun available for use by other Web services and applications. The discussion focuses on two main categories of Core Web Services—Location Web Service and Communications Applications Web Services.

This book concludes with a Glossary, Bibliography, and Index.

## Who Should Read This Book

Although of interest to anyone involved with the creation and implementation of e-business solutions, Web services, and Services on Demand, the Sun ONE architecture is specifically intended for three primary audiences:

- **Developers** who want the shortest path to Web services and Services on Demand by leveraging the tools and technologies that they already know.
- **CTOs, systems architects, systems integrators, and programmers** involved in the development of e-business initiatives, all of whom will benefit from a greater understanding of the Sun ONE architecture and its role in the creation and implementation of Services on Demand.
- **Technical architects** who need to understand how the Sun ONE architecture can help them achieve a technical architecture that supports their organization's needs.
- **Independent software vendors (ISVs)** who want to leverage the open nature of the Sun ONE architecture by using it as a foundation to build new applications and services or to incorporate major components such as directories or content management systems.

## What This Book Does Not Include

As a technical overview of the Sun ONE architecture, this book is necessarily limited in scope. It does not deal with definitions of “architecture” that can be applied in various domains, such as the architecture of a specific application, a computer room, or a network. It also does not include:

- Product features, detailed product road maps, marketing claims, or competitive analyses.
- Best practices guidance for IT organizations in regards to sizing, installing, configuring, or operating Sun ONE platform-based systems.
- Business justifications such as TCO or ROI that are slanted to executive audiences.
- Design guidance for developers of vertical industry services. Although some components of the Sun ONE architecture can be applied to vertical industries such as telecommunications, this book does not cover industry-specific standards, interoperability requirements, or business rules.

For information regarding the topics listed above, refer to <http://www.sun.com> and <http://www.sun.com/sunone>.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.





# Part 1. Introduction

---



## Delivering Services on Demand

---

The ability to rapidly develop and reliably deploy scalable, highly available business systems based on Internet technology is a major IT industry focus. To be successful, these systems must integrate with legacy applications and data. Furthermore, they must be available across an organization's many communities—including employees, partners, and customers—and have the ability to run on a broad range of devices.

Today, the Web has emerged as a versatile platform for delivering high-value solutions. Services can be accessed from virtually any device, including cellular phones, PDAs, and desktops. Technologies and protocols have been developed to integrate existing business processes and resources and make them available over the Web.

Businesses use a variety of distributed system alternatives to run their operations, including:

- **Local applications** – Applications that run on dedicated workstations or PCs, such as office applications.
- **Client/server applications** – Applications that are split, more or less, between presentation logic on a client device and business logic on a server. Typically these applications perform business-critical functions such as accounting, human resources, and manufacturing. Because of this, they usually require large database back-ends and require tight coordination between the programs running on the clients and servers.
- **Web applications** – Applications that run over the Web, such as shopping, financial account management, and management information services. These applications are based on client browsers and standard Web protocols such as HyperText Markup Language (HTML) and HyperText Transfer Protocol (HTTP).
- **Web services** – Services that run over the Web that can combine with other services to create a more useful or powerful solution. Web services provide a modular, well-defined, and encapsulated function, based on eXtensible Markup Language (XML), that is used for loosely coupled integration between applications or systems.
- **Web clients** – Applications written in the Java™ programming language that are delivered over the Web to Java technology-enabled devices such as personal computers, cellular phones, and PDAs.

Sun uses the term “Services on Demand” to describe all of the above applications and services.

Services on Demand constitute the manner in which enterprises use their information assets to transact and report business operations and to communicate with others—anywhere, anytime, on any device. The Services on Demand concept is the foundation for modular, flexible, integratable, and automated access to digital assets—including computing sources—from virtually anywhere.

The Services on Demand vision is of a comprehensive framework, encompassing traditional Internet-based services, such as security, authentication, and directory, along with more advanced capabilities, including virtualized storage and composite services (those created by combining separate services). Services on Demand represent evolution, not revolution. Therefore, it will not supersede other network and development approaches. In order to make the Services on Demand model attractive, businesses must be able to leverage existing application assets and expose them as services.

Rather than connecting with existing resources, Services on Demand can leverage and extend them. Vendors are now offering tools and technologies that will reduce the cost, risk, and complexity of moving to this new model. The Sun™ Open Net Environment (Sun ONE) architecture includes one such set of tools, which can be easily integrated with existing products from other vendors.

Section 1.2 explores the concept of Services in Demand in greater depth. To provide a background for that discussion, Section 1.1 outlines the past, present, and immediate future of network computing.

## 1.1 Evolution of Networked Computing

Networked computing has evolved in relatively self-contained and distinct stages. As illustrated in Figure 1–1, each stage has introduced an entirely new set of protocols and levels of integration. Each evolutionary shift is equivalent in complexity to the creation and launch of a new operating system. At any point in time, an enterprise is likely to have several instances of each generation of network computing—there are no complete and precise transitions from earlier generations to successive ones. Sun ONE is designed to move enterprises to the current generation of networked applications.







Catch Phrase	The Network is the computer	Objects	Legacy to the Web	The Computer is the Network	Network of embedded things	Network of things
Scale	100s	1000s	1000000s	10000000s	100000000s	1000000000s
When/Peak	1984/1987	1990/1993	1996/1999	2001/2003	1998/2004	2004/2007
Leaf Protocol(s)	X	X	+HTTP (+JVM)	+XML, Portal	+RMI	Unknown
Directory(s)	NIS, NIS+	+CDS	+LDAP(*)	+UDDI	+Jini	+?
Session	RPC, XDR	+CORBA	+CORBA, RMI	+SOAP, XML	+RMI/Jini	+?
Schematic						

Figure 1–1: Evolution of Networked Computing

Although the introduction of client/server computing consisted a significant advancement over the host-based, centralized computing model, the level of integration remained essentially zero. Client front-ends provided a friendlier user interface, but actual access to data, data flow, and the effectiveness of applications showed little improvement.

The current common architecture for networked applications, represented by “Legacy to the Web,” illustrates that integration has taken a major step forward—various applications can now be accessed simultaneously in the browser, and the client handles a major part of the aggregation and assembly tasks. Because of these advances, the user experience has improved considerably.

The fourth column of Figure 1–1 illustrates the rise of Web services that offer a new style of loosely coupled integration between applications or systems. Web services can be processed by desktop or server applications or by Java™ 2 Platform, Micro Edition (J2ME™ platform) applications in a wide variety of handheld devices. The degree of networked interaction increases as the primary service assembly point—the point of integration—moves from the client devices into the server space. Standards, such as Universal Description, Discovery, and Integration (UDDI), Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL) enable the rapid identification, assembly, and interoperation of Web services on the network. For further discussion of Web services, see Section 1.2.2.

The last two columns of Figure 1–1 represent the increasing numbers of network-addressable devices, leading to architectures focused on dynamic, self-organizing systems, where peer-to-peer transactions are as prevalent as server-centric communications in previous generations. New standards and technologies such as Sun’s Jini™ network technology environment will drive the adoption of these future architectures.

The Sun ONE architecture is designed to accommodate the next generations of networked computing that support Services on Demand, including Web applications and dynamic, federated Web services. It provides a migration path from the Legacy to the Web phase of networked applications to the world of Web services that are dynamically assembled.

## 1.2 Scope of Services on Demand

Services on Demand is an umbrella category that encompasses:

- Past Web initiatives
- Today's Web applications, Web services, and Web clients enabled by the Java technology ("Java Web clients")
- Tomorrow's new services, including contextually enhanced Web services that are aware of user context and identity to create a superior interactive, online experience
- Potentially, new technologies such as peer-to-peer (JXTA) and dynamic configuration infrastructure (Jini network technology)

In the Sun ONE architecture, Services on Demand can be delivered over the Web in the three ways illustrated in Figure 1-2.

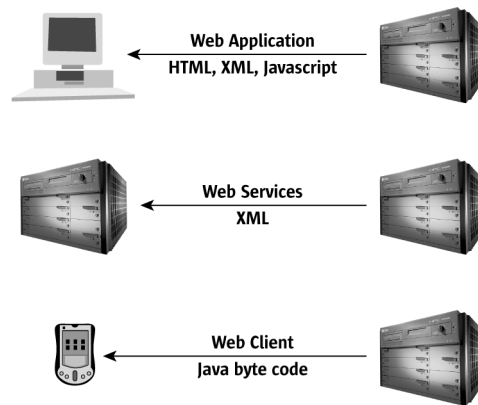


Figure 1-2: Three Ways of Delivering Services on Demand

As shown in Figure 1-2, Web applications consist of markup language delivered over the Web to a client application, frequently a Web browser. Web services, on the other hand, consist of XML documents delivered over the Web by an application running on one computer to an application running on another computer. Finally, Java Web Clients are Java technology applications delivered over the Web to any device capable of running them.

Services on Demand have the flexibility to encompass new protocols and methods of operation in order to deliver customized, personalized services whenever and wherever they are needed.

### 1.2.1 The Web Application Model

Most of the Services on Demand delivered prior to the year 2002 fall under the heading of Web applications. Web applications deliver dynamic content via Web browsers for human interaction. The Sun ONE architecture includes a variety of standards, technologies, and products for developing, deploying, and dynamically updating Web applications. For example:

- JavaServer Pages™ (JSP) and servlets are frequently used to generate dynamic content, as described in Chapter 3, “J2EE™ Components and Containers.”
- The J2EE™ Connector Architecture allows the rapid integration of existing Enterprise Information Systems (EIS) to allow parts of them to be accessed via the Web, as described in Chapter 4, “The J2EE™ Connector Architecture and Web-Service-Based Integration.”
- The iPlanet Message Queue for Java™ provides asynchronous reliable message delivery among Web applications, as described in Chapter 5, “Asynchronous Reliable Messaging.”
- The iPlanet™ Integration Server facilitates the integration of Web services and applications, as described in Chapter 6, “Business Process Integration.”
- The NetBeans architecture provides a set of tools that enables the rapid development and deployment of Web services and applications, as described in Chapter 7, “Development Tools.”
- The Forte™ tools, which are based on Netbeans™ software framework, also support the creation, assembly, and deployment of Web applications, as described in Chapter 7, “Development Tools.”
- Java 2 Platform, Enterprise Edition (J2EE™) and its presentation layer technologies (servlets and JSPs) provide a portable and standards-based foundation for building Web applications, as described in Chapter 8, “Presentation Frameworks.”
- Web applications are typically delivered through a portal server, as described in Chapter 9, “The Portal Server.”
- Security and authentication services, along with systems and application management, are provided to Web applications through the Sun ONE architecture’s identity and policy services, as described in Chapter 11, “Identity and Policy Services.”

## 1.2.2 The Web Services Model

Within the realm of Services on Demand, the term “Web services” has acquired a specific meaning that has been endorsed by industry vendors and analysts. Web services are modular, encapsulated functions that can discover and engage other Web services to complete complex tasks over the Internet. Unlike hard-wired applications—for example, traditional client/server applications based on Remote Procedure Calls (RPCs), Common Object Request Broker Architecture (CORBA), or Distributed COM (DCOM)—Web services are loosely coupled. They can dynamically locate and interact with other components on the Internet to provide services, and can themselves be dynamically located and used by other Web services.

In other words, Web services transform services into clearly defined components and allow those services to be easily interconnected. A Web service is usually invoked by a program, not directly by a human user. It is used to integrate applications—either within the enterprise or over the Internet—between the enterprise and its customers and business partners.

A Web service almost always passes an XML message, either synchronously by using a remote-procedure-call style, or asynchronously in a reliable messaging-passing style. Some vendors highlight their dependency on XML by using the alternative term “XML Web services.” The reliable message-passing style can be over a message bus, such as Java™ Message Service, or through the emerging Electronic Business XML (ebXML) Messaging Service.

Popular standards for transport include SOAP and ebXML. In addition, Web services may be advertised and described in a service registry such as UDDI, which can store references to service interfaces specified in WSDL, a framework for describing a Web service’s XML-based interfaces.

The Web service can be composed of one or both of two styles of programs:

- Programs composed of components written by the developer specifically to create the Web service. These components can be written using technologies such as Enterprise JavaBeans™ (EJB) or as part of J2EE, with a wrapper on the front end to convert the component into XML format.
- Programs that work with legacy systems, such as existing Customer-Relations Management (CRM) or Enterprise Resource Planning (ERP) applications, to make individual pieces of the application’s functionality directly available through the Web service.

A Web service can be accessed programmatically by applications or other Web services. As a result, a complex Web service can be developed from an assembly of several other Web services. For example, a Web service designed to calculate an individual’s net worth, is pre-programmed to access a number of Web sites—such as the person’s bank, brokerage house and insurance company—in order to retrieve the necessary information. This example is illustrated in Figure 1-3.



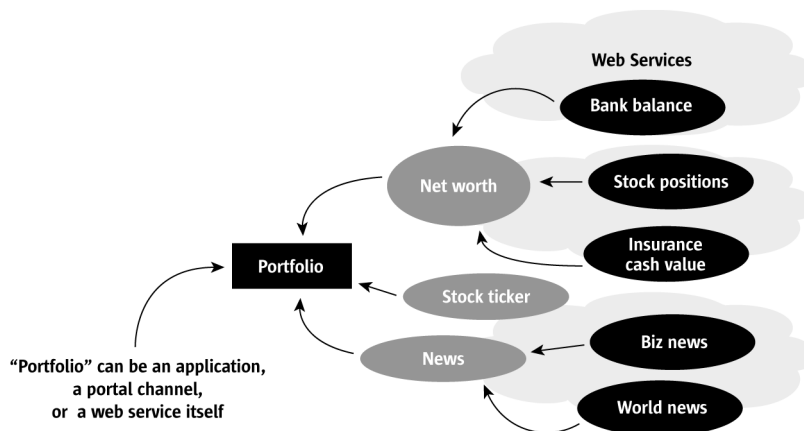


Figure 1-3: Web Services Example

The process shown in Figure 1-3 points up a fundamental difference between Web services and human-oriented services. For a human, it is fairly easy to access the Web sites and calculate net worth by working through the relevant HTML pages. It is far more difficult to write a program that dynamically accesses the requisite sites and parses through the HTML pages. However, Web services allow the developer to write a relatively simple program that parses XML messages in a particular format. The developer is isolated from the ambiguities of HTML and the additional complications associated with Web sites that frequently change their content and organization. In this example, the developer can construct a “portfolio” that could be an application, a specialized channel in a portal server, or the Web service itself.

Another reason Web services are considered an important development in the industry is simplicity of integration. Although it is relatively easy to use Enterprise Application Integration (EAI) or Java Connectors technology to connect any two applications together within an enterprise, challenges arise going beyond this model:

- When integration extends beyond the boundary of a single enterprise, complexities increase as security, firewall access, and version control issues are introduced.
- When integration changes from 1-to-1 to n-to-n, to multiple business partners participating in more complex transactions together, it becomes difficult to scale the custom solutions that may be necessary for each connection.

- In this second case, Web services can be considered as a trigger for an integration “network effect.” (Also known as Metcalfe’s Law—the usefulness, or utility, of a network is proportional to the square of the number of users.) Just as the value of a network of telephones increases as additional subscribers connect to the network, the existence of new business partners employing simple integration interfaces could lead to an era of dynamic supply chains and multi-party business arrangements that would be uninteresting if only a few enterprises participated.

### 1.2.2.1 Services and Registries

The interconnections between the UDDI registry, tools, protocols, and Web services are shown in Figure 1–4. The registry can be either a public or private enterprise registry.

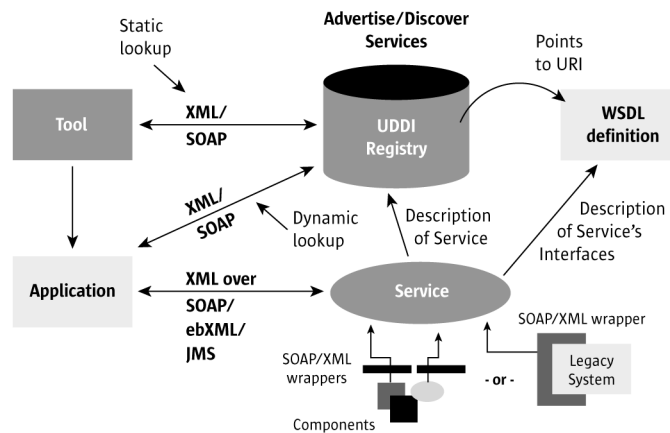


Figure 1–4: Services and Registries

At the center of the diagram is the UDDI registry containing information about the company that provides or operates the Web service, the interfaces that the service provides, and how an application can connect to the service. Interfaces to the Web services can be described in WSDL format.

The UDDI directory can be accessed in the following ways:

- **Static lookup** in which the developer uses various tools to browse through the registry and extract the WSDL definition of the desired service. This interaction is performed by commands expressed in XML and transported over SOAP.
- **Dynamic lookup** in which the application dynamically browses through the directory at runtime, selects the desired service, then binds that service. Once the application is completed or compiled, communication with it is usually achieved using XML over SOAP, although the UDDI registry entry can specify an alternative protocol.

Unfortunately, the UDDI registry currently contains insufficient semantic information to make the dynamic model described above a reality in the short term. Although trivial business-to-consumer applications (order a pizza) are feasible, business-to-business transactions of significant value are another matter. Human users may be able to use the registry information to select appropriate business partners and understand the billing, registration, service-level guarantees, and semantic content of the service, but programmatic access would be challenging.

One solution for this problem is to reach agreements within vertical industries on portfolios of Web services, what the interfaces mean, and what business arrangements will be used. Another is to establish industry-wide standards for business processes, business terms, and frameworks for reliable and secure interactions at runtime.

### 1.2.2.2 ebXML: Enabling Integrated e-Commerce

One of the important functions of the Sun ONE architecture is the support of integrated e-commerce. The Sun ONE architecture embraces ebXML, an important emerging B2B standard that is the result of a joint effort between the United Nations Center for the Facilitation of Procedures and Practices for Administration, Commerce, and Transport (UN/CEFACT) and OASIS, an XML standards body.

Using ebXML, an e-business can publish ebXML message formats, as well as an online description of the steps necessary to conduct business processes. For example, to process a purchase order over the Internet, the PO format is placed into the ebXML registry, along with a description of each step needed to orchestrate the PO process between the partners. A programmer at one of the business' partner companies is able to access the repository and find the information needed to write an application that interacts with the business and its PO processes.

In Figure 1-5, the design-time portion above the dotted line diagrams the process described in the purchase order example. This process includes the creation of a Collaborative Protocol Profile (CPP), which spells out the terms that a company adheres to in order to do business. When the CPPs of two companies that want to do business are coordinated, the lowest common denominator of mutually acceptable business terms makes up the Collaborative Protocol Agreement (CPA).

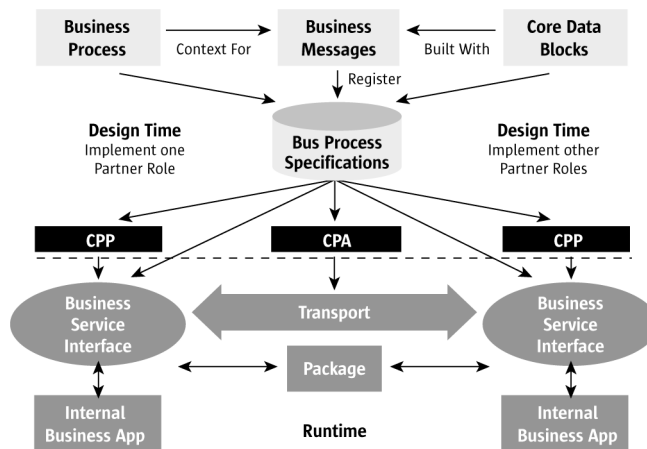


Figure 1-5: ebXML

The runtime portion of the standard deals with the secure, reliable transport of messages from one entity to another. Since commercial exchanges frequently require security and reliability capabilities not provided by SOAP, ebXML messaging builds on the SOAP protocol to provide those necessary capabilities.

The runtime portion of the standard is already being implemented in the industry. The design time portion is being enhanced by OASIS through the creation of a new language known as the Universal Business Language (UBL). UBL will be used by vertical industry segments to describe the semantics associated with e-business XML documents. Many of the industry standards bodies that use XML—for example, RosettaNet and the Online Travel Association—have agreed to align the framework of their standards with ebXML. For a further discussion of ebXML, refer to Section 6.3, “ebXML.”

### 1.2.3 The Java Web Client Model

As described in more detail in Chapter 10, “The Java Web Client Model,” the Java Web client model is a Services on Demand category that allows applications to be downloaded to desktop computers, handheld devices, home gateway computers, or audiovisual devices and set-top boxes. Included are desktop Java technology standards and the use of Java Web Start software to download Java technology applications that, by extension, are being used to create the Java Vending Machine concept. This concept is the basis of an emerging standard that will allow users to purchase or rent applications appropriate to the device they are using, their location, and their individual preferences. The Java Vending Machine also allows service providers to bill for the use of the application and compensate the application’s creator.

## 1.3 Beyond Formal Standards

The Sun ONE architecture is not limited to the formal standards for Services on Demand, such as J2EE and XML. Instead it includes additional middleware interfaces and Core Web services currently available in iPlanet software products—for example, user and group schema definitions and policy, interfaces needed to populate a channel in the iPlanet Portal Server (Portal Server), Sun office productivity functionality, and Sun e-commerce services.

Development tools relevant to Java and Web application services include wrapping legacy languages like C++, or even Fortran, to enable them to speak to Web services. Also included are Web applications/services systems and applications management capabilities that provide control of the major elements of the stack from an operator’s console as well as monitor and manage Quality of Service.

The fact that Platform Services are relevant to Web applications and services containers means that Solaris™ Operating Environment based services, such as resource management, cluster control, and security, are well integrated between the service containers and the underlying OS platform. For a full description of Platform Services, see Chapter 12, “Platform Services.”

## 1.4 Moving Forward

The world of computing is about to be transformed by the new computing model defined by Services on Demand. Service providers and standards bodies are working to make available the final pieces of infrastructure required to unleash the full potential of Web services. ISVs and enterprise developers are conceptualizing the products and applications that will define this phase in the evolution of networked computing.

Sun ONE is Sun’s architecture for delivering Services on Demand. The Sun ONE architecture provides all of the tools and technologies needed for designing, creating, assembling, deploying, executing, and maintaining Services on Demand.

The next chapter provides an overview of the Sun ONE architecture. The remainder of this book provides a detailed description of the tools, technologies, and products that implement it.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

# The Sun™ Open Net Environment (Sun ONE) Architecture

---

The Sun™ Open Net Environment (Sun ONE) architecture supports the creation and delivery Services on Demand applications and services, as described in Chapter 1. This chapter provides an overview of the components of the Sun ONE architecture.

## 2.1 Service Stack

The Sun ONE platform is the basis for achieving Web applications, Web services, and web clients enabled with Java™ technology (“Java Web clients”), collectively referred to as Services on Demand. Figure 2–1 depicts a high-level view of the architecture’s services stack. The architecture is expressed in relatively generic terms because there are different ways to use products to implement the depicted services. The specific Sun products that implement the stack are shown in Section 2.3, “Product Mappings to the Service Stack.”

At the bottom of the figure are the operating system, hardware, storage, and networking platform, and the directory technology required to provide identity, policy, and management services that affect the entire stack. At the top are the tools used to create, assemble, deploy, and test Services on Demand.

In the center of the stack are three layers that are familiar to application programmers as the presentation logic, business logic, and the back-end data-access logic.

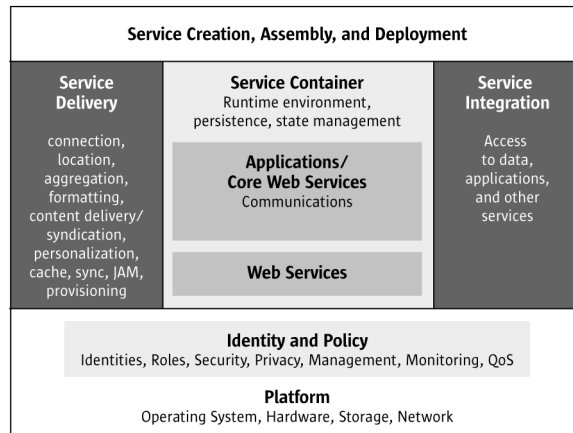


Figure 2–1: Service Stack for Creating Services on Demand

The Service Delivery Box contains the Portal and related services that focus on presentation issues, such as:

- Delivering personalized, context-aware content and services to any device.
- Aggregating content and services.
- Providing secure delivery services.
- Providing knowledge management, content management, and synchronization services.
- Provisioning (delivering) applications for the Java Web client model.
- Providing an application framework for rapid development of complex Web applications.

The Service Container, hosting the business logic for Services on Demand, is typically a Web application server configured with Java™ 2 Platform, Enterprise Edition (J2EE™) technology. Core Web Services, which can be considered “pre-built Web services,” also may be located inside the Service Container. As described in Chapter 13, “Core Web Services,” these services are often exposed as Web services from an existing e-commerce or communications application. Typical services include:

- Large applications such as e-commerce
- Communications services such as calendar and e-mail
- Content services such as office productivity applications



The Service Integration box uses Java technology based connectors (“Java connectors”) such as Java DataBase Connectivity™ (JDBC™) technology, and Enterprise Application Integration (EAI) to integrate legacy applications and databases, to access service registries, and to perform Web services integration within and between enterprises.

## 2.2 Standards Associated with the Sun ONE Platform

As indicated in Figure 2–2, the Sun ONE integratable platform is based on a number of important standards for APIs and protocols, including a host of Java, eXtensible Markup Language (XML), and Web standards. Note that the list of standards shown in the figure is incomplete. Refer to the following sections for explicit requirements on interfaces. Some of the standards listed in the figure appear in italics, which indicates that they are still emerging from their standards committees. Future versions of the Sun ONE architecture will align their interfaces with these standards as they are approved.

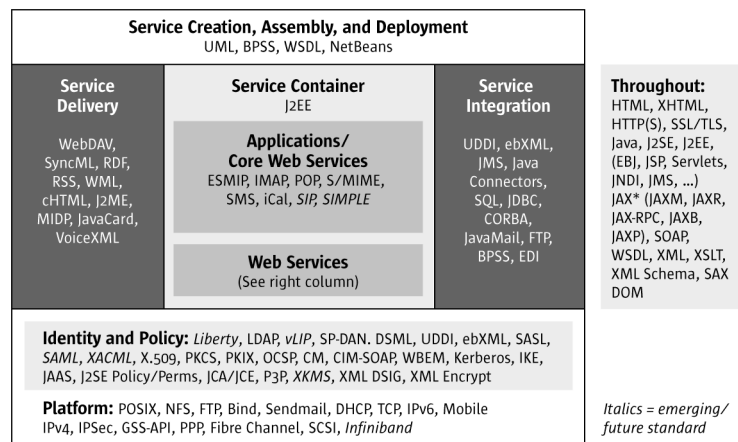


Figure 2–2: Standards Associated with the Sun ONE Platform

## 2.3 Product Mappings to the Service Stack

The Sun ONE Architecture is expressed in terms of the standards and services for delivering Services on Demand. It does not dictate how actual products must be laid out across the services stack diagram to achieve these services. For example, a single product based on application server technology could implement the entire center row of boxes—service container, delivery, and integration—and the Identity and Policy Services as well. However, conformance to the architecture requires that all of the standards for APIs and protocols used to interconnect and manage the boxes be implemented, regardless of the product integration packaging. (Thus, in this single-product example, Light Weight Directory Access Protocol (LDAP) interfaces to the identity information are required, even if that service is implemented with a different underlying technology, such as a relational database management system [RDBMS]).

Figure 2-3 shows how Sun products map to the categories of the services stack. Although the products are available individually, they all share the characteristic of being well-integrated with each other and the Solaris™ Operating Environment in terms of installation, manageability, support, and other factors. Because the Sun ONE platform is integratable, products from vendors that conform to open standards can be used interchangeably. However, the integrated nature of the all-Sun stack is an important value proposition for many developers and enterprise IT departments.

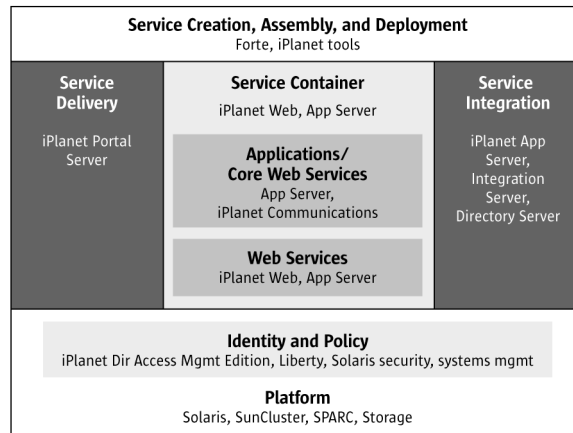


Figure 2-3: Products Delivering the Sun ONE platform

The generic view of the services stack does not depict how real-world products are distributed or connected. Therefore, each vendor will have its own means of describing this. The Sun products are often shown in a diagram similar to Figure 2-4. In this figure, the list of data, applications, reports, and transactions (often referred to as DART) represents the information assets of the enterprise, which can be managed by the Identity and Policy products. They can be created and delivered using the Service Container, Service Integration, and Service Delivery products.

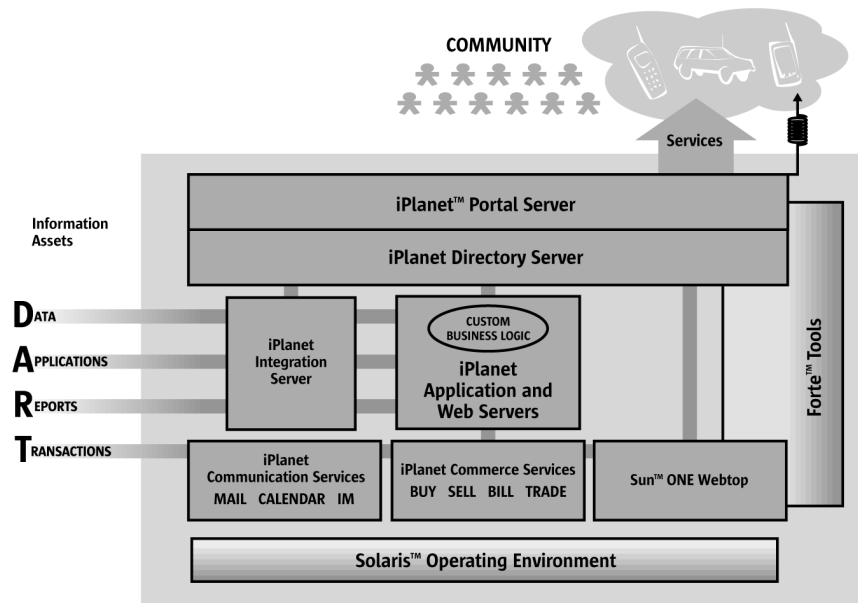


Figure 2-4: The DART Model

## 2.4 Sun ONE Architecture Integration and Interoperability

The Sun ONE architecture is not simply a list of standards. It is an integrated product stack from Sun that includes iPlanet™ software, the Solaris™ Operating Environment (“Solaris OE”), Forte™ tools, and Java technology. It is also an integratable stack, based on open standards for APIs and the protocols used by those products. The definition of this integrated collection of products will be rigorously determined—an effort equivalent to creating a formal standard such as POSIX or Java API’s.

Solaris OE is a valuable foundation of Sun's integrated stack. iPlanet software takes full advantage of the Solaris Operating Environment's advanced features. However, Sun's Java technology and iPlanet and Forte software are committed to a multiplatform strategy. This means that customers can select other operating systems, such as Windows, Linux, AIX, or HPUX for their integratable stacks.

The Sun ONE architecture's interoperability strategy is based on alignment with standard Web interfaces. Connections to, or interoperability with, legacy systems or alternative enterprise technologies such as Microsoft's .NET, are based on the use of standard Web services and integration adapters and connectors.

Interoperability allows customers to leverage their investments in existing software and systems while simultaneously taking advantage of the benefits of the new architecture.

Ineroperability involves two primary dimensions:

- Interoperation with existing applications
- Interoperation with Microsoft .NET

These two dimensions are explained in the sections that follow.

## 2.4.1 Interoperability with Existing Applications

The Sun ONE architecture's ability to interoperate with existing applications is made possible using the following three models supported by the architecture:

- Connectors (synchronous and/or asynchronous) to major applications, such as Enterprise Resource Planning (ERP), Customer Relations Management (CRM), and legacy mainframe. Such connectors to major applications are provided by the application vendors. When appropriate, Sun assists the application vendors in the construction of connectors.
- Containers created with JavaServer Pages™ (JSP™) and Enterprise JavaBeans (EJB™) technology that wrap applications inside components based on the EJB architecture. Once “bean-ized,” the application is easily integrated into a Web Service.
- Web services that completely wrap applications into a standard Web Service.

## 2.4.2 Interoperation with Microsoft .NET

Virtually all corporations have significant investments in Microsoft products and technologies. As such, they are likely to deploy some Microsoft .NET Web services, either intentionally or simply as a consequence of the normal use of Microsoft .NET server products. The Sun ONE architecture is able to consume .NET Web services and provide Web services that may be consumed by the .NET environment.

The Sun ONE architecture's support of XML, Simple Object Access Protocol (SOAP), Universal Description, Discovery, and Integration( UDDI), and Web Services Description Language (WSDL) has been demonstrated to be interoperable with Microsoft's .NET implementations of these same standards.

Web applications enabled by the Sun ONE platform can be delivered to any browser. Furthermore, Web services enabled by the Sun ONE platform ("Sun ONE Web services") can be delivered to any browser supporting Java technology, including Internet Explorer. Through third-party add-ons to Internet Information Server (IIS), JSP technology-based Web services can be delivered to clients of IIS, even alongside Microsoft .NET services being delivered by the same server.

On Microsoft platforms, Visual Studio .NET may be used to develop applications that consume both .NET and Sun ONE Web services. On Forte IDE-supported platforms, Forte for Java software may be used to develop applications that consume both .NET and Sun ONE Web services.

ebXML and the emerging OASIS UBL schemas are likely to become the industry-standard Business-to-Business (B2B) exchange mechanisms. The Sun ONE architecture supports these schemas, and it is expected that Microsoft will support them in its BizTalk Server product.

Files generated by personal productivity applications—in particular those generated by Microsoft's Office suite—are an important component of interoperability. This is because they are extensively used as data delivery vehicles both inside and outside businesses. The office productivity functionality offered by the Sun ONE platform provides a high degree of compatibility with Microsoft Office documents.

## 2.5 Phases of Adoption

Sun's complete vision of Services on Demand is still evolving. Therefore, this book anticipates that the future will arrive in phases as enterprises adopt the new models of computing over time. The phases of adoption described below do not constitute a road map of Sun product releases. Rather, product releases will adapt to the needs and schedules of the adoption cycles.

**Phase 1, Web services**, consists of Web applications and basic XML services. This phase is already here, as exemplified by Sun's full spectrum of products in the form of iPlanet and Forte software, Java technology, and the Solaris Operating Environment. Enough Web services standards exist—including SOAP, UDDI and WSDL—that new Web services concepts and applications can be prototyped and tested. For example, the iPlanet Integration Server was deploying Web services even before the term was coined. Basic XML services are also a part of the Forte for Java software tools.

**Phase 2, Internally Focused Web services**, marks the beginning of the concerted deployment of Web services within the enterprise. Although the enterprise still exerts a high degree of centralized control, this phase uses Web services to integrate major applications within the enterprise in an easier, more loosely coupled fashion than the tightly coupled models of the past. Users, such as employees and business partners, are defined in the enterprise directory, not discovered dynamically. Communications between the enterprise and its business partners are based on bilateral agreements on what Web services to use and how they work together. At this point, Web services depend on pre-defined relationships stored in a service directory; services are not assembled dynamically with unknown business partners.

**Phase 3, Dynamic Web services**, removes these restrictions by taking advantage of emerging federated identity and context services and public service registries populated with semantically rich service descriptions. These service descriptions allow customers and business partners to find and conduct business dynamically. Key to this phase are two pieces of technology: the Liberty Alliance Project methods of providing federated identity service, and standards such as ebXML that provide the semantic definitions and business orchestration that enable dynamic B2B.

The following figures show a simplified depiction of how Sun products have been or will be used to deliver Web services in each of the three phases of implementation.

**Phase 1**, shown in Figure 2-5, includes simple SOAP-based Web services. Developers can create Web services with the Forte for Java software IDE. Web services can be deployed to the iPlanet Portal Server and the iPlanet Application Server. A SOAP Client stub is generated, but any on-the-wire SOAP client can be substituted. The SOAP client uses Synchronous SOAP over HTTP to call the portal/application server. The application server then makes a Synchronous SOAP over HTTP call to the integration server.

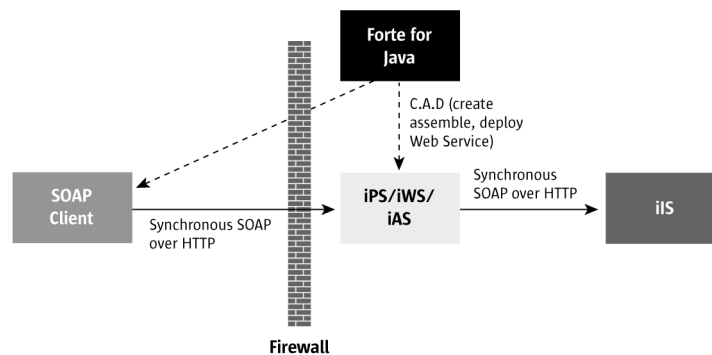


Figure 2-5: Phase 1, Web services

**Phase 2**, shown in Figure 2–6, is the internally focused, enterprise-controlled model of Web services. This phase depicts a UDDI registry server behind the firewall, implying that it is used as a private, internal registry. However, external access is possible for prearranged business partners. Forte for Java software can now deploy a service description to the UDDI server. Developers use the UDDI server and its WSDL descriptions of a Web service to create a Client stub automatically. Asynchronous SOAP is now introduced at each step, allowing guaranteed asynchronous SOAP calls to other systems using iPlanet Message Queue for Java for processing Java Message Service (JMS) messages.

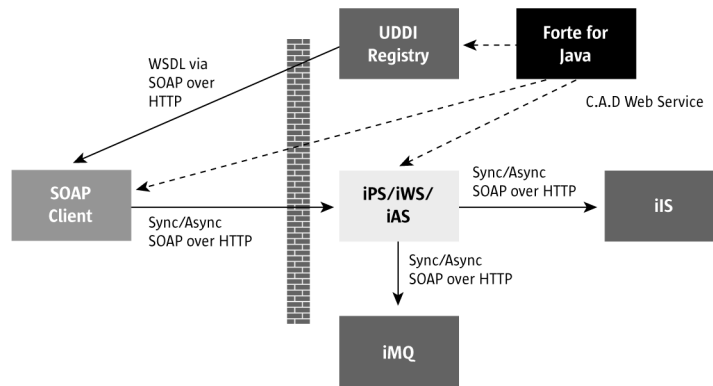


Figure 2–6: Phase 2, Internally Focused Web Services

**Phase 3**, shown in Figure 2–7, is the dynamic model in which all Web services crossing boundaries are authenticated by the federated identity and authentication services established by the Liberty Alliance Project. The UDDI server is shown outside the firewall to imply that it is a public registry, also authenticated by Liberty. Forte for Java software now includes a workflow definition tool integrated within its IDE framework. The iPlanet Integration Server is able to integrate external Web services into its workflow definitions.

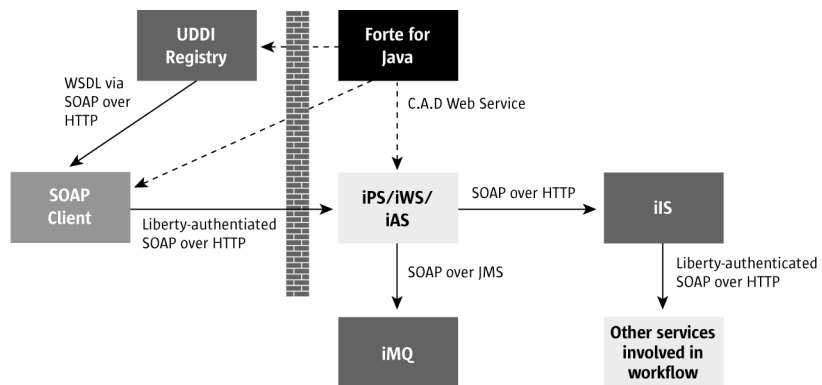


Figure 2–7: Phase 3, Dynamic Web Services

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.



## Part 2. Service Containers

---



## J2EE™ Components and Containers

---

The Java™ 2 Platform, Enterprise Edition (J2EE™ platform) defines the Service Container architecture for the Sun™ Open Net Environment (Sun ONE) architecture. The J2EE technology offers the following advantages:

- A standard, extensible, component-based application programming model.
- Automation of access to many critical services, such as transaction management and persistence.
- Reliable access to a range of additional services, thereby reducing the complexity of developing distributed applications.
- The enabling of Web-services applications and the simplification of their development.

Applications developed with J2EE technology (“J2EE applications”) can integrate a variety of clients, ranging from stand-alone applications on the desktop to Java Extensible Markup Language (XML) and pure HyperText Markup Language (HTML) applications in browsers, as well as to a variety of next generation clients for mobile and personal access to services. J2EE applications can also integrate a range of back-end systems, J2EE applications from relational database management systems (RDBMSs) connected through Java DataBase Connectivity™ (“JDBC™”) interface to legacy Customer Relations Management (CRM) and Enterprise Resource Planning (ERP) systems using the J2EE Connector Architecture.

The J2EE platform provides standard APIs for XML that support the features required for Services on Demand. The supported services include dynamic lookup and access to Web services with the Java API for XML Registries (“JAXR”); object-oriented, synchronous interaction between well-defined application interfaces through the Java API for XML-based RPC (“JAX-RPC”); and models for loosely coupled interaction between applications with the Java API for XML Messaging (“JAXM”). The J2EE specification 1.3 provides these APIs through the Java Web Services Developer Pack, while J2EE specification 1.4 provides them as fully integrated features of the platform.

In addition to these Web-service-focused APIs, two other convenient APIs allow the processing of XML data: the Java API for XML Processing™ (“JAXP”) and the Java Architecture for XML Binding™ (“JAXB”). These APIs provide the ability to process data within a J2EE application as well-defined structures of objects, rather than as raw XML. Combined with the well-defined, component-based development model supported by J2EE, these APIs enable developers to quickly deliver Web service applications. The J2EE platform’s architecture simplifies the development of applications by introducing the concept of redeployable components throughout the layers of multi-tier applications. The support for this architecture is implemented as two fundamental parts: components and containers.

## 3.1 Components

Components represent units of development and deployment that are designed to be simpler to build than other models. They provide standardized functionality, have well-defined application interfaces, and can easily be developed and deployed for specific business purposes. Containers that support the components represent reliable, standardized services and a consistent environment from one product vendor to another. An example of container structure is shown in Figure 3–1.

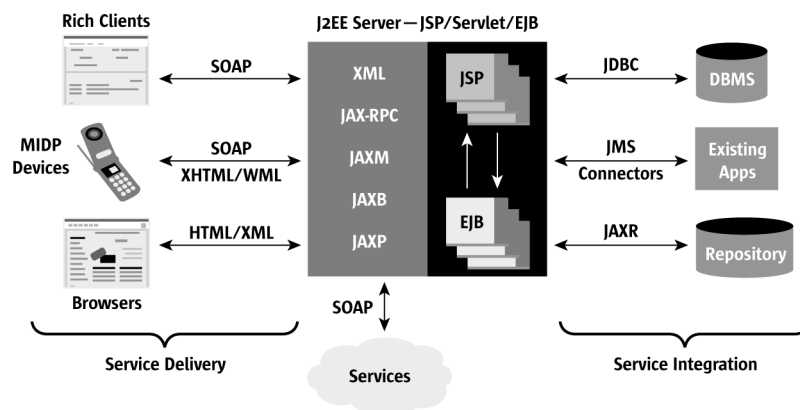


Figure 3–1: J2EE in the Service Container

Containers provide automatic support for certain aspects of application behavior, such as HyperText Transfer Protocol (HTTP) interaction, transaction management, and security. They also offer a set of standardized services that components can employ to perform useful work.

By enabling developers to focus on creating components that encapsulate application specifics—such as graphic look and feel, navigation, and business logic—the J2EE platform reduces the amount of coding and debugging required to develop fully functional applications.

For access to Internet services, the J2EE platform supports the HyperText Transfer Protocol (HTTP), Transmission Control Protocol/Internet Protocol (TCP/IP), and Secure Socket Layer (SSL) protocol. In addition, the J2EE platform supports new eXtensible Markup Language (XML) functionality. XML provides tagged data similar to HTML, but the tags describe the data itself rather than the way the data is displayed. XML can be used to transfer formatted data between applications or servers on the Internet—for example, to support transactions between businesses (B2B). Support for parsing XML and for representing XML as objects is implemented and is being standardized at the time of this writing.

### 3.1.1 Java Servlet API

The Java Servlet API provides a basic mechanism for generating dynamic Web content. Servlets were developed as an improvement over CGI scripts, which are generally platform-specific and are limited in their ability to support rich interaction.

Like all J2EE software components, servlets run in a container implemented by the J2EE platform provider. This container manages a servlet's interaction with its client and provides a rich environment by which the servlet can access various services based on Java technology. A servlet container implements the Java 2 Platform, Standard Edition (J2SE™ platform). This provides servlets with a variety of technologies based on the Java programming language, including the JDBC API, Java Naming and Directory Interface™ (“JNDI”) API, Remote Method Invocation (RMI), a component based on the JavaBeans™ architecture (“JavaBeans component”), and others.

The container also implements features that allow servlets to share information about a particular client and session, overcoming the obstacles generally presented by the stateless HTTP protocol. The flexibility of servlets is enabled through the servlet API, which implements a mechanism for more complex interaction with the requesting client than is possible with CGI. Various servlet methods provide information to the servlet and allow it to respond. Because of the object-oriented programming model, items key to servlet behaviors are provided as objects with a well-defined API.

### 3.1.2 JavaServer Pages™ Technology

JavaServer Pages™ (JSP™) technology is an extension of the servlet technology created to simplify the development of dynamic Web content. Building on the services and JSP software pages presented in this chapter, additional services such as a presentation framework may run in a J2EE technology-based application server.

The iPlanet™ Application Framework performs the functions of a presentation framework. This book treats the iPlanet Application Framework as part of the Service Delivery Layer. The iPlanet Application Framework is discussed in detail in Chapter 8, “Presentation Frameworks.”

JSP technology supports a page-based metaphor that separates dynamic and static Web content. The page created with the JavaServer Pages technology (“JSP page”) defines a static HTML template with embedded calls to code written in the Java programming language to fill in dynamic portions of the page. JSP pages contain the following four types of elements, each with a specific role in the presentation of dynamic content.

- Text elements are normally content that is formatted through standard HTML or XML. These represent the static portion of the page.
- Directives are instructions to the JSP page processor. A JSP page container processes these directives when compiling the page into an efficient executable form.
- Tags invoke JavaBeans components to generate dynamic content or perform other computations. Tag libraries are a powerful feature of JSP technology, which are used to encapsulate specific functionality invoked via HTML tags. These allow the JSP technology to be easily extended in a portable fashion. For example, tag libraries can be implemented to support embedded database queries for an application.
- Scripting elements may be declarations, scriptlets, or expressions. Like tags, scripting elements can be used to perform computations to generate dynamic content. They are useful when standard tags are inappropriate or have not been defined.

The combination of these four elements makes it easy to generate Web pages for client browsers.

### 3.1.3 Enterprise JavaBeans™

In addition to servlets and JSP software components for providing a rich user experience, the J2EE platform includes the Enterprise JavaBeans™ (“EJB™”) component model for transaction-processing support. This component model provides a standard component architecture for building distributed, object-oriented business applications.

EJB technology allows the developer to focus on business logic without having to manage the details of transaction processing, security, load balancing, connection pooling, and other performance concerns in an application server system. These details are automatically handled by the container for EJB technology-based components (“EJB container”) implemented by the J2EE platform provider. The EJB specification is designed to make it easy for container providers to build on legacy systems by “wrapping and embracing” existing technologies.

The EJB specification clearly defines the life cycle of an enterprise bean, from development to deployment to runtime, and clearly divides the responsibility for relieving most concerns about such issues. By interceding between clients and components at the method-call level, containers can manage transactions that propagate across calls and components, and even across containers running on different servers and different machines. The EJB architecture also has the following advantages:

- The EJB technology model can be implemented by known programmers who encode business logic, guaranteeing the integrity of corporate data. Then different user interfaces can be built on top. They are client-neutral: a single EJB may be accessed from a Web client through JSP page or servlets, or it may be invoked directly by a Java application client in a standard two-tier model. Component developers are free to focus on business logic, since containers provide services automatically by interceding in component method calls. A simple set of callback interfaces is all a developer must implement to participate in container-provided services.
- The EJB technology allows the developer to develop business logic without regard to the details of a particular installation. A separate deployment descriptor is used to make customizations when they are assembled and deployed. Deployment descriptors are XML-based text files whose elements declaratively describe how transactions, security, and other installation specifics are to be handled in an EJB technology-based application. A variety of EJB attributes, including the default component transaction type, can be specified at either development or deployment time and enforced through mechanisms built into the container architecture.

A client's view of an EJB remains the same, regardless of the container in which the EJB is deployed. Any container in which it is deployed presents the same interfaces to the client. This extends to containers from various vendors, running against different servers and databases, on diverse systems on a network. This client transparency ensures wide scalability for multi-tier applications. The client view of components based on the EJB specification ("EJB components") is provided through two interfaces—the home interface and the remote one. These interfaces are provided by classes constructed by the container when a bean is deployed, based on information provided by the bean.

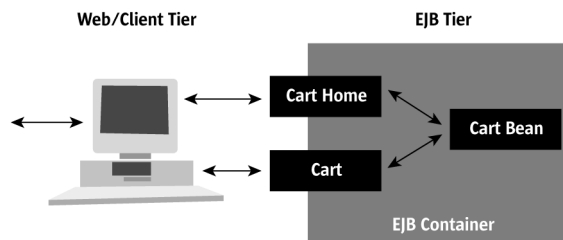


Figure 3–2: EJB Container—Client Interaction

As shown in Figure 3–2, the home interface (cart home) provides methods for creating a bean instance, while the remote (cart) interface provides the business logic methods for the component. By implementing these interfaces, the container can intercede in client operations on a bean and thereby offer the client a simplified view of the component.

To the client, there appears to be direct interaction with EJB components through the home and remote interfaces. However, the EJB architecture is designed to enable clients and components to exist in different runtimes on various systems on a network. The container intercedes between client and component, completely concealing both the bean instance and its own actions from the client.

Container intervention enables transaction management, security constraints, container-managed persistence, and other important features of the EJB component model. The EJB architecture provides automatic support for distributed transactions in component-based applications. Such distributed transactions can atomically update data in multiple databases, possibly even distributed across multiple sites. The EJB component model shifts the complexities of managing these transactions from the application developer to the container provider.

A container supports a variety of transaction properties for beans. Beans can be invoked entirely outside the context of a transaction. They can be required to initiate a transaction when they are called. Furthermore, they can be allowed to participate in an existing transaction when they are called by another bean. In addition to container-managed transactions, an EJB component can participate in client-managed transactions, or it can manage its own transactions using the Java Transaction API specification (JTA).

The EJB component model supports session beans, entity beans, and message-driven beans. Each is designed for specific, well-defined roles, so developers can easily pick the appropriate bean type for each specific architectural requirement.

The three types of beans function as follows:

- **Session beans** represent behaviors associated with client sessions. They are generally implemented to perform a sequence of tasks within the context of a user session.
- **Entity beans** are intended to represent persistent objects, such as a record or a set of related records in a database. Entity beans could be developed to represent business records, such as a customer (name, address, phone) or a purchase order (customer, items purchased, purchase price). Entity bean methods provide operations for acting on the data represented by the bean. The entity bean provides a mechanism for multiple users of an application to have shared transactional access to data. Because of their transactional, data-specific nature, entity beans are designed to be persistent and robust.
- **Message-driven beans** provide a mechanism for constructing loosely coupled applications that can communicate indirectly, using the queuing and subscription models supported by the Java Message Service (JMS) API.



## 3.2 J2EE Platform Container Provided Services

J2EE software containers support the component-based application programming model in two major ways. First, they automate much of the standard functionality that requires programming expertise in other models, such as transaction management and security. Second, they provide standardized APIs in the Java programming language for other features of value to components, such as messaging (JMS) and database access (JDBC).

Because containers are based on the J2SE platform, they provide standard features of the Java runtime environment automatically. These features include cross-platform development support and memory management to simplify debugging. In addition, the J2EE platform and component specifications define features and enhancements to containers that include security management, transaction management, life-cycle management, and other features.

Containers provide a working environment for their components. They also offer a way for services to be “injected” into the operations of the components, without the component developer needing to write specific code. This is especially important in distributed application development, in which the complexity of providing such services may be daunting.

One example of container intervention in a component is container-managed transactions in the EJB architecture. Container-managed transactions let multiple EJB components automatically work together in the same transaction, without the developer of each component needing to know or program any of the transaction details. This facilitates assembling applications from preprogrammed components.

All J2EE software components can use security mechanisms that are built into the platform. Containers can control access to components through these security mechanisms, checking a client’s access privileges for individual methods or a whole interface. In addition, components’ security attributes can be specified at deployment time to ensure that the application’s security model maps to the deploying organization’s security environment.

The containers supporting the J2EE software components provide a number of standardized services, specific to the needs of distributed, enterprise applications. These include:

- Web services, including XML registry access, XML object interaction based on Simple Object Access Protocol (SOAP), ebXML-based messaging, and XML data processing.
- Communication services, including Remote Method Invocation/Internet Inter-ORB Protocol (RMI-IIOP), Java IDL API, the Java Message Service, and JavaMail™.
- Connection services, including the JNDI API for Java naming and directory services, JDBC API for database access, and the Java connector architecture for accessing existing enterprise components from Web components and EJB server side components.
- Internet services, including support for HTTP, TCP/IP, SSL, and XML via a variety of APIs.

## 3.2.1 Web Services

The APIs for XML specified in the J2EE platform allow Web applications to be written entirely in the Java programming language. These APIs fall into two broad categories: those that deal directly with XML documents and those that deal with procedures.

J2EE platform support for XML is as follows:

- Document-oriented JAXP processes XML documents using various parsers.
- JAXB maps XML elements to classes in the Java programming language.
- Procedure-oriented JAXM sends SOAP messages over the Internet in a standard way.
- JAXR provides a standard method for accessing business registries and share information.
- JAX-RPC sends SOAP method calls to remote parties over the Internet and receives the results.

These J2EE platform XML APIs support industry standards to ensure interoperability and flexibility. For example, JAXP code can use various tools for processing an XML document, and JAXM code can use various messaging protocols on top of SOAP. These APIs for XML define strict compatibility requirements to ensure that all implementations deliver the standard functionality, but platform vendors are able to provide implementations tailored to specific uses.

The Java APIs for XML support the following XML-based standards on which interoperable Web services depend:

- SOAP
- WSDL
- UDDI

The J2EE platform provides a variety of tools for developers creating Web services, as described in the following subsections.

### 3.2.1.1 XML Document Parsing API (JAXP)

JAXP leverages the parser standards SAX (Simple API for XML Parsing) and DOM (Document Object Model) so that the developer can choose either to parse data as a stream of events or to build an object representation of it. JAXP also supports the XML Stylesheet Language Transformations (XSLT) standard, providing control over the presentation of the data and enabling developers to convert the data to other XML documents or to other formats, such as HTML. In addition, JAXP provides namespace support in order to work with XML schemas that might otherwise have naming conflicts.

Any XML-compliant parser can be used from within the application. A pluggability layer allows the developer to plug in an implementation of the SAX or DOM APIs. The pluggability layer also allows the developer to plug in an XSL processor to control how XML data is displayed.

### 3.2.1.2 XML-to-Java Object Binding Facility (JAXB)

With the JAXB facility, the developer can create two-way mapping between XML documents and Java objects. Given an XML schema, the JAXB compiler generates a set of Java class files that contain all of the code to parse XML documents based on the schema. A developer using the generated classes can build a Java object tree representing an XML document, manipulate the content of the tree, and regenerate XML documents from the tree.

### 3.2.1.3 SOAP RPC API (JAX-RPC)

JAX-RPC enables procedure calls between clients and remote servers. Servers define a service as a collection of procedures callable by a remote client, and the client then calls the procedures to make use of the service.

In JAX-RPC, remote procedure calls are represented using the SOAP XML-based protocol, which defines a convention for representing remote procedure calls and responses. Web services applications can define, describe, and export capabilities as RPC-based services. The Web Service Description Language (WSDL) specifies the format for describing a service as a set of endpoints on messages. An abstract description of such services can be bound to an XML-based protocol and underlying transport.

By basing its procedure-calling mechanism on the SOAP XML standard, JAX-RPC enables interaction between a variety of servers and clients, including both J2EE technology-compatible and non-J2EE technology-compatible clients and servers.

### 3.2.1.4 Business Registry and Repository (JAXR)

JAXR provides a convenient way to access standard business registries over the Internet. Business registries contain listings of businesses and the products or services they offer. JAXR gives developers writing applications in the Java programming language a uniform way to use business registries that are based on open standards such as ebXML or industry-consortium-led specifications such as Universal Description, Discovery, and Integration (UDDI).

Businesses can register themselves with a registry or discover other businesses with which they might want to do business. In addition, they can submit material to be shared and search for material that others have submitted. Because the schema is stored in a standard business registry, both parties can use JAXR to access it.

### 3.2.1.5 SOAP Messaging API: JAXM

JAXM provides a standard way to send messages over the Internet from the Java platform. Based on the SOAP 1.1 and SOAP with Attachments specifications, it can be extended to work with higher-level messaging protocols such as ebXML or BizTalk. In order to do JAXM messaging, a business uses a messaging provider service, which does the behind-the-scenes work required to transport and route messages. The messaging provider implements the JAXM API, much like the way in which a driver for a database implements the JDBC API.

All JAXM messages go through the messaging provider, so when a business sends a message, it first goes to the sender's messaging provider, then to the recipient's messaging provider, and finally to the intended recipient. It is also possible to route a message to go to intermediate recipients, called actors, before it goes to the ultimate destination. Because all messages go through a messaging provider, it can take care of housekeeping details such as assigning message identifiers and keeping track of whether a message has been delivered before. A messaging provider can also try resending a message that did not reach its destination on the first attempt at delivery.

The client using JAXM technology is totally unaware of what the provider is doing in the background. The JAXM client simply makes JAXM method calls, and the messaging provider—working with the container, if there is one—makes everything happen.

Although it is not required, JAXM messaging usually takes place within an EJB container, generally a servlet or a J2EE platform container. A container includes a listener, which makes it possible to receive messages asynchronously. The listener receives the message as one operation, and the recipient sends a reply as a subsequent operation, thus making the messaging asynchronous.

A JAXM message is made up of two parts—a required SOAP part and an optional attachment part. The SOAP part, which consists of a SOAPEnvelope object containing a SOAPHeader object and a SOAPBody object, can hold XML data as the content of the message being sent. To send one or more complete XML documents, or content that is not XML data, the message will need to contain an attachment part. There is no limitation on the content in the attachment part, allowing images or any other kind of Multipart Internet Mail Extension (MIME)-encoded content to be sent.

### 3.2.1.6 Internet Mail API (JavaMail)

The JavaMail API supports a different kind of asynchronous messaging: electronic mail. The JavaMail implementation supports widely used Internet mail protocols, allowing J2EE software components to send mail to users—for example, to confirm an online order. JavaMail abstract classes may be subclassed to support new protocols and functionality.

## 3.2.2 Cross-platform and CORBA Interoperability

To better support distributed applications with containers running on multiple machines, as well as to enable enterprise applications to communicate with one another more effectively, the J2EE platform supports several standard communication technologies. These include Remote Method Invocation (RMI)-Internet Inter-ORB Protocol (RMI-IIOP), Java IDL, JMS, and JavaMail as means of communicating on a network, sending messages, or invoking services.

The Object Management Group (OMG) has defined the Common Object Request Broker Architecture (CORBA) to allow object interfaces to be defined and invoked in a variety of programming languages and environments on a network. CORBA objects are defined using OMG's IDL. OMG has standardized Java IDL API, allowing objects written in the Java programming language to participate in a distributed CORBA environment.

Java IDL API is now required as part of both the J2SE and J2EE programming environments. It allows objects written in the Java programming language to invoke other CORBA objects written in other languages, and vice versa, via OMG's Internet InterORB Protocol. The use of Java IDL API requires that an IDL definition be compiled into Java programming language stubs and skeletons to support Java technology clients and servers.

RMI-IIOP is a simpler alternative to Java IDL API. It allows interfaces to be defined in the Java programming language instead of in IDL. The remote interface can be converted to IDL and implemented or invoked in another language, since RMI-IIOP uses the same on-the-wire protocol as does Java IDL API (namely, IIOP). RMI-IIOP thus provides interoperability with CORBA objects implemented in any programming language. The J2EE platform allows EJB components to be invoked via RMI-IIOP.

In contrast to Java IDL API and RMI-IIOP, the Java Message Service provides an API for asynchronous messaging. Rather than invoke a service and wait for a response, a JMS message is queued for delivery, and control returns to the invoker. In addition to supporting specific message queues (for example, a queue for a specific EJB components), Java Message Service supports publish-and-subscribe messaging. In this type of messaging, any number of clients can subscribe to (that is, request messages on) well-known topics in a hierarchy of topics, and any number of clients can publish to (that is, send messages to subscribers of) a specific topic. Java Message Service supports reliable, guaranteed delivery of messages.

### 3.2.3 Access to Database Servers

For access to database management systems and other existing enterprise computing resources, the J2EE platform provides support for JDBC, JNDI, and connectors.

JDBC technology-based driver ("JDBC driver") provides J2EE's database connectivity. Structured Query Language (SQL) commands or queries can be issued to a relational database, and the results returned to any Java application environment. The JDBC API supports stored procedures, transactions, connections, and user authentication. JDBC drivers may support connection pooling, distributed transactions, and caching of rows from the database.

For more information, refer to Chapter 6, "Business Process Integration."

## 3.2.4 Access to Name and Directory Servers

The JNDI API provides access to a naming environment. It also provides methods for performing directory operations, such as associating attributes with objects and searching for objects using their attributes.

The purposes of the JNDI API are many. For example, JDBC data sources and JTA transaction objects can be stored in a JNDI API naming environment. A container provides an environment to its components via a JNDI *naming context*. Components in a distributed application can use JNDI API to locate one another and initiate communications. Existing corporate directory services can be accessed via JNDI API.

For more information, refer to Section 11.1.1, “iPlanet™ Directory Server Products.”

## 3.2.5 Component and Container Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for J2EE components and containers.

Interface Name	Level	Status	Reference *	Comments
J2EE 1.3 Specification	Application and System	Footnote 1	<a href="http://www.jcp.org/jsr/detail/58.jsp">http://www.jcp.org/jsr/detail/58.jsp</a>	(JSR 58)
Java API for XML Processing 1.1 (JAXP)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/63.jsp">http://www.jcp.org/jsr/detail/63.jsp</a>	(JSR 63)
Java API for XML Binding (JAXB)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/31.jsp">http://www.jcp.org/jsr/detail/31.jsp</a>	(JSR 31)
Java API for XML Messaging 1.0 (JAXM)	Application	Footnote 2	<a href="http://www.jcp.org/jsr/detail/67.jsp">http://www.jcp.org/jsr/detail/67.jsp</a>	(JSR 67)
Java API for XML Registries 1.0 (JAXR)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/93.jsp">http://www.jcp.org/jsr/detail/93.jsp</a>	(JSR 93) Expect specification to be final in April 2002
Java API for XML-based RPC (JAX-RPC)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/101.jsp">http://www.jcp.org/jsr/detail/101.jsp</a>	(JSR 101) Expect specification to be final in June 2002
J2EE Software Management	System	Footnote 3	<a href="http://www.jcp.org/jsr/detail/77.jsp">http://www.jcp.org/jsr/detail/77.jsp</a>	(JSR 77)
J2EE Software Application Deployment	System	Footnote 3	<a href="http://www.jcp.org/jsr/detail/88.jsp">http://www.jcp.org/jsr/detail/88.jsp</a>	(JSR 88)

Interface Name	Level	Status	Reference *	Comments
SOAP Version 1.1	System	Footnote 1	<a href="http://www.w3.org/TR/2000/NOTE-soap-20000508">http://www.w3.org/TR/2000/NOTE-soap-20000508</a>	
Web Services Description Language (WSDL) 1.1	System	Footnote 1	<a href="http://www.w3.org/TR/2001/note-wsdl-20010315">http://www.w3.org/TR/2001/note-wsdl-20010315</a>	
UDDI Version 2.0 API	System	Footnote 1	<a href="http://www.uddi.org/">http://www.uddi.org/</a>	

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

## 3.3 The Application Server

All platforms that conform to the Sun ONE architecture require a J2EE technology-conformant service container to run and manage Java software applications, Web servers, EJB components, and Web services. To satisfy this need, most enterprises choose a J2EE-conformant application server, which performs all of the functions listed in the Service Container box of Figure 3-3. In addition, it performs many of the functions of the Service Integration box and may also perform some of the functions shown in the Service Delivery box.

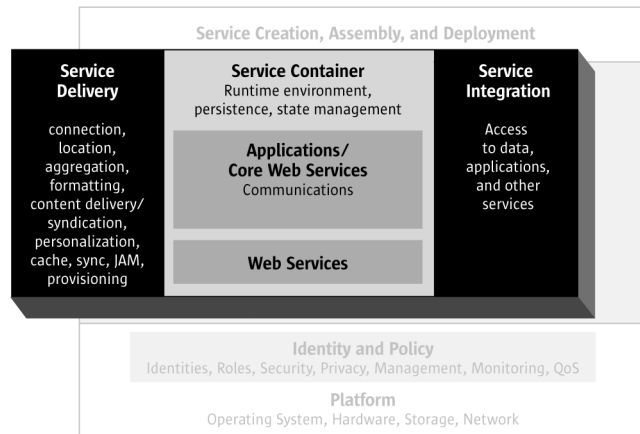


Figure 3–3: Position of the Application Server in the Sun ONE Architecture

The J2EE platform standards permit application server vendors to add functionality to the generic service container to create application server products that differ widely in terms of feature sets. All application server products that conform to the J2EE specification perform the functions of a Service Container as prescribed by that specification. At the same time, each application server product offers a unique set of extended features, performance characteristics, and price to meet the needs of the market segment for which it was intended.

As is the case with other architectural components such as portal servers, web servers, integration servers, and directory servers, Sun offers a fully integrated application server product—the iPlanet Application Server—that satisfies the architectural requirement for a J2EE platform service container. Consistent with the generic description of application servers given earlier, the iPlanet Application Server extends the basic service functionality to provide a rich product offering, as described in the next section.

### 3.3.1 The iPlanet™ Application Server

The iPlanet Application Server is a fifth-generation architecture that provides high levels of performance, scalability, and reliability in a J2EE technology-conformant application server. For those customers who want to purchase all components of their Sun ONE platform from Sun, the iPlanet Application Server offers a solution that is tightly integrated with the Sun ONE platform. The following subsections describe specific attributes of the iPlanet Application Server.



### 3.3.1.1 Deployment Options

The J2EE specification offers the concept of a single Execution Container. The Service Container is, however, an abstraction, and the J2EE specification is flexible enough to permit it to be implemented in a variety of ways. Taking advantage of this flexibility, the iPlanet Application Server supports four deployment modes:

- A single logical container instance implemented on a single hardware node.
- A single logical container instance implemented on multiple hardware nodes.
- Multiple logical container instances implemented on a single hardware node.
- Multiple logical container instances implemented on a multiple hardware nodes.

The last three deployment options offer opportunities for enhancing scalability, performance, and high availability.

### 3.3.1.2 Scalability

When deployed across multiple hardware nodes while functioning as a single logical container, the iPlanet Application Server performs internal load-balancing. In addition, multiple instances of the iPlanet Application Server can be deployed behind an external load balancer.

### 3.3.1.3 High Availability

For customers requiring high transaction integrity and continuous uptime, the iPlanet Application Server eliminates single points of failure through application failover at multiple levels for JSP framework, Java Servlet API, and EJB components. The iPlanet Application Server also helps ensure that user information and application data are not lost during a failure by distributing transaction state and session information across multiple servers. The administration tools can also be used to facilitate hot upgrades of hardware or software.

### 3.3.1.4 Management

The iPlanet Application Server provides management tools for the following task groups:

- Adding and configuring resources for use by databases, legacy enterprise information systems, messaging-oriented middleware systems, and other applications. The application server implements resource connection pooling.
- Configuring, monitoring, and tuning the server itself.
- Deploying, modifying, and managing deployed applications.

The iPlanet Application Server supports secure remote administration. It also provides logging and alert management tools to facilitate administration.

### 3.3.1.5 Tools Integration

While many products can be integrated into the Sun ONE architecture as replacement components, the Sun products provide the benefit of tighter integration, which provides the following benefits:

- Integration with Forte™ developer tools for efficient enterprise-class team development using JSP and EJB technology and the Java Servlet API.
- Support for “hot” redeployment of applications while debugging, which shortens the develop-debug-deploy cycle. The application server can pick up certain types of application changes, without requiring a restart.
- Support of the iPlanet Application Framework, offering the development benefits of using proven J2EE technology-based design patterns in conjunction with the Forte integrated development environment (IDE).
- Provision of plug-ins for integration between the iPlanet Application Server and third-party development environments. Using the Java technology developers can choose between several leading integrated development tools to build and deploy J2EE applications rapidly on the iPlanet Application Server.
- Provision of GUI-based interfaces for creating and modifying XML descriptors for standard J2EE platform and iPlanet software value added features.

### 3.3.1.6 Component Life Cycle Optimizations

J2EE specifications allow some optional optimizations to the bean life cycle, such as various commit options for Entity beans. Some vendors go even further and provide variations on the life cycle that are not portable, but which provide better performance in specific situations. The iPlanet Application Server product line supports component life-cycle optimizations, and will add more in future releases.

### 3.3.1.7 Platform Integration

When running on the Solaris™ Operating Environment (“Solaris OE”), the iPlanet Application Server will take advantage of the services that the Solaris OE provides for performance and scalability. It will use the underlying logging, file, patching, and security facilities to enhance administration. Platform integration reduces training and administration costs at a data center, because administrators use familiar tools and patterns to accomplish common tasks. Performance improvements in the Solaris OE will transparently benefit J2EE technology-based applications.

### 3.3.1.8 Interchangeable Components

The iPlanet Application Server supports the use of add-on or replacement components. Users of the iPlanet Application Server can choose between multiple providers for subsystems for JDBC-compatible drivers, security and access policy managers, asynchronous reliable messaging (JMS), and container-managed persistence.

## 3.3.2 Application Server Interfaces

The iPlanet Application Server supports all of the interfaces described in Section 3.2.5, “Component and Container Interfaces.” In addition, it provides the extended functionality described in Section 3.3.1, “The iPlanet™ Application Server.” Interfaces for this extended functionality are not yet required parts of the Sun ONE architecture. However, interface definitions for some or all of this extended functionality might be provided in a future version.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.



## Part 3. Service Integration

---



# The J2EE™ Connector Architecture and Web-Service–Based Integration

The Sun™ Open Net Environment (Sun ONE) architecture provides three facilities for integrating applications written for the Sun ONE platform (“Sun ONE applications”) with existing Enterprise Information Systems (EISs). EISs include commercial systems such as enterprise resource planning (ERP) and customer-relations management (CRM) systems as well as the many custom systems that enterprises have developed to meet their special business requirements.

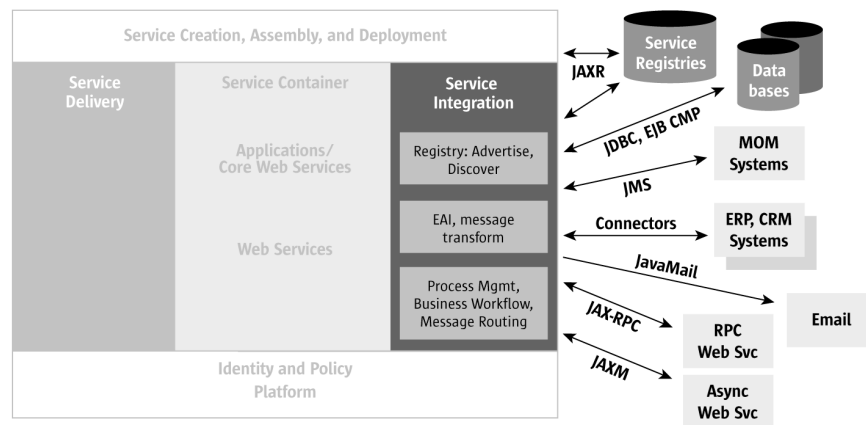


Figure 4–1: Service Integration

## 4.0.1 Overview of the EIS Integration Facilities

EIS applications can be integrated through the following three Sun ONE architecture integration facilities:

- The J2EE™ Connector Architecture
- Asynchronous reliable messaging
- The Sun ONE architecture's native support for Web services

The Java™ 2 Platform, Enterprise Edition Connector Architecture defines a standard way to extend the service container within the Sun ONE platform ("Sun ONE Service Container") to include the functionality of an Enterprise Information System (EIS). With the appropriate connector installed, a Sun ONE application is able to use the functionality of the EIS without having to deal with the complexity of integrating EIS remote access, transactions, and security. The functionality of the EIS appears to the developer as a new service provided by the service container. The first part of this chapter concentrates on the EIS integration capabilities of the connector architecture.

In cases where more loosely coupled integration is desired, Sun ONE architecture provides the standard Java Message Service (JMS) API asynchronous reliable messaging mechanism for integrating Sun ONE applications with an enterprise's Message Oriented Middleware (MOM) environment. This facility allows Sun ONE applications to exchange messages with the EIS systems and other enterprise applications that form an enterprise's existing MOM infrastructure. Because the Sun ONE architecture recognizes that messaging is an important integration facility, all Sun ONE Service Containers provide built-in asynchronous reliable messaging systems. This allows Sun ONE applications to rely on the existence of messaging facilities, even though such facilities may not be otherwise present in the enterprise. For a full discussion of the Sun ONE architecture's asynchronous reliable messaging component, see Chapter 5, "Asynchronous Reliable Messaging."

The third Sun ONE architecture EIS integration facility is that provided by its native support for the Web- service distributed computing paradigm. The Sun ONE Service Container makes it simple both to use an existing Web service from a Sun ONE application and to implement a Web service as a Sun ONE application.

By their nature, Web services easily cross machine and software boundaries. This is due to the fact that they are based on the open protocols of the Internet. Consequently, Web services are ideally suited for solving integration problems. However, it should be noted that this technology is still evolving and should be used with caution. While it is potentially powerful, the versions available at the time of this writing still have performance, complexity, and security limitations that need to be factored into Sun ONE architecture design decisions. The second part of this chapter focuses on the integration capabilities of the Sun ONE architecture's native support for Web services.



## 4.0.2 Overview of the Connector Architecture

The connector architecture defines a standard architecture for connecting the J2EE platform to heterogeneous EISs. It addresses the key issues and requirements of EIS integration by defining a set of scalable, secure, and transactional mechanisms that enable the integration of EISs with service containers and enterprise applications.

## 4.0.3 Advantages of the Connector Architecture

Using the connector architecture reduces the scope of integration and simplifies application development. It also makes it easier to use tools for EIS integration and avoids vendor lock-in.

The connector architecture provides the most direct mechanism for integrating a Sun ONE application with an EIS. This mechanism results in a close coupling between a Sun ONE application and an EIS, which provides both high performance and high reliability. The connector architecture achieves this close coupling by minimizing the layers of software between the application and the EIS, and by directly coupling the service container transaction and security services with those of the EIS.

Another advantage of the connector architecture is that it minimizes the complexity needed to integrate an enterprise's custom EISs with the Sun ONE Service Container. These EISs are typically orphaned, because most middleware systems require arcane knowledge to integrate with their proprietary architectures. The Sun ONE connector architecture is specifically designed to make this "one-off" integration job practical.

## 4.0.4 Connector Architecture Contracts

EIS vendors or third-party independent software vendors (ISVs) specializing in enterprise application integration use the connector architecture to develop standard resource adapters for different EIS types. Because these resource adapters conform to the connector architecture specifications, they can plug into any J2EE technology-compliant service container and can provide connectivity between the EIS, the service container, and the Sun ONE application.

The connector architecture allows a service container and resource adapter (and its underlying EIS) to collaborate to keep all system-level mechanisms—remote access, transactions, security, and connection pooling—transparent to the application. As a result, an application developer can focus on the development of business and presentation logic for its application components, and does not need to get involved in the system-level issues related to EIS integration.

To accomplish its goals, the connector architecture defines two types of contracts:

- Three system-level contracts between a service container and a resource adapter.
- An application contract between a Sun ONE architecture component and a resource adapter.

These contracts are discussed in the sections that follow.

#### 4.0.4.1 System-Level Contracts

The connector architecture's system-level contracts define a "pluggability" standard between service containers and EISs.

The EIS vendor or resource-adapter provider implements its side of the system-level contracts in a resource adapter. A resource adapter is a Java programming language library that is specific to the EIS. It is designed to provide connectivity to the EIS. By plugging a resource adapter into a service container, the container's functionality is extended with connectivity to the resource adapter's EIS. Examples of resource adapters include ERP system resource adapters and CICS resource adapters.

In addition to a resource adapter's system-level contracts with a container, each resource adapter provides an EIS-specific interface that represents the EIS's functionality as seen by components. (Note that the connector architecture does not define this interface).

The resource adapter abstracts the details both of the interface and of the communication between the underlying resource adapter library and the EIS system. Typically, the EIS and the resource adapter communicate over an EIS-specific protocol. A resource adapter can also use a native library as part of its implementation.

The service container is a J2EE technology-compliant server that hosts Web containers and containers for Enterprise JavaBeans™ technology-based components ("EJB™ containers"). The service container also provides a set of services—including transaction management, security services, and connection pooling. The resource adapter uses the Connector Service Provider Interface (SPI) to plug in to a service container by implementing the EIS side of these contracts. The service container, through its Connector SPI, manages a resource adapter's transactions, security, and connection pooling.

Currently, the Connector SPI is composed of the three contracts listed below. They will be extended in the next version of the connector architecture to provide support for thread management and asynchronous communication with EISs.

**Connection management contract** – This contract enables a service container to pool connections to an underlying EIS. At the same time, it enables application components to connect to an EIS. The pooling of connections is important to the creation of a scalable application environment, particularly when large numbers of clients require access to the underlying EIS.

**Transaction management contract** – This contract is between the transaction manager that is provided with the service container and an EIS that supports transactions. It gives a service container’s transaction manager the ability to manage transactions across multiple EIS resource managers. (A resource manager provides access to a set of shared resources.) The contract also supports transactions that do not involve an external transaction manager; that is, local transactions that an EIS resource manager handles internally.

**Security contract** – This contract enables secure access to an EIS. It provides support for a secure application environment and protects the EIS-managed resources.

#### 4.0.4.2 Application-Level Contract

The connector architecture also defines an application-level contract between an application component and a resource adapter. In particular, this contract defines the client API that an application component uses for EIS access. The client API may be the Common Client Interface (CCI), or it may be an API specific to the particular type of resource adapter and the underlying EIS. Java DataBase Connectivity™ (JDBC™) technology is an example of a client API specific to one type of resource adapter—in this case, a relational database.

The CCI defines a common client API for accessing multiple heterogeneous EISs. This API is well suited for enterprise application integration (EAI) and enterprise tool vendors.

#### 4.0.5 Packaging and Deployment

Because the connector architecture emphasizes the pluggability of resource adapters into service containers, it also provides a standard packaging model for resource adapters and a deployment model that enables such adapter pluggability.

A resource-adapter provider is expected to develop a resource adapter according to the connector architecture’s packaging model. By adhering to this model, the server’s deployment tools can easily deploy the packaged resource adapter in the service container’s operational environment.

#### 4.0.6 Web-Service-Based Integration

At first glance, the connector architecture and Web services appear to be on opposite ends of the integration spectrum. From a functionality standpoint, this is true. Connectors allow Sun ONE applications to interact with an EIS via fine-grained, closely coupled operations. On the other hand, integration with a Web service typically occurs through coarse-grained, loosely coupled operations.

The linkage between these two integration mechanisms is that the Sun ONE Service Container plus a resource adapter is an ideal building block for putting a Web service face in front of an EIS. The Web service is simply a Sun ONE Web service application that provides a customized Web service view of EIS functionality. The EIS resource adapter is the shortcut for customizing EIS functionality as a Web service.

A Web service is a new and evolving service delivery model that is based on the Simple Object Access Protocol (SOAP) standard. The basic elements of this service model are transport (SOAP/ HTTP), data encoding (XML Schema), service description (WSDL), and service lookup (UDDI). This chapter will focus on the significance of this model for service integration and on the facilities the Sun ONE architecture provides for supporting it.

Even though there is still much to be done in the standards arena to fully define Web services, developers are using Web services today to solve some classes of integration problems.

The promise of Web services is a secure, unified service delivery model that is based on open, interoperable standards. The reality is that much of the security work has yet to be done, and that the first generation of this technology is therefore best suited to coarse-grained exchange of XML-structured data between trusted applications within a secure environment.

Even with these limitations, Web services is still an important integration technology that developers are already using to open up the flow of information within the enterprise.

## 4.0.7 Support for Web Services in the Sun ONE Architecture

The Sun ONE architecture provides full support for implementing and using Web services. This is an integral part of the Sun ONE architecture service delivery model. Web services are used for integrating service tiers within the Sun ONE architecture. It also uses Web services for service integration between Sun ONE architecture and other environments. For a discussion of the J2EE tools for creating Web services, see Chapter 3, “J2EE™ Components and Containers.”

### 4.0.7.1 Using Web Services from the Sun ONE Architecture

The Sun ONE architecture provides the Java API for XML-based RPC (“JAX-RPC”), which is a full-featured API for using Web services. It can be used uniformly to invoke a Web service within the same service container or across the globe. If a Web service provides a WSDL description of its capabilities, JAX-RPC tools will generate a customized Java API for the service that minimizes the complexity of using it. JAX-RPC also provides lower-level facilities for using services in more dynamic scenarios.

## 4.0.7.2 Implementing Web Services with the Sun ONE Architecture

The primary Sun ONE architecture service delivery model is the broader definition of the Web services model. This definition includes both service delivery to browser-based clients, and service delivery to SOAP-based applications, which is the focus of this section.

Implementing a Web service is as simple as implementing the methods of a JAX-RPC interface. This interface may have been generated by a JAX-RPC tool from a WSDL service description, or it may have been provided as a Java interface from which a JAX-RPC tool will generate its WSDL description. In both cases, the result is an interoperable, SOAP RPC Web service.

The Sun ONE Service Container provides two service component models—servlets and components based on the Enterprise JavaBeans™ specification (“EJB™ components”). Servlets are an informal service component preferred by Web developers. EJB components are a more formal business component model with more support for transactions and state management. Both servlets and EJB components provide direct support for implementing a JAX-RPC interface.

When an EJB component that implements a JAX-RPC interface is deployed in a service container, it is a Web service. The configuration of the service determines how broadly accessible it is. Part of this configuration information is provided by its standard J2EE software application packaging information, while part of it is site-specific and customized by the deployer.

Due to lack of Web service-specific security standards, the security of a Web service is currently limited to the basic link-level authorization, privacy, and integrity facilities of the HyperText Transfer Protocol Secure (HTTPS) protocol. In the not-too-distant future, standards for the use of digital signatures (XMLDSIG) with Web services will provide the basis for Web service non-repudiation facilities.

## 4.0.8 J2EE Connector Architecture Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for connectors based on J2EE technology.

Interface Name	Level	Status	Reference *	Comments
J2EE 1.3 specification (JSR 58)	Application and System	Footnote 1	<a href="http://www.jcp.org/jsr/detail/58.jsp">http://www.jcp.org/jsr/detail/58.jsp</a>	Defines the J2EE 1.3 specification

Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

## Asynchronous Reliable Messaging

---

In an era of e-commerce between loosely integrated business partners, asynchronous operation is an absolute necessity. This is because traditional synchronous messaging has the serious limitation of requiring that all infrastructure elements between distributed components be available at the time of the transaction. Ensuring constant availability is difficult enough within single enterprises; it becomes much more difficult between communities of business partners. Even if all of the business partners agree to high standards of availability and implement costly fault-tolerant systems, the result is to simply reduce, not eliminate, down-time.

The problem of unavailability is an annoyance to customers who try unsuccessfully to access services over the Internet through a Web browser. Unavailability results in lost business when customers give up or make their purchases from another seller whose service is available. However, in the coming age of Web services in which operations will be conducted between machines automatically and without human interaction, the risks related to unavailability are even more daunting without the ability to communicate asynchronously.

### 5.1 Messaging Basics

The term “messaging” is quite broadly defined in computing. It is used for describing systems such as the following:

- Various operating system concepts
- E-mail and FAX products
- Asynchronous communication between enterprise applications

This chapter employs the latter meaning of “messaging.” Messages, as described here, are asynchronous requests, reports or events that are consumed by enterprise applications, not humans. They contain vital information needed to coordinate these systems. In addition, they contain precisely formatted data that describe specific business actions. Through the exchange of these messages, each application tracks the progress of the enterprise.

## 5.1.1 Existing Messaging Systems

Messaging systems are peer-to-peer facilities. In general, each client can send messages to, and receive messages from, any client. Each client connects to a messaging agent that offers facilities for creating, sending, and receiving messages.

Each system provides a way of addressing messages and furnishes a way to create a message and fill it with data. Some systems are capable of broadcasting a message to many destinations. Others only support sending a message to a single destination.

Some systems provide facilities for asynchronous receipt of messages. In such systems, messages are delivered to a client as they arrive. Others support only synchronous receipt in which a client must request each message.

Each messaging system typically provides a range of service that can be selected on a per-message basis. One important attribute is the lengths to which the system will go to ensure delivery. This varies from simple best effort to guaranteed, only-once delivery. Other important attributes are message time-to-live, priority, and whether a response is required.

The iPlanet™ Message Queue for Java™ software messaging system is an example of an existing asynchronous reliable messaging system. The manner in which it is implemented is discussed in Section 5.4, “iPlanet™ Message Queue for Java.”

## 5.1.2 Asynchronous Reliable Messaging Systems

A prerequisite for asynchronous operations within and between enterprises is an asynchronous reliable messaging system. An asynchronous reliable messaging system allows an application to interact with other applications using a local message queue, regardless of whether the remote application is actually available when the application initiates the interaction. The only requirement is for the application to be able to write to its local message queue; it needs to deliver each message exactly once.

The message is delivered by the reliable-messaging provider when the service at the other end is receiving. The application does not need to maintain an open thread waiting for the remote application to receive the message. The availability of asynchronous messaging increases the reliability of systems during periods of peak demand, which is an important consideration for e-commerce applications.

The Enterprise JavaBeans™ (EJB™) 2.0 specification recognizes the importance of asynchronous operation by introducing the message-driven EJB components, which allows EJB components to spawn threads, respond to asynchronous requests, and initiate multiple concurrent actions. The Java Message Service (JMS) API is a natural adjunct to the Message Driven Bean (MDB).



## 5.2 The Java™ Message Service Technology

To address the needs of enterprises for asynchronous reliable messaging, the Java Community created the JMS, which became an integral part of the Java 2 Platform, Enterprise Edition (J2EE™ platform). JMS is a standard API for messaging that supports reliable point-to-point messaging as well as the publish-subscribe model.

The asynchronous reliable messaging component of the Sun ONE architecture is built upon the JMS API specification. The Java Message Service provides a common way for Java programs to create, send, receive, and read an enterprise messaging system's messages.

Enterprise messaging products or, as they are sometimes called, Message-Oriented Middleware (MOM) products, are becoming an essential component for integrating intra-company operations. They allow separate business components to be combined into a reliable yet flexible system. In addition to the traditional MOM vendors, enterprise messaging products are provided by several database vendors and a number of Internet-related companies.

Enterprise messaging provides a reliable, flexible service for the asynchronous exchange of critical business data and events throughout an enterprise. The JMS API adds to this a common API and provider framework that enables the development of portable, message based applications in the Java programming language.

The JMS API improves programmer productivity by defining a common set of messaging concepts and programming strategies that will be supported by all JMS technology-compliant messaging systems.

### 5.2.1 Objectives of the JMS Technology

If Java Message Service provided a union of all the existing features of messaging systems, it would be much too complicated for its intended users. On the other hand, JMS is more than an intersection of the messaging features common to all products. It is crucial that Java Message Service include the functionality needed to implement sophisticated enterprise applications.

The JMS API defines a common set of enterprise messaging concepts and facilities. It attempts to minimize the set of concepts a Java language programmer must learn to use enterprise messaging products. It strives to maximize the portability of messaging applications.

#### 5.2.1.1 Java Message Service Technology Provider (“JMS Provider”)

A JMS provider is the entity that implements the JMS API for a messaging product. Ideally, JMS providers run in applets, simplify installation, and work across architectures and operating systems. An important goal of the JMS API is to minimize the work needed to implement a provider.

### 5.2.1.2 Java Message Service Technology Clients (“JMS Clients”)

JMS clients are the programs or components written in the Java™ programming language that produce and consume messages.

### 5.2.1.3 Java Message Service Technology Messages (“JMS Messages”)

JMS defines a set of message interfaces. Clients use the message implementations supplied by their JMS provider. A major goal of JMS is that clients have a consistent API for creating and working with messages that is independent of the JMS provider.

### 5.2.1.4 Java Message Service Technology Domains (“JMS Domains”)

Messaging products can be broadly classified as either point-to-point (PTP) or publish-subscribe (Pub/Sub) systems.

PTP products are built around the concept of message queues. Each message is addressed to a specific queue; clients extract messages from the queue(s) established to hold their messages.

Pub/Sub clients address messages to some node in a content hierarchy. Publishers and subscribers are generally anonymous and may dynamically publish or subscribe to the content hierarchy. The system takes care of distributing the messages arriving from a node's multiple publishers to its multiple subscribers. The Java Message Service provides client interfaces tailored for each domain.

### 5.2.1.5 Portability

The primary portability objective is that new, JMS-only applications must be portable across products within the same messaging domain. This is in addition to the expected portability of a JMS client across machine architectures and operating systems (when using the same JMS provider).

Although JMS is designed to allow clients to work with existing message formats used in a mixed-language application, portability of such clients is not generally achievable. This is because porting a mixed language application from one product to another is beyond the scope of the JMS technology.

## 5.2.2 Java Message Service Does Not Include

The JMS specification does not address the following areas of functionality:

**Load Balancing/Fault Tolerance** – Many products provide support for multiple, cooperating clients implementing a critical service. The Java Message Service does not specify how such clients cooperate to appear to be a single, unified service.

**Error/Advisory Notification** – Most messaging products define system messages that provide asynchronous notification of problems or system events to clients. Java Message Service does not attempt to standardize these messages. By following the guidelines defined by the JMS specification, clients can avoid using these messages, thus preventing the portability problems introduced by their use.

**Administration** – Java Message Service does not define an API for administering messaging products.

**Security** – Java Message Service does not specify an API for controlling the privacy and integrity of messages. It also does not specify how digital signatures or keys are distributed to clients. Security is considered to be a JMS provider-specific feature that is configured by an administrator rather than controlled via the JMS API by clients.

**Wire Protocol** – Java Message Service does not define a wire protocol for messaging.

**Message Type Repository** – Java Message Service does not define a repository for storing message type definitions and it does not define a language for creating message type definitions.

### 5.2.2.1 Java Message Service Requirements

The functionality discussed in the JMS specification is required of all JMS providers unless it is explicitly noted otherwise. Providers of Java Message Service point-to-point functionality are not required to provide publish/subscribe functionality and vice versa. The following table provides information relevant to the JMS specification.

Interface Name	Level	Status	Reference *	Comments
Java Message Service (JMS) Specification	Application and System	Footnote 1	<a href="http://www.jcp.org/aboutjava/community/process/maintenance/JMS/index.html">http://www.jcp.org/aboutjava/community/process/maintenance/JMS/index.html</a>	Supports message-driven EJBs and Java Transaction API (JTA) transactions.

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

## 5.3 Requirements Beyond Java Message Service

In order to conform with the Sun™ Open Net Environment (Sun ONE) architecture, an asynchronous reliable messaging product must implement all of the JMS API specification. Further more, it must implement the functionality described in the following sections.

### 5.3.1 Multiple Queue Delivery Styles

The JMS API specification does not define specific behaviors when multiple receivers attach to a queue. In fact, the only required capability is that a single receiver be able to attach to a queue. In reality, it is quite common for an enterprise to want to dynamically adjust the number of processing agents consuming messages on a queue. Therefore, it is a requirement that a Sun ONE architecture compliant asynchronous messaging provider be able to support multiple receivers on a queue. That provider must also have the capacity to support at least the following two models for message delivery:

*Failover* – A queue has a primary receiver defined for it, as well as failover receivers. If delivery to the primary receiver fails, messages are routed to a secondary receiver until the primary reconnects to the system.

*Round-robin* – Messages are delivered to all the subscribing clients in a balanced, round-robin fashion.

### 5.3.2 Multiple Protocol Support

Asynchronous messaging systems are useful in both the intranet and extranet world. Because of the firewalls and other hurdles placed between cooperating enterprises in an Internet-style deployment (and between divisions in large intranet situations), it is important for a reliable messaging provider to be able to support multiple protocols. In addition to a sockets-based protocol, a Sun ONE platform provider should support an HTTP mode of transport.

### 5.3.3 Security

The JMS API specification does not require that communication between clients be secure. The Sun ONE architecture requires that it be possible to configure the asynchronous messaging system to make the transmission of data secure.

### 5.3.4 Object Management

A Sun ONE architecture compliant messaging system must support Light Weight Directory Access Protocol (LDAP) as a store for its administered objects. Furthermore, its schema must be integratable with Sun ONE architecture compliant naming systems.

### 5.3.5 Pluggable Persistence

A compliant JMS technology system must allow the developer or deployer to specify high levels of reliability for certain sets of messages in the system. The highest levels of reliability require a media-based persistence mechanism to allow the data to survive across software crashes and errors, or even hardware faults. A Sun ONE architecture compliant implementation must allow the pluggable use of a customer's existing database software instead of a closed, proprietary system delivered with the messaging software.

### 5.3.6 Distributed Transaction Support

While a compliant JMS provider can make a choice about whether to support the XA interfaces for distributed transactions, the Sun ONE architecture requires that these interfaces be supported for compliance.

## 5.4 iPlanet™ Message Queue for Java

The iPlanet Message Queue for Java software messaging system, whose architecture is shown in Figure 5-1, is a current example of an asynchronous reliable messaging system. It implements many of the functionalities described in the previous sections of this chapter.

The iPlanet Message Queue for Java software provides asynchronous reliable messaging through the coordination of the following main components:

- Administered Objects
- Client Runtime
- Message Service

The rest of this chapter contains an in-depth description of these components and the ways in which they work together to provide asynchronous reliable message delivery.

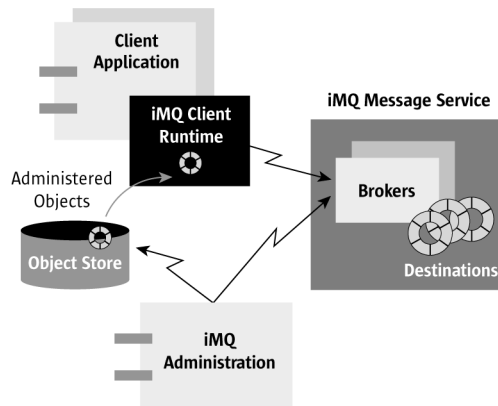


Figure 5-1: Messaging System Architecture

### 5.4.1 Administered Objects

Administered Objects encapsulate provider-specific implementation and configuration information in objects that are used by client applications. Such objects are created and configured by an administrator, stored in a name service, accessed by client applications through standard Java Naming and Directory Services™ (“JNDI”) lookup code, then used in a provider-independent manner.

iPlanet Message Queue for Java software provides two types of administered objects: `ConnectionFactory` and `Destination`. While both encapsulate provider-specific information, they have very different uses within a client application. `ConnectionFactory` objects are used to create connections to the Message Service, while `Destination` objects (which represent physical destinations) are used to identify physical destinations.

Administered Objects make it easy to control and manage a Message Service. That is because the behavior of connections can be controlled by requiring client applications to access preconfigured `ConnectionFactory` objects through a JNDI API lookup. The proliferation of physical destinations can be controlled by requiring client applications to access only `Destination` objects that correspond to existing physical destinations. The broker’s auto-create capability may have to be disabled, as explained in Section 5.4.3.7.3, “Auto-Created (vs. Admin-Created) Destinations.”

This arrangement therefore provides control over Message Service configuration details. At the same time, it allows client applications to be provider-independent. They do not need to know about provider-specific syntax and object naming or provider-specific configuration properties.

## 5.4.2 Client Runtime

As the second main component of the Messaging System, the Client Runtime provides client applications with an interface to the Message Service by supplying them with all the JMS programming objects. It supports all operations necessary to enable clients to send messages to destinations and to receive messages from them.

Figure 5–2 illustrates how message production and consumption involve an interaction between client applications and the Client Runtime, while message delivery involves an interaction between the Client Runtime and the Message Service.

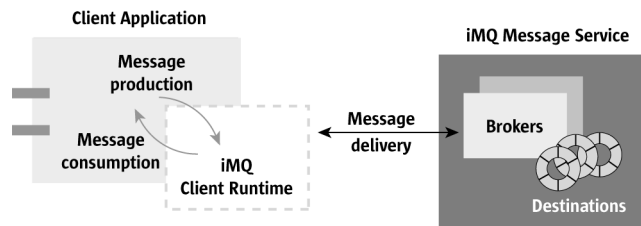


Figure 5–2: Messaging Operations

### 5.4.2.1 Message Production

In message production, a message is created by the client, then sent over a connection to a destination on a broker. If the message delivery mode of the MessageProducer object has been set to persistent (guaranteed delivery, once and only once), the client thread blocks until the broker acknowledges that the message was delivered to its destination and stored in the broker's persistent data store. If the message is not persistent, a broker acknowledgement message (referred to as "Ack" in property names) is not returned by the broker, and the client thread does not block.

### 5.4.2.2 Message Consumption

Message consumption is more complex than production. Messages arriving at a destination on a broker are delivered over a connection to the Client Runtime under the following conditions:

- The client has set up a consumer for the given destination.
- The selection criteria for the consumer, if any, match that of messages arriving at the given destination.
- The connection has been told to start delivery of messages.

Messages delivered over the connection are distributed to the appropriate sessions in which they are queued to be consumed by the appropriate MessageConsumer objects, as shown in Figure 5-3. Messages are retrieved off each session queue one at a time (a session is single threaded). They are consumed either synchronously (by a client thread invoking the receive method) or asynchronously (by the session thread invoking the onMessage method of a MessageListener object).

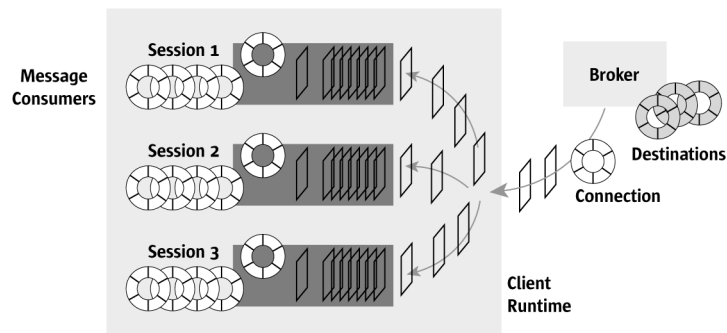


Figure 5-3: Message Delivery to Client Runtime

When a broker delivers messages to the client runtime, it marks them accordingly. However, it does not really know if they have been received or consumed. Therefore, the broker waits for the client to acknowledge receipt of a message before deleting the message from the broker's destination.

### 5.4.2.3 ConnectionFactory Administered Objects

A ConnectionFactory object is used to create physical connections between a client application and a Message Service. A ConnectionFactory object has no physical representation in a broker. Instead, it is used simply to enable the client application to establish connections with a broker. A ConnectionFactory object is also used to specify behaviors of the connection and of the client runtime that is using it to access a broker. By configuring a ConnectionFactory administered object, the administrator can specify the attribute values (that is, the properties) common to all the connections that it produces.

ConnectionFactory attributes can be grouped into a number of categories, depending on the behaviors they affect, including:

- Connection specification
- Auto-reconnect behavior
- Client identification
- Reliability and flow control
- Queue browser behavior



- Application server support
- Java Message Service technology-defined properties support

Because they impact client application design and performance, each of these categories and its corresponding attributes is discussed in some detail in the *iPlanet Message Queue for Java Developer's Guide*.

While the Messaging System administrator might be called upon to adjust the values of these attributes, it is normally an application developer who decides which attributes need adjustment to tune the performance of client applications.

#### 5.4.2.4 Destination Administered Objects

A Destination administered object represents a queue or topic physical destination in a broker to which the publicly named Destination object corresponds. By creating a Destination object, the administrator allows a client application's MessageConsumer and/or MessageProducer objects to access the corresponding physical destination.

### 5.4.3 The Message Service

The Message Service provides the core functionality of the asynchronous reliable messaging system. It is made up of the following main components:

*One or More Brokers* – A broker provides delivery services for the messaging system. Message delivery relies upon a number of supporting components that handle connection services, message routing and delivery, persistence, security, and logging. A Message Service can employ a single- or multi-broker configuration. Section 5.4.3.1, “Broker Components and Functions,” describes broker connection services, the Persistence Manager, the Security Manager, and the Logger. Section 5.4.3.6, “Multi-Broker Configurations (Clusters),” discusses multi-broker Message Services, which allow client connections to be distributed among a number of brokers.

*Physical Destinations* – Delivery of a message is a two-phase process—delivery from a producing client to a physical destination maintained by a broker, followed by delivery from the destination to one or more consuming clients. Physical destinations represent locations in a broker's physical memory and/or persistent storage. Section 5.4.3.7, “Physical Destinations,” discusses the two main types of physical destinations—queue destinations and topic destinations.

#### 5.4.3.1 Broker Components and Functions

Message delivery in the Messaging System—from producing clients to destinations, and then from destinations to one or more consuming clients—is performed either by a single broker or a cluster of brokers working in tandem. This lengthy section describes the features and functions of a single broker. Multiple broker configurations are discussed in the briefer Section 5.4.3.6.

In order to perform message delivery, a broker must set up communication channels with clients, perform authentication and authorization, route messages appropriately, guarantee reliable delivery, and provide data for monitoring system performance.

As the broker executes this complex set of functions, it uses a number of different components, each with a specific role in the delivery process. These internal components can be configured to optimize the performance of the broker, depending on load conditions, application complexity, and so on. The main broker components are illustrated in Figure 5-4 and described briefly in Table 5-1.

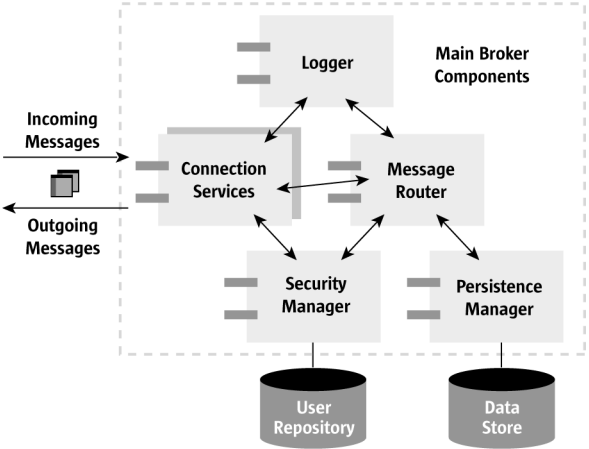


Figure 5-4: Broker Components

Component	Description/Function
Connection Services	Manage the physical connections between a broker and clients, providing transport for incoming and outgoing messages.
Message Router	Manages the routing and delivery of messages. These include JMS messages as well as control messages used by the iPlanet Message Queue for Java software messaging system to support Java Message Service technology based message delivery ("JMS Message Delivery").
Persistence Manager	Manages the writing of data to persistent storage so that system failure does not result in failure to deliver JMS messages.
Security Manager	Provides authentication services for users requesting connections to a broker and authorization services (access control) for authenticated users.
Logger	Writes monitoring and diagnostic information to log files or the console so that an administrator can monitor and manage a broker.

Section 5.4.3.2 through Section 5.4.3.5.4 more fully describe the functions performed by the different broker components and the properties that can be configured to affect their behavior.

### 5.4.3.2 Connection Services

A broker supports communication with both (JMS) clients and iPlanet Message Queue for Java software administration clients via connection services that manage the physical connections between that broker and its clients. A broker can be configured to run a wide variety of connection services. Each service has a Thread Pool Manager and registers itself with a common Port Mapper service, as shown in Figure 5-5.

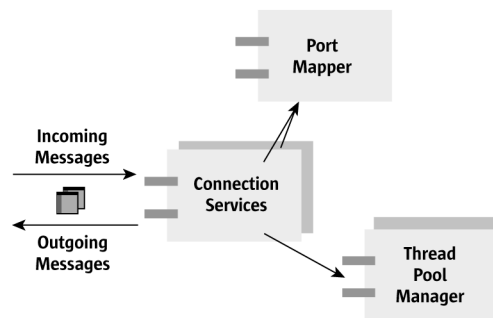


Figure 5-5: Connection Services Support

Each connection service is defined by its service type and connection type. The service type specifies whether the connection service provides JMS message delivery (NORMAL) or broker administration (ADMIN) services. The connection type indicates the underlying transport protocol layer that supports the connection service.

The port from which each connection service is available is specified by the broker's host name and a port number. That port can be statically or dynamically allocated. iPlanet Message Queue for Java software provides a Port Mapper that maps dynamically allocated ports to the different connection services. The Port Mapper itself resides at standard port number 7676.

When a client sets up a connection with the broker, it first contacts the Port Mapper to request the port number of the connection service it desires. Note that a static port number can be assigned for each service when configuring broker connection services, but this is not recommended.

Each connection service is multi-threaded, supporting multiple connections. Each connection to the broker requires two threads: one to manage incoming messages and one to manage outgoing messages. To conserve resources, a Thread Pool Manager component allocates threads to connections from a shared thread pool, as they are needed. The Thread Pool Manager can be configured to optimize connection resources. For example, it is possible to set the minimum number and maximum number of threads in a thread pool.

Note that each connection service also supports specific authentication and authorization (access control) features.

### 5.4.3.3 Message Router

Once connections have been established between clients and a broker using the supported connection services, the routing and delivery of messages can proceed via the Message Router. In addition to JMS messages, the Message Router can manage the routing and delivery of control messages used by the iPlanet Message Queue for Java software messaging system to support JMS message delivery.

#### 5.4.3.3.1 *Basic Delivery Mechanisms*

Broadly speaking, the messages handled by the Message Router fall into the following two categories:

- JMS messages sent by producer clients, destined for consumer clients-payload messages.
- Control messages that are sent to and from clients in order to support the delivery of the JMS messages.

If the incoming message is a JMS message, the broker routes it to consumer clients, based on whether it will go to a queue or topic destination:

- If the destination is a topic, the Message Router immediately routes the JMS message to all active subscribers to the topic. In the case of inactive durable subscribers, the Message Router holds the message until the subscriber becomes active, then delivers the message to that subscriber.
- If the destination is a queue, the Message Router places the JMS message in the corresponding queue, then delivers it to the appropriate consumer when the message reaches the front of the queue. The order in which messages reach the front of the queue depends on the order of their arrival and on their priority.

Once the Message Router has delivered a message to all its intended consumers, it clears it from memory. If the message is persistent, it removes it from the broker's persistent data store.

This relatively straightforward delivery mechanism becomes more complicated with the addition of reliability requirements. Mechanisms for providing reliability are discussed below.

### 5.4.3.3.2 *Reliable Delivery*

Reliable delivery has two requirements: ensuring that delivery of messages to and from a broker is successful, and ensuring that the broker does not lose messages or delivery information before the messages are actually delivered. The first requirement is filled through the mechanisms of acknowledgements and transactions. The second one is filled through the mechanism of persistence. Both requirements are supported by the efficient allocation of router system resources.

#### *Acknowledgements*

To ensure that messages are successfully delivered to and from a broker, the Messaging Service uses a number of control messages called acknowledgements. For example, when a producer sends a JMS message (a payload message as opposed to a control message) to a destination, the broker sends back a control message, which is a broker acknowledgement of the fact that it received the JMS message. (In practice, the Messaging Service does this only if the producer specifies the JMS message as persistent.) The producing client uses the broker acknowledgement to guarantee delivery to the destination.

#### *Transactions*

The client and broker acknowledgement processes described above also apply to JMS message deliveries grouped into transactions. In such cases, client and broker acknowledgements operate on the level of a transaction rather than on the level of individual JMS message sends or receives. When a transaction commits, a broker acknowledgement is sent automatically. The broker employs a transaction manager to commit transactions or roll them back should they fail.

#### *Persistence*

Persistence ensures that the broker does not lose messages or delivery information before messages are actually delivered. In general, messages remain in memory until they have been delivered or they expire. However, if the broker should fail, these messages would be lost.

#### *Efficient Allocation of Router System Resources*

The performance of a broker depends on the system resources available and how efficiently resources such as memory are utilized. For example, the Message Router includes a mechanism for locally swapping messages to disk when memory resources become scarce.

The broker's memory management functions can be configured using properties that perform the following functions:

- Governance of how often memory reclamation takes place.
- Setting of limits on the total number and total size of messages in memory.

- Control of the swapping of messages to disk. For example, thresholds for the number of messages or the total size of messages that will trigger swapping (whichever is reached first will trigger the operation) can be set. In addition, the percentage of messages remaining after swapping takes place can be set.

#### 5.4.3.4 Persistence Manager

In order for a broker to recover in case of failure, it must recreate the state of its message delivery operations. This requires it to save all persistent messages, as well as essential routing and delivery information, to a data store. A Persistence Manager component manages the writing and retrieval of this information.

To recover a failed broker requires more than simply restoring undelivered messages. The broker must also be able to:

- Re-create destinations.
- Restore the list of durable subscribers for each topic.
- Restore the acknowledge list for each message.
- Reproduce the state of all committed transactions.

The Persistence Manager manages the storage and retrieval of all of this state information. When a broker restarts, it recreates destinations and durable subscriptions, recovers persistent messages, restores the state of all transactions, and recreates its routing table for undelivered messages. It can then resume message delivery.

iPlanet Message Queue for Java software supports both built-in and plugged-in persistence modules. Built-in persistence is based on a flat file data store. Plugged-in persistence uses a Java DataBase Connectivity™ (JDBC) interface and requires a JDBC interface-compliant data store. The built-in persistence is generally faster than plugged-in persistence. However, some users prefer the redundancy and administrative features gained through the use of a JDBC interface-compliant database system.

#### 5.4.3.5 Security Manager

iPlanet Message Queue for Java software provides authentication and authorization (access control) features, as well as encryption capabilities.

The authentication and authorization features depend upon a user repository. This consists of a file, directory, or database that information about the users of the messaging system, including their names, passwords, and group memberships. The names and passwords are used to authenticate a user when a connection to a broker is requested. The user names and group memberships are used, in conjunction with an access control file, to authorize operations such as producing or consuming messages for destinations.

### 5.4.3.5.1 Authentication

iPlanet Message Queue for Java software security supports password-based authentication. When a client requests a connection to a broker, the client must submit a user name and password. The Security Manager compares the name and password submitted by the client to those stored in the user repository. On transmitting the password from client to broker, the passwords are encoded using either base64 encoding or message digest, MD5. In a separate manner, the type of encoding used by each connection service configured or encoding on a broker-wide basis can be set.

### 5.4.3.5.2 Authorization

Once the user of a client application has been authenticated, that user can be authorized to perform various Message System-related activities. The Security Manager supports both user-based and group-based access control. Depending on a user's name or the groups to which the user is assigned in the user repository, that user has permission to perform certain operations. As shown in Figure 5-6, these access controls are specified in an access control properties file.

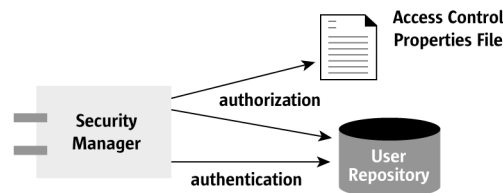


Figure 5-6: Security Manager Support

### 5.4.3.5.3 Encryption

To encrypt messages sent between clients and broker, a connection service based on the Secure Socket Layer (SSL) standard is used. SSL provides security at a connection level by establishing an encrypted connection between an SSL-enabled broker and an SSL-enabled client.

### 5.4.3.5.4 Logger

The broker includes a number of components for monitoring and diagnosing its operation. Among these are components that generate data (a Metrics Generator) and a Logger component that writes out data (metrics information as well as error messages and warnings) through a number of output channels. The scheme is illustrated in Figure 5-7.

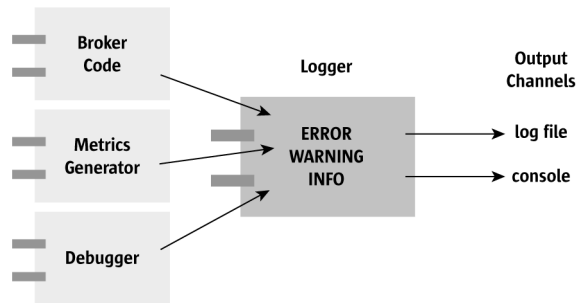


Figure 5-7: Logging Scheme

### 5.4.3.6 Multi-Broker Configurations (Clusters)

iPlanet Message Queue for Java software supports the implementation of a Message Service using multiple interconnected brokers known as a broker cluster. Cluster support provides for scalability of the Message Service.

As the number of clients connected to a broker increases, and as the number of messages being delivered grows, a broker will eventually exceed its limitations for such resources such as file descriptor and memory. One way to accommodate increasing loads is to add more brokers to a Message Service, thus distributing client connections and message delivery across multiple brokers.

Multiple brokers may also be used to optimize network bandwidth. For example, slower, long-distance network links can be used between a set of remote brokers, while reserving higher speed links for connecting clients to their respective brokers.

#### 5.4.3.6.1 *Multi-Broker Architecture*

A multi-broker Message Service allows client connections to be distributed among a number of brokers, as shown in Figure 5-8. From a client point of view, each client connects to an individual broker (its home broker) and sends and receives messages as if the home broker were the only broker in the cluster. However, from a Message Service point of view, the home broker is working in tandem with other brokers in the cluster to provide delivery services to the message producers and consumers to which it is directly connected.



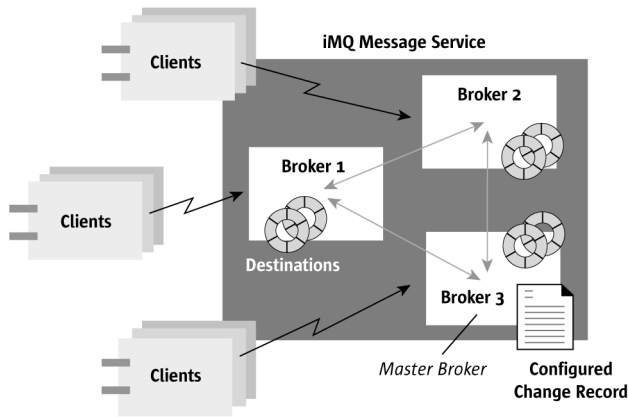


Figure 5-8: Cluster Architecture

In a multi-broker configuration, instances of each destination reside on all of the brokers in a cluster. In addition, each broker knows about message consumers that are registered with all other brokers. Each broker can therefore route messages from its own directly connected message producers to remote message consumers. It can also deliver messages from remote producers to its own directly connected consumers.

In a cluster configuration, the broker to which each message producer is directly connected performs the routing for messages sent to it by that producer. Hence, a persistent message is both stored and routed by the message's home broker.

Whenever an administrator creates or destroys a destination on a broker, this information is automatically propagated to all other brokers in a cluster. Similarly, whenever a message consumer is registered with its home broker, or whenever a consumer is disconnected from its home broker—either explicitly or because of a client or network failure, or because its home broker goes down—the relevant information about the consumer is propagated throughout the cluster. In a similar fashion, information about durable subscribers is also propagated to all brokers in a cluster.

### 5.4.3.7 Physical Destinations

iPlanet Message Queue for Java software messaging is based on a two-phase delivery of messages:

- Delivery of a message from a producer client to a destination on the broker.
- Delivery of the message from the destination on the broker to one or more consumer clients.

There are two main types of physical destinations: queues (point-to-point delivery model) and topics (publish/subscribe delivery model). These destinations represent locations in a broker's physical memory where incoming messages are marshaled before being routed to consumer clients. In addition, auto-created and temporary destinations must be taken into consideration.

The sections below describe the properties and behaviors the various types of physical destinations.

#### 5.4.3.7.1 *Queue Destinations*

Queue destinations are used in point-to-point messaging, wherein a message is meant for ultimate delivery to only one of a number of consumers that has registered an interest in the destination. As messages arrive from producer clients, they are queued and delivered to a consumer client.

The routing of queued messages depends on the queue's delivery policy. iPlanet Message Queue for Java software implements the following three queue delivery policies:

*Single* – This queue can only route messages to one message consumer. If a second message consumer attempts to register with the queue, it is rejected. If the registered message consumer disconnects, routing of messages no longer takes place, and messages are saved until a new consumer is registered.

*Failover* – This queue can route messages to more than one message consumer, but it will only do so if its primary message consumer (the first to register with the broker) disconnects. In that case, the routing goes to the next message consumer to register. It continues to be routed to that consumer until such time as that consumer fails, and so on. If no message consumer is registered, messages are saved until a consumer registers.

*Round-Robin* – This queue can route messages to more than one message consumer. Assuming that a number of consumers are registered for a queue, the first message into that queue will be routed to the first message consumer to have registered, the second message to the second consumer to have registered, and so on. Additional messages are routed to the same set of consumers in the same order. If a number of messages are queued up before consumers register for a queue, the messages are routed in batches to avoid flooding any one consumer. If any message consumer disconnects, the messages routed to that consumer are redistributed among the remaining active consumers. Due to such redistributions, there is no guarantee that the order of delivery of messages to consumers is the same as the order in which they are received in the queue.

#### 5.4.3.7.2 *Topic Destinations*

Topic destinations are used in publish/subscribe messaging, wherein a message is meant for ultimate delivery to all of the consumers that have registered an interest in the destination. As messages arrive from producers, they are routed to all consumers subscribed to the topic. If consumers have registered a durable subscription to the topic, they do not have to be active at the time the message is delivered to the topic. Instead the broker will store the message until the consumer is once again active, then deliver the message.

#### 5.4.3.7.3 *Auto-Created (vs. Admin-Created) Destinations*

There may be situations in which it is desirable to create destinations dynamically. For example, during a development and test cycle, it may be desirable for the broker to automatically create destinations as they are needed, without requiring the intervention of an administrator.

iPlanet Message Queue for Java software supports this auto-create capability. When auto-creation is enabled, a broker automatically creates a destination whenever a `MessageConsumer` or `MessageProducer` attempts to access a nonexistent destination. However, destinations that are created automatically instead of explicitly, can produce clashes between different client applications that use the same destination name or degraded system performance (due to the resources required to support a destination). For this reason, an auto-created destination is automatically destroyed by the broker when it is no longer being used, that is, when it no longer has message consumer clients and no longer contains any messages. If a broker is restarted, it re-creates auto-created destinations only if they contain persistent messages.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.



## Business Process Integration

---

Chapter 4 discussed the integration of Enterprise Information Systems (EISs) applications via the Java™ 2 Platform, Enterprise Edition Connector Architecture and the Sun Open Net Environment (Sun ONE) architecture's native support for Web services. Chapter 5 considered service integration via the asynchronous reliable messaging systems such as the iPlanet™ Message Queue for Java. This chapter explains the top-level federated software system integration capability provided by the Sun ONE architecture.

The generally accepted industry definition of integration can be paraphrased as the controlled sharing of data and business processes among any connected applications and data sources within an enterprise and between trading partners. To make economic sense for today's IT organizations, this must be possible without having to make sweeping changes to the corporations existing information assets, including applications and data structures.

Such flexible system integration of diverse clients and services requires a comprehensive infrastructure, especially when those clients and services are defined and limited by others outside the control of the local enterprise. Typical real-life examples are off-the-shelf packages and vendor-supplied external services.

Because the needs of application systems are often as diverse as the source, age, and technology of their implementations (as with programming languages such as, C++, CICS, or Java), system integration product architectures often include a plug-in approach in the data translation and messaging layers. This plug-in approach allows the basic framework to be extended in the field to encompass a wide variety of situations.

For example, the system integration infrastructure must be capable of integrating services and applications "within the firewall," as well as those that are resident anywhere on the Web. The iPlanet Integration Server product facilitates the integration of services and applications in both types of locations. It is described in Section 6.4, "iPlanet™ Integration Server."

## 6.1 The Integration Challenge

Increasingly over the recent years, IT departments have focused less on building new applications. Instead they have devoted more effort to connecting existing applications. Mergers and acquisitions have increased, while corporations have found that they cannot continue to leverage their accumulated IT assets.

New technologies and product releases often leave these business assets on orphaned, legacy infrastructure islands. Over the next decade, integration will continue to be the dominant challenge facing IT and corporate software development experts.

The following three example settings illustrate how integration plays a significant role in all development projects, no matter how small:

- Business document exchange
- Connecting internal applications
- Establishing new partnerships and businesses

### 6.1.1 Setting 1: Business Document Exchange

A corporate objective is to connect separate systems within an enterprise. For example, an enterprise might need to make a Customer Relationship Management (CRM) package and a financial application package such as Oracle Financials communicate with each other.

The development strategy here might be to devote IT resources toward reviewing each of the application interface libraries for these packages, developing a connection strategy that is unique to the two systems, and finally implementing the project. This strategy works well until another, separate system needs to interact with one or more of the two integrated systems. As new systems are added to the mix, the process can become unmanageable.

### 6.1.2 Setting 2: Connecting Internal Applications

In this setting, a corporation needs to establish connectivity between internal applications within its own corporate environment. These applications have key corporate business functionalities implemented in relatively isolated technology islands. These functionalities may include, for example:

- Typical transaction-style systems, perhaps running in large corporate data centers.
- Systems built on specialized business-oriented technologies from various vendors such as SAP, PeopleSoft, and Siebel.
- Home-grown applications that provide unique business value such as financial portfolio analysis, or insurance risk or rating applications.

The corporate objective here is to tie these applications together to provide a more uniform or expanded capability to a larger organizational or business function. Often the tasks are driven by the need to simplify or regularize repetitive functions, such as updating customer contact information, or using a single application to perform a common business function.

As in the Business Document Exchange setting, often these tasks are handled as unique projects. Each project is concerned only with connecting the systems and applications currently of interest to the project. Because the average enterprise has at least 50 unique and disparate business applications, before long these types of projects become so intertwined and complex that it is almost impossible to effectively manage them or predict how changes at one end might impact another.

### 6.1.3 Setting 3: Establishing New Partnerships and Businesses

As business becomes more global, the quantity of available partners has dramatically increased in recent years. It is now possible to interact with consumers and suppliers across the globe as if they were around the corner. In addition to the usual business relationship concerns, global associations are often dramatically affected by rapidly changing and often unforeseen events, such as public policy, monetary fluctuations, and available labor. As the available marketplace of consumers and suppliers grows, so do the possibilities for creating partnerships. From an electronic exchange perspective, effectively managing and supporting global associations and partnerships is an enormous challenge to any company.

At the time of this writing, only one reliable and effective integration strategy can address this type of a dynamic, international marketplace of collaboration. That strategy is embodied by the Electronic Business for eXtensible Markup Language (XML) Interchange collection of specifications, referred to as ebXML. For more information, see Section 6.3, “ebXML.”

The Sun ONE architecture embraces both the Web service native specifications and the business extensions offered by ebXML. In many cases, these standards can coexist, but certain compromises may not be acceptable for a reliable business.

## 6.2 Existing Styles of Integration

Integration-based products can be categorized into a number of useful types. The following three are of primary interest to the discussions in this chapter:

- Integration between businesses, often called Business to Business, or B2B.
- Integration between applications within a business, often called Application to Application or A2A. When A2A is coupled with a process-oriented sequencing engine, it is often referred to as Enterprise Application Integration (EAI).

- E-Commerce Integration, an emerging new style of integration whereby businesses establish and change collaborations in a highly dynamic and rapidly changing environment.

These three styles of integration are discussed in the sections that follow.

## 6.2.1 B2B Integration

B2B integration provides some methodology that allows businesses to share data between their respective information systems. This data-sharing generally consists of the exchange of documents between business partners. Typically accomplished using pre-defined documents or forms, this type of integration is referred to as Electronic Document Interchange (EDI).

While there are many specification standards, the most prominent is the X.12 standard, which is defined by the Accredited Standards Committee or <http://www.x12.org>. This committee defines a collection of documents that can be used as exchange media between partners. Many categories of documents are standardized by this organization and their international partner, which is part of the UN/EDIFACT (United Nations rules for Electronic Data Interchange for Administration, Commerce, and Transport). These organizations define the document to be exchanged—including the format, fields and their contextual relationship—with extensions and updates are released annually.

Widely varying messaging protocol formats have grown up as technologies have advanced. Support for private Value Added Networks (VANs), simple File Transfer Protocol (FTP), Multipart Internet Mail Extension (MIME), and other formats are commonly used. Originally the exchange documents were pure text. As technologies have evolved, however, they have evolved to use XML.

B2B integration, or document exchange, tends to be relatively static—businesses know the identity of their partners beforehand. Based on these partnerships, mutual interest drives both sides to invest in the information infrastructure to support this exchange.

How each company chooses to implement its business behind the exchange end-points is entirely up to that company. The standards say nothing about this. Further, with the exception of efforts such as RosettaNet PIPs, the standards generally do not attempt to describe how or in what context the exchanges could or might fit into the overall application.

However, the dividing lines are clear. Each organization has its inputs and outputs. Each is free to implement and host these exchanges in any fashion they desire. So long as the overall partnership remains in good business health, so do these exchanges.



## 6.2.2 EAI Integration

Within an enterprise, applications have been developed to provide various service functions for the organization. These may include functions such as customer tracking or account management systems, or they may be more inwardly focused employee management systems. Even in a conservative organization, applications that were once considered completely isolated from the rest of the business eventually interact with other systems and information technology assets within it. Adding mergers and acquisitions into this mix simply increases the certainty that there will be overlap, redundancy, and a need for interconnections between information systems.

Traditionally, systems connections have been built on a per-project basis. Often, the first effort combines all the applications that reference customers using the common element (or abstraction) of the customer, along with a customer-tracking system. These projects start with a team of IT staff that research the combination of the application interfaces. This effort might then be followed by a project that connects abstractions like Orders, Claims, or Trades.

As enterprises are forced to grapple with the ever-increasing need to respond to business changes, they must come to terms with this legacy of information assets and the need to maintain these connections in a fashion that allows for dynamic, agile system upgrades. Maintaining agile and dynamic connections between corporate assets is critical to an organization's ability to rapidly respond to business climate changes. To effectively support such requirements, systems must be loosely connected. Furthermore, the process specifications must have the capacity to support migration from today's processes to tomorrow's as-yet unknown needs.

The personnel that are most uniquely positioned to understand how the enterprise accomplishes its business are most often domain experts, sometimes called business analysts. They have expertise in how the enterprise accomplishes its goals. Typically valued for their ability to understand what is important and to provide key business value to the particular enterprise, these business analysts often have little or no software development background. Nonetheless, they have expertise in core aspects of the business such as Medical Management, Portfolio Risk Analysis, or Claims Administration. They work with concepts that are even more abstract than the collaboration exchange documents of B2B integration systems. These abstractions may not even have a real-world analog.

A graphical process-specification tool is essential to effective management of information assets in conjunction with business abstractions. Such a tool can specify how the system is to manage, coordinate, and drive to conclusion all the information and personnel aspects of a system. Graphical process specifications allow domain experts to route work to automated services such as credit check and print check, as well as to send messages to human users based on events such as documents received and pending deadlines.

EAI is the overall term that encompasses this effort to draw together, manage, and control the overall enterprise application as a collection of independent yet loosely connected system components and services.

## 6.2.3 e-Commerce Integration

When B2B and EAI integration are combined, then confronted by the need to integrate with partners rapidly, reliably, and securely, the result is Electronic Commerce (e-Commerce) integration. In this environment, partners rapidly move in and out of business collaborations via the dynamic relationships that are commonplace across the Internet.

The e-Commerce flavor of integration relies on utilization and ubiquity of common standards that allow for reliable interaction and discovery between partners. These standards must provide complete business functions with authentication, signatures, nonrepudiation, and business context. Furthermore, they must be ubiquitous, reliable, and well-understood across the entire class of potential partners.

When working with standards for the Web, open forum standards committees with corporate sponsorship are the norm for developing this part of the interconnection requirements. These processes can take months or possibly years to complete. It is important to choose standards that perform to the requirements of the enterprise in order to accomplish reliable, predictable, and safe interactions can be accomplished. While many proposed specifications for XML-based interactions are in development, the Sun ONE architecture and emerging ebXML standard are the only collection of e-Commerce standards that are ready for business use at the time of this writing.

## 6.3 ebXML

Initiated in 1999, the Electronic Business eXtensible Markup Language (ebXML) collection of specifications was initiated by a consortium of businesses, vendors, and governments. The 1.0 set of completed standards was released in 2001, and many of them are well into their second revision.

ebXML is sponsored by UN/CEFACT and OASIS. UN/CEFACT is the acronym for United Nations Centre for Trade Facilitation and Electronic Business, headquartered in Geneva. OASIS is the Organization for the Advancement of Structured Information Standards, a nonprofit organization dedicated to the creation of international interoperability specifications based open, public standards.

As a collection of standards, ebXML builds on top of the same collection of requirements and prerequisites that have driven other Web-services standards, such as Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). However ebXML picks up where these standards leave off, providing many features that are still only in the formative stages as alternative specifications.

ebXML comprises a collection of key system components, each of which can be implemented or adopted independently or in conjunction with the other components. This allows for easy adoption of the technology in a gradual, controlled fashion.

## 6.3.1 ebXML Objectives and Architecture

The core objectives of ebXML are to build a collection of standards that lower the expense associated with reliable electronic interchange for small- and medium-sized business. Implementation of these standards will allow organizations that may not have dedicated development staff (and may have only a single desktop platform) to participate in these electronic business exchanges. Another goal of ebXML is to allow these exchanges to be performed across the open Internet and provide an alternative to using EDI VANs.

Figure 6–1 illustrates the overall architecture of ebXML.

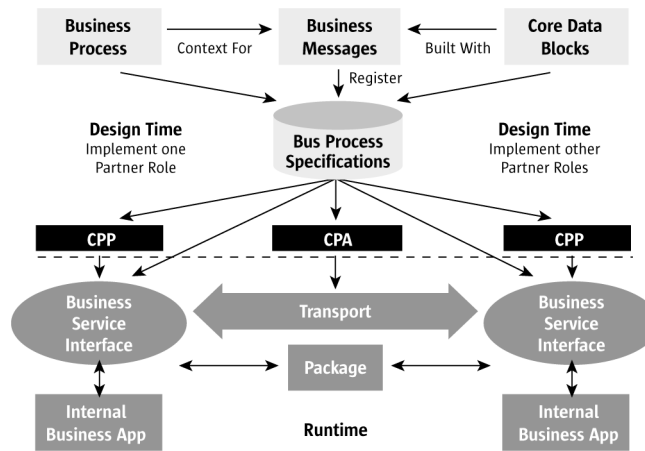


Figure 6–1: ebXML System Architecture

Each of these blocks is discussed in the following sections.

## 6.3.2 ebXML Messaging

As discussed in Chapter 5, “Asynchronous Reliable Messaging,” messaging is the heart of any business exchange. Exchanging messages can take many forms. These include sending application binary, text, or XML payloads within EDI, MIME, or SOAP envelopes sent via e-mail, FTP, or HTTP, with or without encryption.

Reliable messaging requires that message deliveries are acknowledged and delivered once-and-only-once per message. Reliable messages may need to be processed in the order they were sent. They also depend on security mechanisms to prevent inadvertent or intentional modification, to validate the source and sender, and to provide an audit trail of what was sent and received prevent a partner from repudiating a transaction.

Secure messaging requires that the message contents cannot be revealed to unintended recipients. Secure messaging includes encrypting application data as well as message routing metadata.

The ebXML messaging specification provides the framework for reliable messaging, allowing for a wide range of service levels, authentication, and record-keeping options. Key elements of the ebXML specification include:

- **Delivery guarantees** – These guarantees range from best-effort delivery to guaranteed once-and-only-once delivery. They include support for missing acknowledgement timeouts, retransmissions, retry intervals, maximum retry attempts, and duplicate elimination as necessary.
- **Ordered message delivery** – This messaging element guarantees that messages with sequence identifiers will be delivered to the partner application in the proper sequence.
- **Security profiles** – These profiles range from none to various combinations of digital signing and encryption.
- **Content agnostic specifications** – This messaging infrastructure can be used by collaborators for exchanging any data they like, from EDI X.12 documents to XML documents to the more mundane business documents, often binary, such as drawings, spreadsheets, or scanned images”
- **Multipart messages** – These types of messages allow multiple documents to be “clipped” together into a single message package.

## 6.3.3 ebXML Collaboration Elements

In order to effectively perform electronic commerce, it is critical to know the capabilities of each collaborator, how to exchange messages, and, most importantly, when the specified exchanges are appropriate. Once these capabilities are reviewed and the collaboration begins, a record of the agreement used during the collaboration is required. This agreement contains the subset of the intersection of two capability profiles and defines exactly how the exchanges are going to take place.

### 6.3.3.1 Collaboration Protocol Profiles

The first collaboration element is the Collaboration Protocol Profile (CPP). This document, as described in the ebXML specification, defines the capabilities of a specific partner. It can simply describe what forms of messaging can be utilized, or it can contain a complete, detailed view of an entire collaboration offering.

For example, a simple CPP document might specify:

- An end-point URL
- The types of messaging capabilities that business wants to use (for example, encrypted Multipart SOAP messages with trusted signatures)

A more complex CPP might also specify:

- Once-and-only-once delivery
- Sequence ordering
- The process specification
- The business document schemas to be exchanged

These profiles can list alternatives supported. For example, a business might accept messages over a specific HTTP port, or via FTP or via SMTP.

### 6.3.3.2 Collaboration Protocol Agreements

When businesses come together to engage in business, they form agreements. These agreements can comprise technical and mechanical details, as well as describe business issues, response time-frames, problem remediation, and other matters.

With ebXML, the business-technical aspects of these agreements are described in the Collaboration Protocol Agreement (CPA). This document refines the profile capabilities found in the CPP document for each business. It also establishes an agreement that provides resolution to any optional or mismatched capability provisions. For example, if Partner A's profile lists HTTP and FTP as allowable messaging protocols, but partner B's profile identifies only HTTP, then the CPA would identify only HTTP as the messaging transport.

Beyond messaging details, CPAs can contain response timeframes. An example response timeframe is: Whenever document A is received, the recipient must send a message acknowledgment within 30 seconds, or the document will be sent again. After five re-tries, the message will be considered undeliverable and the exchange will be aborted.

CPAs can also specify the business documents that will be used for exchange. Such business documents can even specify complete multi-exchange process descriptions.

### 6.3.3.3 Document Exchange Processes

Document exchange processes are an important facet of e-Commerce system design. This is a key differentiator between ebXML and other currently approved standards available for general Web-services development. The process specification allows collaborators to provide contextual organization for collaboration sequences. These sequences can be as simple as describing a single document exchange as outlined above. Alternatively, they can be as complex as describing a complete process such as bid, acceptance, purchase order, funding, type of auction, and means of exchange.

### 6.3.3.4 Business Process Schema Specification

Business process definitions and descriptions are captured in the ebXML Business Process Specification Schema (BPSS). While this XML schema document can describe arbitrarily complex processes, in practice the BPSS often merely describes simple exchange procedures such as how to send and acknowledge a document, and what to do if certain error conditions occur.

### 6.3.3.5 Registry Repository

It is a given that any company will participate in any number of CPAs. These collaborations may be with one or more organizations and may describe one or more related or unique business opportunities. Users have the need to store and categorize these CPAs in a logical and extensible manner that allows their applications to reference these documents when messages arrive. For example, when a message containing document X arrives, the message references a CPA ID. The system can consult the corresponding CPA to determine how to process the document, how to respond, how to ascertain if the message has been properly sequenced, and how to acknowledge receipt.

ebXML provides a complete specification for a repository that can store and retrieve these documents and artifacts. The documents may contain the public, advertised portions of the system or the private portions of the system that describe, for example, how a particular company adds its own unique business value.

Another core value that ebXML adds to the e-Commerce palette of system development is a framework that allows collaborations to be established with the same type of dynamic behavior characteristic of the Internet (with respect to Business to Consumer or B2C styles of interactions). ebXML provides specifications for the repository that allow public access to the artifacts that the corporation wants to make public. Unlike UDDI, these specifications provide facilities for determining who has access to what.

The goal is to develop a framework that provides dynamic advertisement and collaboration so that businesses can enter into partnerships with little or no coding or IT-centric overhead. This would allow each business system to read, interpret, and respond to documents with a much higher level of automation than ever possible.

## 6.3.4 ebXML Core Components Project

The final objective of the ebXML initiative is to solve the vertical impedance mismatch between documents and terminology of one business domain to another. As an example, purchase orders for the medical industry are not readily transferable to the automotive industry. The Core Components project seeks to simplify the crosstalk between different business vertical industries. The Core Components group has focused on definitions of schema elements and hierarchies, finding definitions for concepts common across different industries.

Another approach to solving this problem is being promoted under the guise of Unified Business Language (UBL). At the time of this writing, UBL is being run as an OASIS Technical Committee effort. UBL is an extension to the XML Common Business Library (xCBL). More information on UBL can be found at <http://www.oasis-open.org> and <http://www.xcbl.org>.

### 6.3.5 ebXML Functional Overview

As shown in Figure 6-2, systems developers construct specialized business logic, interface elements, and interconnections between existing and new information technology assets. Business analysts and contracts administrators, on the other hand, specify the business profiles (CPPs), agreements, Business Documents (BD), Workflow and Collaboration definitions (BPSS), Core Components, and other elements. These are all stored in a repository that allows them to be referenced at runtime, reused for new collaborations, and referenced for historical and statistical work. All the actors can work with development parallelism to achieve optimum system development performance.

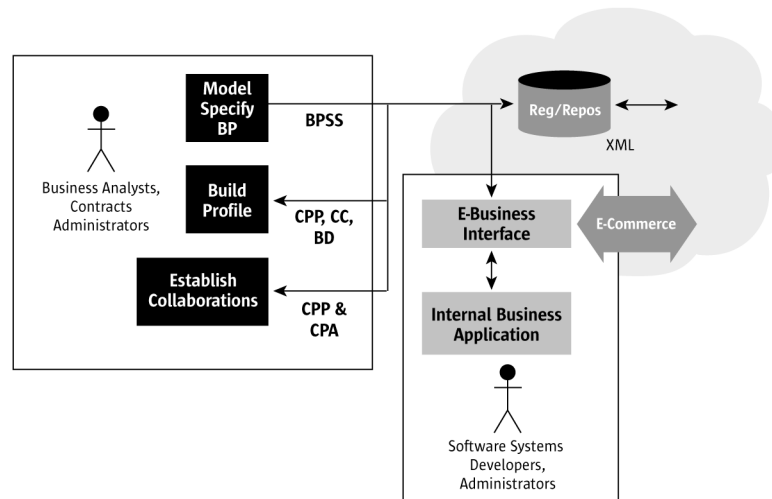


Figure 6-2: ebXML Functional View

Global-sized organizations might implement their own business logic, applications and components by developing Java and other applications. On the other hand, a small business may simply use the ebXML messaging for document exchange, creating and manipulating documents with off-the-shelf forms and page editors.

### 6.3.5.1 Reliable Electronic Business Exchange

A reliable electronic system for exchanging messages is essential to an enterprise's process of engaging in business with partners. Reliability is absolutely critical. For any system to become widely accepted by an enterprise, it must reliably solve the needs of that enterprise and do so at a cost that is attractive, especially when compared to current systems for doing business.

For enterprises, reliability includes not only the standard features associated with highly scalable systems, but also extends to the ability to reliably track, reconstruct, and drive business exchanges. Message tracking, guaranteed-once delivery, message sequencing, and nonrepudiation must all be available to assure collaborators that their processes will work smoothly. Collaborators also demand assurance that they will be able to reliably defend their actions if needed, as well as to reliably find and track down problems easily.

While content is critical to the ultimate success of any business system, the designers of ebXML had the foresight to focus on the internal infrastructure and architecture, regardless of the document payloads that are to be exchanged.

#### 6.3.5.1.1 *Advantages over Fax Messaging*

ebXML messaging is, by its definition, multi-point aware. Transmission, Reliability, and Packaging (TRP) specifications provide a complete suite of specifications that define the spectrum of available options that must be supported for any compliant messaging service. While server-side services require high-volume, multi-staged routing capabilities, smaller scale businesses are more concerned that their messages are secure and can be reliably exchanged with their partners using a simple, straightforward methodology.

The most common messaging solution in use by business today is the FAX machine. Because most business have at least one desktop computer, ebXML messaging can replace this functionality at similar or lower cost. The messaging infrastructure can add levels of tracking, authentication, nonrepudiation, and secrecy that are simply not possible via FAX documents. When XML documents are exchanged, the added benefits of an easily machine-readable exchange medium make this a combination that has clear compelling value to business of any size, so long as the price of entry is sufficiently low.

#### 6.3.5.1.2 *Differentiation from Web Browser Messaging*

It is important to differentiate the solutions discussed in this chapter from today's most common, Internet-based messaging solution—the Web Browser form. Easy and straightforward to implement, for many user-style consumers (and even corporate users), the Web Browser form offers a sufficient level of quality. However, these exchanges have low reliability and add little, if any, of the additional value that ebXML messaging specifications provide. Basing an automated exchange on the reliability level of browser-based forms is simply not a reasonable starting point.



### 6.3.5.1.3 *Importance of Quality of Service*

Guaranteed once-delivery of messages is a critical feature of ebXML TRP. Additional features provide for scaling back this option to levels such as best effort (if it fails, try it again) and at least once (if duplicates are received, the receiver can eliminate them).

When people drive the end-point applications, the elimination of duplicates, the confirmation of messages, and message missing (NAK) procedures are relatively easy for people to sort out—but problems do occur. For example, customers submit orders from HTML forms and receive duplicate shipments or nothing and critical documents get lost in FAX hoppers.

Businesses have become accustomed to these types of problems and have developed management processes and procedures in the face of these issues. However, building automation systems that must react to such low service quality are difficult to design and hard to maintain. The ebXML messaging system can reliably deliver messages at nearly any level of service quality necessary for efficient, automated operation. Furthermore, they can do this with complete security.

### 6.3.5.2 Authentication and Audit

Users must be certain that the message has actually been received from the party that is expected to have sent it. Messages need to have secure authentication schemes for the message packages, the payload documents need to be authenticated, and message exchange must be accomplished using secure encryption.

Furthermore, audit tracking must be an integral part of the messaging system. Tracking must extend to each of the collaboration partners as well. Message exchanges must be monitored and tracked even after the container package has been received. Each payload section may have a unique destination within the ebXML application. Furthermore, the messaging system must be able to provide a complete record of what was received, how it was processed, and what the credentials of the message actually were.

This is critical to the support of nonrepudiation. However, nonrepudiation is not as simple as archiving each and every step of the message-processing chain. Some businesses operate in environments in which documents can be rescinded. For example, a customer may sign a contract for a loan, then decide that he or she does not want the loan. (In certain jurisdictions, rescinding a loan is legal within a cooling-off period.) Such requirements must be allowed in order to support effective commerce. In other words, the systems need to support the business, rather than making the business alter its practices to conform to the demands of the systems.

## 6.3.6 Profiles and Agreements in Practice

It is unlikely that corporations will quickly accept the notion of rapid dynamic collaboration environments. However, if there is sufficient value, it is quite possible that users will rapidly adopt this strategy.

In any case, the results of profile lookup must be reliable, consistent, and well understood. At present, a competing technology, Web Services Description Language (WSDL), is also designed to provide capability descriptions. However, at the time of this writing, there is insufficient specification and common agreement about how exactly to define a service and, perhaps more interestingly, no common mechanism for reliably searching and finding a service. Much of the selection criteria has been reserved and may be withheld from the specification so that the repository service providers can choose how to respond to under-specified queries.

With ebXML, there is a clear specification of how to describe the service, as well as of how to retrieve the service definition. Messaging specifications are clearly defined; exchange procedures can be described and used in automated systems; and business documents can be written in formats that can be readily interpreted electronically.

### 6.3.7 Exchange Processes

Once the appropriate set of collaboration profiles are assembled, the stage is set for the actual business exchange. With the ebXML specifications included in this architecture, it is always possible to determine when and how it is appropriate to perform this interaction in a reliable and automated fashion. Generally, the exchange processes can be of a simple pattern: Send document, await acknowledgment, finish. However as outlined above, these transaction processes can be arbitrarily complex.

With SOAP and UDDI alone, the interactions must be implemented independently by the separate applications. If there are discrepancies between these implementations, error signals and system administrators must be used to sort it out. WSDL can help, but it does not identify the roles nor the documents to be exchanged in the collaboration.

Table 6-1 summarizes the level of capabilities associated with each of the methodologies and standards that have been discussed above.

Table 6-1:

Function or Quality of Service	Human Agent	Paper Contract	FAX	Web Service	ebXML
Machine Readable	N/A	N	N	Y	Y
Audit Trail	N	Y	Y	?	Y
Authenticated	Y	N	N	?	Y
Choreography	Y	N	N	N	Y
Nonrepudiation	N	N	N	N	Y
Reliable	N	N	N	?	Y

Function or Quality of Service	Human Agent	Paper Contract	FAX	Web Service	ebXML
Secure	N	N	N	?	Y
Tamperproof	N	N	N	?	Y
Validation	Y	N	N	?	Y

Given time, Web services specifications will undoubtedly provide equivalent features and functions to ebXML. However, ebXML is ready for business use today.

## 6.4 iPlanet™ Integration Server

The iPlanet Integration Server is designed and built to incorporate many of the features described by the ebXML specifications discussed in previous sections. It is described here as an example of an ebXML application that enterprises can use for both collaboration and reliable messaging in the dynamic, international marketplace.

Note that the iPlanet Integration Server is an optional component of the Sun ONE architecture. The facilities needed by the clients and servers can be provided by any software that meets the messaging, data, and delivery requirements of the enterprise. From an external perspective, this can be as simple as XML and HTTP Web-service-only integration, using, for example, integration based on Simple Object Access Protocol (SOAP), Universal Description, Discovery, and Integration (UDDI), and Web Services Description Language (WSDL). However, this could be a completely arbitrary collection of collaborations for e-commerce, internal applications with specialized API requirements, or business logic available from a suite of Java components. The architecture described in this book presents this integration structure and illustrates how to select the proper building blocks for runtime and tools to meet the requirements of any type of integration architecture.

The externally visible architecture of the iPlanet Integration Server appears in Figure 6-3.

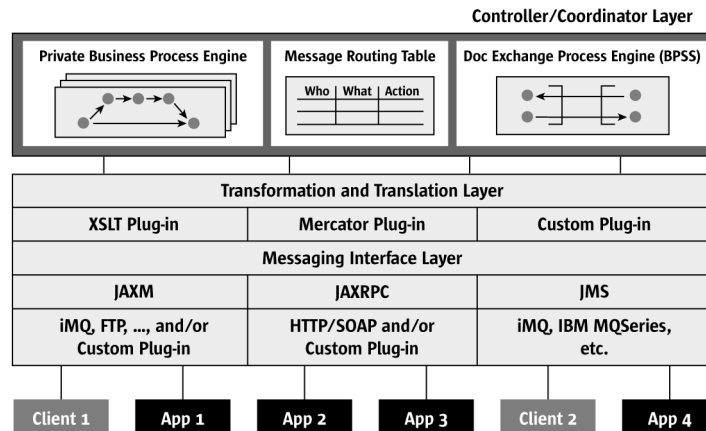


Figure 6-3: iPlanet Integration Server Architecture

As Figure 6-3 shows, the main components of the iPlanet Integration Server architecture are as follows:

- Controller/Coordination Layer
- Transformation and Translation Layer
- Messaging Interface Layer
- Clients and servers

These components are discussed in the following sections.

### 6.4.1 Controller/Coordination Layer

This layer includes three separate controllers that can be deployed and used separately or concurrently:

- Private Business Process Engine
- Message Routing Table
- Document Exchange Process Engine, operating in accordance with Business Process Specification Schema (BPSS) Business Signals

### 6.4.1.1 Private Business Process Engine

A business process is a sequence of operations and the associated data. The classic example is order processing. In this case, the operations may include steps such as:

- Credit check
- Order validation
- Inventory check
- Back order hold
- Shipping
- Customer notification

The above sequence of operations includes decisions such as “if a part is not in inventory, generate a back-order request.” It also may also include timers, which can be either periodic, such as “wait until 8 AM to notify shipping” or elapsed, such as “if the part is on back-order for more than two days, send an e-mail to the customer.” While most operations are automatic and carried out by services, some may be manual. “If the amount is large, send to a manager for further verification” is an example of a manual operation.

The data associated with a business process typically includes:

- Order details, which consist of a description of the order, including line items and the name of the customer.
- Data that changes as the order is processed; for example, the order status and the name of the order processing clerk.

Each business process is defined using a graphical tool that allows a business analyst to specify the complete map or sequence of operations, as well as the data that is needed to complete every operation. That tool also allows the analyst to make decisions about which path in the map is to be performed. A map may be named, for example, “Shoe Order,” and stored in a repository that is available to the engine.

Either a client or a service that specifies both the name of the business process and its associated data can initiate a business process. The engine uses the name to find the map for the process and saves a copy of the data in a database. It then begins to interpret the process definition, starting at the defined entry point. Each operation is invoked in its turn. As this occurs, decisions are made that select various paths within the process. The process terminates when the exit point is encountered.

The role of the Private Business Process Engine is to act as the central “traffic cop” of the application system. It invokes services and interacts with clients in a controlled manner, as specified by the business analyst. Each of these services and clients simply responds to the process engine; it does not communicate directly with any other service or client. This enables effective encapsulation of the services and clients, thus providing maximum reuse of these software components.

### 6.4.1.2 Message Routing Table

The Message Routing Table is simpler than the Private Business Process Engine in several ways:

- It employs the “fire and forget” paradigm, which means that the Message Routing Table does not maintain any state associated with the operation or sequence of operation.
- The service or application invoked does not return a result. Therefore, the process map that is interpreted by the Private Business Process Engine is absent. Instead, the business analyst creates a simple table of operations. Each entry in this table contains:
  - An optional identification of one or more message senders
  - A message content pattern
  - A message destination, which is almost always a service

When a message is received, the integration server consults the Message Routing Table. For each row that matches the identity of the sender, the message content is compared to the corresponding content pattern. If a match is found, the message is sent to the specified message destination.

### 6.4.1.3 Document Exchange Process Engine

The Document Exchange Process Engine manages document flow between the integration server and external sources according to rules specified using ebXML. Most often, these rules are used to control information passed among trading partners. They can also be used for information flow between departments or entities in a single business. These rules are constructed by a business analyst or imported from definitions supplied by industry trade groups.

Exchange processes, or choreography, are published as BPSS and are available for use by the partner as well. Typically these are associated with specific CPAs, but choreography can be used independently if needed.

## 6.4.2 Data Transformation and Translation Layer

If necessary, messages are passed through the Message Transformation layer. This layer is responsible for converting data from one dialect to another. A unity transform requires will bypass this layer. Simple transformations may consist of stripping out unnecessary data for the next action, or simple reorganization of the fields into an alternate schema. Complex transformation may require splitting elements apart or performing lookup operations to convert from one series to another.

This layer is typically used to transform data to and from the end-point applications into the common, regular format that is used within the enterprise. This canonical transformation is useful in the following scenarios:

- Messages arriving from clients and services
- Messages sent to clients and services

The Data Transformation and Translation Layer solves the problem of matching the data format and content requirements of the service or application to the available data format and content. Consider this example: In the order-processing scenario described above, the data provided by the initiator of the order includes order details comprised of the customer identity, the list of shoes being ordered, and the total value of the order. This is encoded in XML. The credit check service expects only the customer's credit number and the amount of the purchase. In such a situation, the Data Transformation and Translation Layer would be used to perform three operations:

- Remove the extraneous data and pass only the customer's credit number and total purchase amount.
- Change the field names as needed in the likely situation that the order details XML document contained different field names compared to the credit check service.
- Convert the information from the XML document maintained by the process engine into the EDI format needed by the credit check service.

In addition, the response from the credit check service would be translated and then merged with data held in the Private Business Process Engine:

- Convert the arriving Electronic Document Interchange (EDI) format to XML.
- Translate the simple numeric response (1 means yes and 0 means no) into Good or Bad.
- Move the translated value into the CreditStatus field of the XML document held by the Private Business Process Engine.

Because most messages can be translated using XML Stylesheet Language Transformations (XSLT), this is the primary data translation and transformation facility. In cases where more complex translations are needed, the architecture provides for a general-purpose plug-in capability.

### 6.4.3 Messaging Interface Layer

Just as with the Data Transformation and Translation Layer, all received and transmitted messages pass through the Messaging Interface Layer. Three message interface types are supported:

- Java API for XML Messaging (“JAXM”)
- Extended JAXM for ebXML

- Java API for XML-based RPC (“JAX-RPC”)
- Java Message Service

A variety of message handling plug-ins are provided for each, including the capability for customers to provide additional custom plug-ins.

For outbound messages, the layer uses a destination descriptor attached to the message by the Controller/Coordination Layer to select a message interface type and plug-in. Inbound messages are tagged by the Message Interface Layer to identify the sender. In both cases, the information in the tag varies according to the particular plug-in. For example, a message arriving via Java Message Service would include the name of queue it was retrieved from and other associated information.

## 6.5 Future Directions for Messaging

Universal Business Language (UBL) is an emerging standard for providing a more canonical framework for the document payloads for electronic commerce. At the time of this writing, the architecture neither includes nor excludes use of UBL documents. For more information, refer to the discussion of “The Next Step for Global e-Commerce” white paper at <http://www.oasis-open.org>.

## 6.6 Process Integration Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for process integration components.

Interface Name	Level	Status	Reference	Comments
EbXML Messaging Specification, 2.0	Application	Footnote 2	<a href="http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf">http://www.oasis-open.org/committees/ebxml-msg/documents/ebMS_v2_0.pdf</a>	Approved in January 2002
CPP and CPA Specifications	Application	Footnote 2	<a href="http://www.oasis-open.org/committees/ebxml-cppa/documents/working_drafts/index.shtml">http://www.oasis-open.org/committees/ebxml-cppa/documents/working_drafts/index.shtml</a>	Index of draft Specifications and Schemas
Collaboration Exchange Processes	Application	Footnote 2	<a href="http://www.ebxml.org">http://www.ebxml.org</a>	EbXML Business Process Schema Specification (BPSS) is currently at version 1.01 and was approved in May 2001.
Enterprise Process Exchange	Application	Footnote 2	UMM - Published by UN/CEFACT, July 2001 at <a href="http://www.unece.org">http://www.unece.org</a> and WfXML - Published by the Workflow Management Collation (WfMC). Published January 2002 at <a href="http://www.wfmc.org">http://www.wfmc.org</a>	EbXML Business Process Schema Specification (BPSS) is currently at version 1.01 and was approved in May 2001.



Interface Name	Level	Status	Reference	Comments
Registry - Repository	Application	Footnote 2	<a href="http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf">http://www.oasis-open.org/committees/regrep/documents/2.0/specs/ebrs.pdf</a>	2.0 specification was approved in Dec. 2001.
XML Digital Signatures	Application	Footnote 1	<a href="http://www.w3.org/TR/xmlsig-core/">http://www.w3.org/TR/xmlsig-core/</a>	See Chapter 11
Java API for XML Registries™ 1.0 (“JAXR”) (JSR 93)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/93.jsp">http://www.jcp.org/jsr/detail/93.jsp</a>	See Chapter 3. Supporting Specifications Provides an API for a set of distributed Registry Services that enables business-to-business integration between business enterprises, using the protocols being defined by ebXML.org, Oasis, ISO 1117
Java API for XML-based RPC (JAX-RPC) (JSR 101)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/101.jsp">http://www.jcp.org/jsr/detail/101.jsp</a>	See Chapter 3

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

Footnote 4: This is a published proprietary interface, and support of this interface is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 5: This is an unpublished proprietary interface. A published definition for this interface will be provided in a future version of the Sun ONE architecture.

Footnote 6: This interface is not yet defined. A definition for this interface will be provided in a future version of the Sun ONE architecture.

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.



## Part 4. Service Creation, Assembly, and Deployment

---



## Development Tools

---

The Sun™ Open Net Environment (Sun ONE) platform development tools enable the creation, assembly and deployment of applications and services designed to maximize the facilities available on the Sun ONE platform. Such services and applications are designed to be available on a wide range of client devices—from desktop PCs to PDAs to mobile phones. They are meant to be provided by any number of back-end server platforms—from databases to application servers to high-performance server environments.

To conform to version 1.0 of the Sun ONE architecture, a development environment or tool must either:

- Provide support for:
  - The creation, assembly, and deployment of Sun ONE architecture compliant applications and services.
  - The use of compliant servers, applications, and services for the creation, assembly, and deployment of conformant applications and services.
- Be deliverable as a plug-in module for NetBeans™ software or the Forte™ IDE, conforming to the NetBeans software Open API specification.

A development tool suite can choose either of these approaches to deliver the capabilities needed to create, assemble, deploy, debug, and maintain applications enabled by the Sun ONE platform (“Sun One applications”) and Services on Demand.

The fully capable NetBeans software and Forte IDE are strong platforms that can be used to deliver useful developer tools, enabling vendors to concentrate on their core competencies rather than on becoming full-blown IDE vendors. The underlying NetBeans software Open APIs provide a rich, open set of interfaces that allow vendors to easily deliver tools that target the Sun ONE architecture and integratable stack. Alternatively, vendors that have pre-existing toolsets can opt to include their own solutions for Sun ONE platform development, providing tools that can deliver the support for the rich set of APIs and server technologies required for Sun ONE applications and services.

## 7.1 Sun™ Open Net Environment (Sun ONE) Platform Tool Suite Requirements

A development tool suite conforming to the Sun ONE architecture specification is required to support the creation, assembly, and deployment of Sun ONE platform compliant services and applications. In principle, this implies a large number of standards supported by the Sun ONE architecture, although some are evolving and thus not required initially. In general, a tool suite needs to provide support for the primary components of the Sun ONE platform, namely:

- Service Delivery (See Chapter 8, “Presentation Frameworks,” Chapter 9, “The Portal Server,” and Chapter 10, “The Java Web Client Model.”)
- Service Containers (See Chapter 3, “J2EE™ Components and Containers.”)
- Service Integration (See Chapter 6, “Business Process Integration.”)
- Identity and Policy (See Chapter 11, “Identity and Policy Services.”)
- Platform support (See Chapter 12, “Platform Services.”)

For details on the required supported interfaces, see the relevant chapters for the above components.

While modern tool suites generally may provide a number of the required features for Sun ONE platform development, service providers and product vendors frequently do not have such tool suites available for deployment with their applications or services. As such, the Sun ONE architecture provides a rich, extensible tools platform that developers can use to create, assemble, and deploy Sun ONE applications and services. In addition, vendors who want to integrate into the platform can use the same underlying tools platform available in NetBeans software and the Forte product line as a foundation for modular tools that can be plugged into that architecture. As a result, tools can augment the rich functionality already available in the Forte tools suite for Java™ 2 Platform, Enterprise Edition (J2EE™ platform)/Web Services development with their own customized solutions for specific development tasks needed for integrated products.

### 7.1.1 NetBeans™ Software IDE

The NetBeans software integrated development environment (IDE) is open source, modular, standards-based, and integrated. Because it is written in the Java programming language, it can run on any platform with a Java™ Virtual Machine (JVM™) that is compliant with the J2EE platform. Based on NetBeans software, the Forte IDE provides the additional tools necessary to create, assemble, and deploy Sun ONE applications and Services on Demand. Tools that will be used to create applications and services can be based on either NetBeans software or the Forte IDE. While both provide the same set of APIs, the Forte IDE provides a richer set of functionality. This is because it already ships with a full set of tools designed to support the Sun ONE architecture and products developed for the Sun One platform.

Because NetBeans software and the Forte IDE are modular, developers can:

- Add modules that provide editing, debugging, syntax coloring, error highlighting, and additional functions for the Java programming language as well as other languages. The IDE works with C, C++, Unified Modeling Language (UML), Interface Definition Language (IDL), eXtensible Markup Language (XML), and others, as well as with Java programming language.
- Switch any of the IDE modules on or off. By switching off unneeded modules, the IDE consumes less memory and no longer offers unnecessary information and actions.
- Write modules that add new features or replace functionality in the IDE.
- Update the IDE online through the Update Center.

The new, standards-based Metadata Repository component of the NetBeans software architecture makes it easier to build modules that support other programming languages, in addition to enhancing performance and features that are related to re-factoring. By supporting standardized models for metadata in a language-neutral way, the Metadata Repository also makes it easier to integrate third-party products, such as UML tools.

## 7.1.2 NetBeans Software Core and APIs

The NetBeans software IDE is based on a thin core that is responsible for basic services and infrastructure, such as windowing, actions, and file management. This core implements the NetBeans Software Open APIs, to which the modules are written.

This core can be extended with plug-in modules written to the NetBeans software Open APIs. In fact, the basic functions of the IDE—including editor, Java programming language support, compilation execution, and debugging services—are implemented in modules written to these APIs. Virtually every aspect of the NetBeans software IDE is extensible. At the center of the architecture, the APIs cleanly abstract the IDE’s functionality, creating a modular, dynamic environment. More than 400 classes are shipped in 15 API sets, along with use instructions and Javadoc™ software documentation. The NetBeans software Open APIs are illustrated in Figure 7–1.

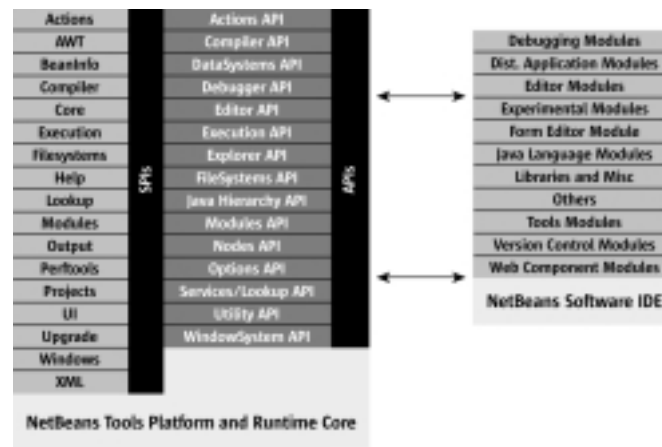


Figure 7–1: NetBeans software API Implementation

The NetBeans software APIs all share the following characteristics:

- They are standalone—none of the classes contained therein refer to any other parts of the IDE. There is no reference to code outside the APIs, except to code contained in the standard Java runtime environment and a couple of basic libraries—such as XML parsing and regular expressions.
- Most APIs have what are essentially inner and outer components. This refers to the concept that APIs can be used to either create new functionality (called the service provider interface [SPI]), or use the functionality accessible through the client API.
- The APIs create a dynamic environment—that is, new functionality can be added or removed simply by adding or removing modules without recompiling or even restarting the IDE on any platform. The module author builds a module, then the user installs and uses it.
- Because the APIs are provided in separate Java Archive (JAR) files, they can be easily used as standalone libraries. For example, writing a Java application using the FileSystems library does not affect that application’s independence. Instead, its functionality is enhanced, client code is simplified, and development time is reduced.



This chapter focuses on the NetBeans software Open APIs rather than the specific capabilities within the IDE or the NetBeans software runtime other than for general descriptions. For detailed information regarding the specific implementation issues surrounding the NetBeans software runtime or the IDE, refer to <http://www.netbeans.org>.

All of the NetBeans software Open APIs are designed for use in implementing modules. Modules permit dynamic extension of the IDE. By abstracting functionality into a well-defined API, module authors and users alike are assured that new functionality can be added quickly and easily.

Modules may range in complexity from a single Java class library, which may do something as elementary as adding a menu item, to new actions or analyses that can be performed on Java source files. Likewise, a module can contain a full-scale integration of a major external application, such as a Java profiling suite. All modules are distributed and installed as JAR files, with special entries in the JAR manifest that are recognized by the IDE.

For example, the FileSystems API is a storage-neutral abstraction of a hierarchical file system. It is the only part of the IDE that knows about the physical storage of data. Therefore, other modules in the IDE can use it to access files in a store such as a JAR archive, an FTP site, or a database-based repository. The physical storage area remains transparent to the modules requesting access.

Figure 7–2 shows the NetBeans software FileSystems API in relation to other modules. The model for other NetBeans software APIs is similar to this one.

Note that modules can publish additional APIs. Module authors may choose to build a public interface to the modules that they develop, so others can add to or extend the functionality of their products.

The primary APIs in NetBeans software Open APIs are described in Table 7–1, which follows the figure.

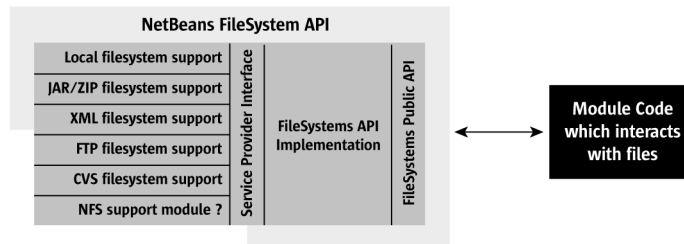


Figure 7–2: NetBeans software FileSystem API

## 7.2 Other Key Modules for Tool Developers

A number of other NetBeans software modules and APIs aid developers by delivering additional value-add functionality on top of NetBeans software or the Forte IDE. While not a part of the core API set, they are valuable tools that can be used to facilitate the creation of new modules, extending existing tools or modules, or providing custom solutions. Examples include the API Support module, the Form Editor, and the Metadata Repository, which are discussed below.

### 7.2.1 API Support Module

The API Support module can help the developer work with the API classes, including local documentation, templates, and testing tools. It is designed for people using the NetBeans software IDE to write other NetBeans software modules using the Open APIs. Because of the dynamic nature of Java technology and of the IDE in particular, it is possible to speed development time of new modules, as well as the testing and maintenance of existing ones. Features include:

- **Test module** – This allows the developer to install a module in test mode. After changing some code, he or she can simply hit *Execute* again to recompile, rebuild the JAR, and reinstall the module. The changes can be seen without restarting the IDE.
- **Templates** – Thorough templates are included for many common types of objects created as part of a module. These help demonstrate the APIs and handle the boilerplate code.

- **Testers** – Various special executors permit the developer to test individual classes (or objects) inside the IDE without bundling them into a module.
- **Documentation and code completion** – These include full local API documentation with convenient searching options, as well as code completion for API classes, to make editing quicker.
- **XML layer support** – This type of support allows the developer to interactively edit “files” in an XML layer as if they were files on disk. It also provides tools to help fix folder ordering, give files special icons, and perform other useful functions.
- **Bean Browser** – This handy tool allow for the inspection of many details of the running IDE from the Explorer. It be used for such functions as debugging, exploring, and testing. It shows the structure of the IDE presented as JavaBeans™ component architecture, with special annotations for items of interest to the API developer.

For more details, refer to <http://apisupport.netbeans.org/>.

## 7.2.2 Form Editor

The form module lets developers visually create forms with AWT, Swing, and JavaBeans component support. Often referred to as the Form Editor, this module includes the following features:

- Support for both visual and nonvisual forms.
- Fully WYSIWYG designer with the “Test Form” feature.
- Extensible Component Palette with pre-installed Swing and Abstract Window Toolkit (AWT) components.
- The Component Inspector that shows a component’s tree and properties.
- Automatic, fully customizable, one-way code generation.
- Support for all AWT/Swing layout managers with full drag and drop.
- A powerful GridBagLayout visual customizer.
- Support for null layout.
- In-place editing of text labels of components such as labels, buttons, and text fields directly in the designer area.
- Full JavaBeans component support, including the installation, use, and customization of beans; the support of properties, property editors, custom property editors, events and event handlers, bean customizers, and other tools.
- Visual beans customization that allows the creation of forms from any JavaBean component classes.
- The ability to connection beans using the Connection Wizard.

By using the Form Editor in combination with other facilities within the environment (such as the API Support module), tool integrators can rapidly create and integrate custom solutions for Sun ONE platform application and service development.

For more information, see <http://form.netbeans.org/>.

## 7.2.3 The Metadata Repository

The Metadata Repository (MDR), which has recently been added to the NetBeans software source code, provides support for modules that must create, store, and retrieve metadata. Metadata refers to data that describes the structure and characteristics of program elements or data; for example, the structure and method signatures of the Java programming language's class file. Another example of metadata is a model of the Java programming language's class structures that determines such characteristics as the set of members that any element can contain and specifies the fact that classes can contain inner classes and methods.

The MDR contains an implementation of the Meta Object Facility (MOF), an abstract language for describing metamodels defined by the Object Management Group (OMG). The set of constructs used by the MOF for metamodeling is based on the Unified Modeling Language (UML) model.

The MDR offers the following benefits:

- Support for developers who implement modules that generate or use metadata; for example, modules that support a programming or modeling language.
- Interoperability with tools that support the open standards on which the MDR is based (e.g., MOF, XMI, and OCL). For example, many UML modelers already support XMI.
- Generation of all the interfaces representing an API for support of a given language model within the IDE. In other words, MDR provides a language-neutral standard that saves developers the effort of writing their own APIs for each language.
- Similarity in appearance of all generated APIs. The developer can become familiar with the API by simply examining the metamodel.

The MOF standard can be used to integrate various types of tools for functions such as modeling, code generation, code analysis, and dependency management. By using the reflective package contained in the MOF, developers can write generic, language-neutral tools for working with metadata, such as class browsers, search tools, and others specialized tools.

For more information, see <http://mdr.netbeans.org/>.

## 7.2.4 Other Tool-Related NetBeans Software Modules

In addition to the Open APIs and the key modules listed above, NetBeans software includes a large number of modules which may be useful for tool integrators and developers. While not complete, the following table provides an example of the types of functionality that are available, usable, and extensible by integrated modules. For the complete list, see <http://www.netbeans.org/modules.html>.

Table 7–1: Other Tool-Related NetBeans Software Modules\*

Module	Location
CORBA	CORBA development support. See <a href="http://corba.netbeans.org/">http://corba.netbeans.org/</a>
Database Explorer	Database explorer. See <a href="http://db.netbeans.org/">http://db.netbeans.org/</a>
JINI™ Architecture Support	Jini architecture support module. See <a href="http://jini.netbeans.org/">http://jini.netbeans.org/</a>
Java Naming and Directory Interface™ (“J.N.D.I.”) Support	J.N.D.I. API support module. See <a href="http://jndi.netbeans.org/">http://jndi.netbeans.org/</a>
RMI Support	Remote Method Invocation support. See <a href="http://rmi.netbeans.org/">http://rmi.netbeans.org/</a>
Projects	Projects development framework. See <a href="http://projects.netbeans.org/">http://projects.netbeans.org/</a>
Autoupdate	IDE auto-update feature. See <a href="http://autoupdate.netbeans.org/">http://autoupdate.netbeans.org/</a>
J2EE Platform Server	J2EE Platform Deployment API integration. See <a href="http://j2eeserver.netbeans.org/">http://j2eeserver.netbeans.org/</a>
Tomcat Integration	Tomcat Server integration. See <a href="http://tomcatint.netbeans.org/">http://tomcatint.netbeans.org/</a>
XML Services	Generic XML tools and infrastructure. See <a href="http://xml.netbeans.org/">http://xml.netbeans.org/</a>

\*This is a third party site: Sun has no responsibility, and makes no representation or warranties, regarding information on this third party site.

## 7.2.5 NetBeans Software Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for IDEs.

Interface Name	Level	Status	Reference *	Comments
NetBeans software Open APIs ver 3.3.1	Application	Footnote 1	<a href="http://openide.netbeans.org/">http://openide.netbeans.org/</a>	
NetBeans software Specification ver. 3.3	System	Footnote 1	<a href="http://java.netbeans.org/docs.html">http://java.netbeans.org/docs.html</a>	

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

## 7.3 Forte™ IDE

The Forte Tools integrated development environment (IDE) is used to create, assemble, and deploy Services on Demand for the Sun ONE platform. Based on the NetBeans software IDE and runtime, the Forte IDE shares the same set of basic features as well as the extensibility available as part of the NetBeans software Open API. As a result, developers creating tools that target products to be integrated into the Sun ONE platform will likely find that basing these tools on the Forte IDE will provide an easy solution for meeting the requirements for Sun ONE platform compliance.

Forte Tools is an integrated development environment that opens the era of Web services to Java developers. The tool suite delivers the modules, wizards, templates, and generators that enable development and deployment of robust J2EE platform-based applications and standards-based Web services in a team-oriented environment. Furthermore, the tools provide support for teams of developers building database-aware applications and JavaServer Page™ (JSP™) technology-based applications. Its fully modular environment also delivers integrated graphic user interface (GUI) design, editing, compilation, and debugging capabilities for cross-platform development of Java technology applications and applets.

### 7.3.1 Primary Components

The primary components of the Forte IDE provide direct support for a wide variety of standards and products that are defined by the Sun ONE architecture. These include:

- **J2EE platform 1.3 Support**
  - Support for the EJB 2.0 specification (including Message Driven Beans and Container Managed Persistence)
  - Java Servlet
  - JavaServer Pages™(JSP™)
  - J2EE Connector Architecture support
- **XML Support in the Java programming language**
  - Java API for XML Processing (JAXP)
  - Java API for XML-based RPC (JAX-RPC)
  - Java API for XML Messaging (JAXM)

- Java API for XML Registries (JAXR)
- Java Architecture for XML Binding (JAXB)
- **Web Services Support**
  - Tools that enable the use of Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI) for applications and services.
  - The Java Web Service Designer that enables the creation of standards-based Web Services from Java Enterprise and Web tier components.
  - In conjunction with the Web Service Designer and the Native Connector Tool, mechanisms that provide the ability to generate Web services from native (C/C++) Solaris™ Operating Environment libraries.

- **Legacy Integration**
  - Java DataBase Connectivity™ (JDBC™).
  - J2EE Connector Architecture integration for Enterprise Information Systems (EIS) and Enterprise Resource Planning (ERP) systems.
  - Native connector architecture (binding of native C/C++ applications to the Java runtime environment and Web services).
  - iPlanet™ Integration Server, which includes Business Process integration via the Process Design tool.
- **Interoperability**
  - Common Object Request Broker Architecture (CORBA) integration.
  - Integration using J2EE platform or Native Connectors.
  - Integration using Java Message Service technology, specifically through the iPlanet™ Messaging Server.
- **Client support**
  - Mobile device application development using Java 2 Platform, Micro Edition (J2ME™ platform).
  - Rich client development using GUI elements which are part of the Java 2 Platform, Standard Edition (J2SE™ platform).
  - Rich client development using the NetBeans Software Applications Framework (the NetBeans software core libraries).
  - Highly dynamic clients using the iPlanet Applications Framework and JavaServer Faces™ specification.
  - Applications that take advantage of portal services, such as iPlanet Portal Server.
  - Clients based on the standard JavaServer Pages™ (JSP™) software tag library Java Specification Request (JSR) 52.
- **Server Integration**
  - Broadened application server support, including seamless integration with iPlanet Application Server for deployment.
  - Implementation of JSR-88: Server Integration APIs.
  - Open-source server integration, such as the J2EE platform reference implementation and Apache Tomcat.
  - Expose application server administration facilities (JSR-77).
  - Support for additional iPlanet servers, including the portal and integration servers, and others iPlanet software.



- **Solaris Operating Environment Support**

- Forte IDE Developer Compilers for C, C++, and Fortran.
- High-performance computing support for the latest Sun hardware.
- Advanced native development tools including analysis tools and debugging tools that allow for the debugging of mixed native and Java programming language applications in the Solaris Operating Environment.

## 7.3.2 Partner and Third-Party Tools

As part of a rich external-partner support program, Sun and its partners provide plug-in modules for published APIs and products, including all public Java platform APIs, as well as products and APIs from iPlanet. Third-party vendors can provide plug-in modules for their products as part of their distributable products. Such plug-ins are dynamic and modular, allowing for “on-the-fly” upgrades and/or fixes as well as configurations that can allow system administrators and project managers to control module versions, features, or entitlement. Forte tools assembles certified collections of plug-ins as part of IDE configurations or products.

## 7.3.3 Services Development in the Forte IDE

Forte Tools provide the fundamental capabilities to create, assemble, and deploy Sun ONE applications and services that target the Sun ONE architecture and stack. To fulfill these requirements, Forte software provides:

- Services-centric functionality
  - Assembling solutions from services
  - Building services from components
  - Building modern client access to services
- Integrated architecture capabilities
- An extensible architecture design
- Services creation
- Services assembly
- Services deployment

### 7.3.3.1 Services-Centric Functionality

Forte Tools provide services-centric functionality to perform the following tasks:

### **Assembling Solutions from Services**

Since solving a business problem or creating a new application has come to be done as a collection of loosely coupled services, the Forte Tools enable services by:

- Providing integration facilities to translate existing applications such as C, C++, Fortran, and ERP packages into XML services.
- Enabling these XML packages to be sequenced together after they have been translated or wrapped. This sequencing is accomplished either through the use of a business process or workflow metaphor or through simple service-to-service sequencing.
- Providing XSLT rules to do data transformation, since not all XML schemas are the same.

### **Building Services from Components**

Because Services on Demand are created from a collection of Enterprise JavaBeans™ (EJB™) or other components, Forte software provides:

- Facilities to search and share components between teams across the Internet, build new components, and graphically assemble components to form services.
- Facilities to automatically generate the component bindings and XML interfaces such as UDDI and WSDL.

### **Building Modern Client Access to Services**

In the traditional computing environment, each device type uses its own tools with a device-specific markup language to create the appropriate user interface. To fill the growing needs of the Services on Demand computing world, more and more tools will draw upon the Services Grid for core business functionality. Forte's device-specific tools enable their users (usually graphic artists rather than XML service creators) to browse Services on Demand and easily attach to XML data streams via XML tags.

## 7.3.3.2 Integrated Architecture Capabilities

An integrated architecture means that a tool can be knowledgeable about the specifics of the underlying architecture, thus shielding the developer from the need to understand how and where to deploy the different components. In this way, the tasks of the developer are greatly simplified, and a rapid iterative development style can be deployed.

Forte Tools leverage the top-level functions of the Sun ONE platform—Service Creation, Assembly, and Deployment. For example, the tools can generate out components that bind into the application server and Web server. These components can be defined as services using XML and SOAP, then sequenced together using the Process Manager. In addition, they can utilize the directory and Policy Manager for such functions as checking user credentials and performing security processing.

### 7.3.3.3 Extensible Architecture Design

The foundation for all Sun tools is an open tools platform designed to support C, C++, Fortran, and the Java programming language, among others. Sun's open tools platform, in turn, is built upon the NetBeans Software Open Source Code project. This architecture ensures partners and developers that the framework will be a stable environment, thereby facilitating the development of plug-ins by the open-source community, Sun, and Sun's partners.

### 7.3.3.4 Service Creation

As shown in Figure 7-3, the Forte IDE provides a comprehensive set of service creation tools. These tools are so flexible and intuitive that developers can use them to build new intelligent services. Likewise, business professionals can rapidly assemble them for personalized user experience and have them deployed to the range of different network-connected products.

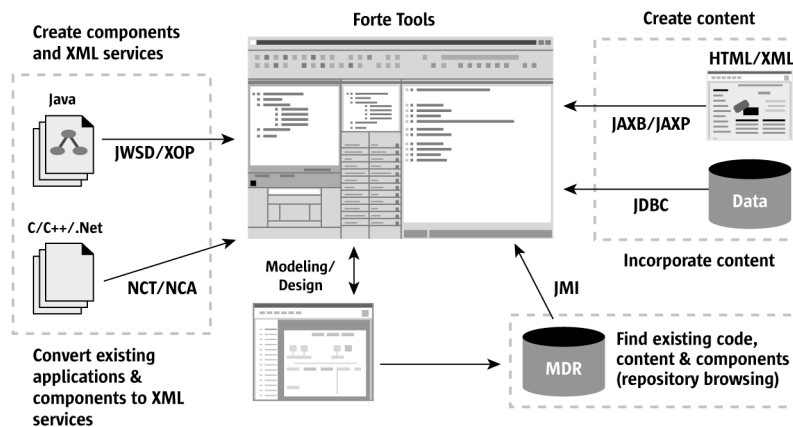


Figure 7-3: Service Creation

Service creation can be implemented through some of the following strategies:

- Through the use of the Java Web Service Designer (JWS), existing Java class libraries, Beans and Enterprise JavaBeans are exposed as SOAP-based Web services via the creation of XML Operations (XOP).
- Through the use of the Native Connector Tool (NCT) and the Native Connector Architecture (NCA), existing legacy/native libraries written in C, C++, Fortran, or even Microsoft CLR-compliant languages are exposed as either Java platform-based classes/beans or as SOAP-based Web-services.
- Through the use of the Metadata Repository (MDR), existing code and services can be incorporated into new applications and services.

- Using the XML Designer tool, developers can build adapters for existing application components.
- Content and content delivery logic can be incorporated using:
  - The JAXB or JAXP APIs, which can be used to create the content-management logic necessary for the service or service-based application.
  - JDO (Java Data Objects), which can be used to create the data-management logic necessary for services or service-based applications.
  - Modeling and design tools, which can be integrated into the development environment, thus facilitating service and application creation.

### 7.3.3.5 Service Assembly

In addition to assembling complete applications, it is important to be able to integrate services created on the Microsoft .NET and other platforms. For the service-driven network, these services will be dynamically discovered via UDDI and initially contacted via SOAP/XML. Actual invocation will be determined by the service properties.

The ability to integrate legacy applications into the Service Grid is another key function. This can be viewed as “dis-integrating” monolithic applications into dynamic services. For example, the Telco industry is rethinking its monolithic order-entry systems and creating services that allow the procurement of Telco services via XML over the Internet.

As shown in Figure 7–4, Process Definition tools assemble and determine the activities needed for coordinated service-based applications.

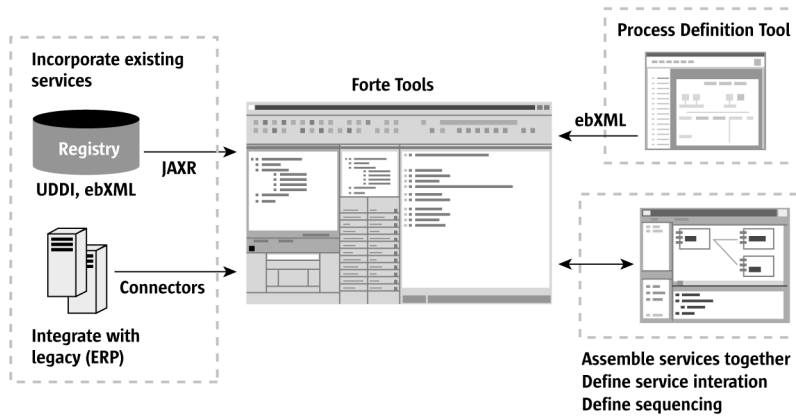


Figure 7–4: Service Assembly

### 7.3.3.6 Service Deployment

As shown in Figure 7-5, services are deployed to Sun ONE platform servers, primarily those that adhere to the J2EE specification. Service-based applications use SOAP/WSDL and UDDI for discovery, but soon they will also be comprised of application and business logic encapsulated as EJB technology, Message Driven Beans, JavaServer Pages technology, and Java Servlet APIs.

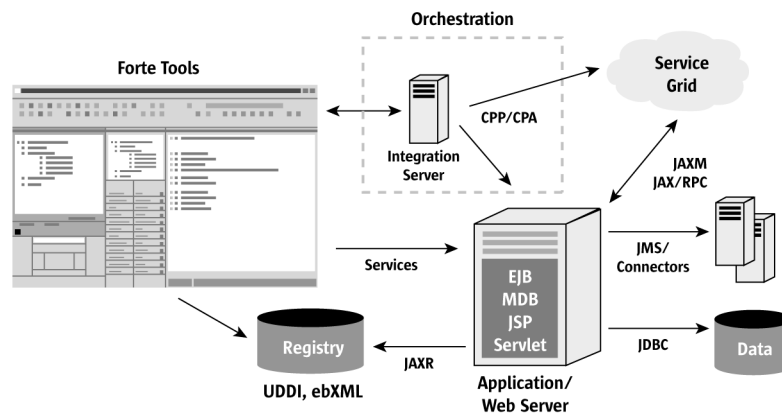


Figure 7-5: Service Deployment

Deployed applications will integrate with other services within the Service Grid, connect to legacy/ERP systems via the J2EE Connector Architecture API, and integrate with database servers and services via APIs such as JDBC. In addition, orchestration tools will orchestrate Web services within an enterprise.

### 7.3.4 Forte IDE Interfaces

The Forte IDE supports all of the interfaces described in Section 7.1.1, “NetBeans™ Software IDE” and Section 7.2.4, “Other Tool-Related NetBeans Software Modules.” In addition, the Forte IDE provides the extended functionality described in Section 7.2.5, “NetBeans Software Interfaces.” Interfaces for this extended functionality are not yet required parts of the Sun ONE architecture. However, interface definitions for some or all of this extended functionality may be provided in a future version of the Sun ONE architecture.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

## Part 5. Service Delivery

---





## Presentation Frameworks

---

Presentation frameworks gather information from both end users and the business layer of an application. They then generate the user interface and process the user's interaction with it.

The iPlanet™ Application Framework is the iPlanet implementation of a presentation framework. It will be described following a general discussion of presentation frameworks, including their adherence to the Model-View-Controller (MVC) design model.

### 8.1 Overview of Presentation Frameworks

Java™ 2 Platform, Enterprise Edition (J2EE™ platform) and its presentation layer technologies (JavaServer Pages™ (JSP™) and the Java Servlet™ API) provide a portable and standards-based foundation for building Web applications. These technologies address the issues involved in using a language to generate markup that was originally intended for publishing static documents. Pages created with the JavaServer Pages technology (“JSP pages”), and servlets, do not, however, address the before and after issues of client-type detection, form validation, localization, and other well-known presentation layer tasks. Instead, these tasks are addressed by presentation frameworks. In doing so, they present a consistent model and set of interfaces for providing a consistent application development environment.

Presentation frameworks are responsible for gathering information from end users and the business layer of an application. They also generate the user interface (UI) from that information, and process the user interaction with the delivered user interface.

Swing is a standard presentation framework that makes it easier for developers to work with the presentation layer in a traditional desktop-bound application environment. Currently there is no standard presentation framework for presentation-layer development in a Web application environment.

The J2EE platform, while providing the core technologies for building Web applications and their presentation layers, is not a presentation framework in itself. Presentation frameworks based on J2EE technology have appeared to address developer needs for building small- and medium-scale Web applications. These frameworks, which utilize the MVC (Model-View-Controller) design model discussed in the Section 8.2, vary in their implementation and capabilities. While standards-based, they do not provide a standardized approach to presentation layer development for Web applications.

Java Specification Request (JSR) 127 will provide the functionality on top of the J2EE architecture and its core technologies on which these MVC-based presentation frameworks may standardize.

## 8.2 The Model-View-Controller

As mentioned in Section 8.1, presentation frameworks based on J2EE technology typically adopt modified versions of MVC called MVC1 and MVC2, which are described in the 0.91 version of the JSP specification and *J2EE Blueprints* respectively. To view the “J2EE Blueprints” document, go to the Web page referenced in this chapter’s section of the Bibliography.

### 8.2.1 The MVC Design Model

In terms of presentation frameworks, the idea behind the Model-View-Controller design model (MVC) is that an application consists of three things: a Model, some Views of the Model, and some Controllers. The Model is the part of the application that contains the actual application logic. The Model does the database access, computes numbers, and manipulates data structures. The View and Controller represent the user interface of the application. The user interface is conceptually split into input and output components. The Controller is an input component that supplies information to the Model. The View is an output component which displays information from the Model. The View typically communicates with the Model by registering itself as a callback and responding to events generated by the Model. The iPlanet Presentation Framework’s implementation of the MVC design model is considered in Section 8.7.1 of this chapter. For a different approach, see the applied discussion of the MVC design in Chapter 10, “The Java Web Client Model”.

The architecture of the MVC triad enforces the independence of the Model from the external representation of its information. The factoring-out of code from content enables flexibility of design and reuse of code. This is the primary reason for its adoption by desktop presentation layer frameworks such as Swing.

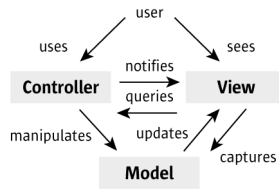


FIGURE 8-1: Classic MVC

## 8.2.2 MVC1

MVC1 was a first generation approach that used JSP pages and the JavaBeans component architecture to implement the MVC architecture for the Web. As shown in Figure 8-2, HTTP requests are sent to a JSP page that implements Controller logic and calls out to the “Model” for data to update the “View.” This approach combines Controller and View functionality within a JSP page and therefore breaks the MVC paradigm. MVC1 is appropriate for simple development and prototyping. It is not, however, recommended for serious development.

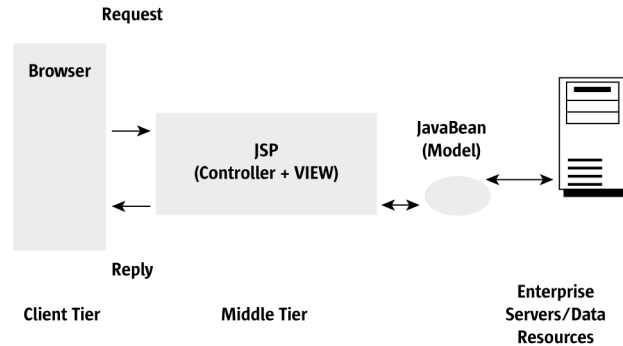


FIGURE 8-2: MVC 1

## 8.2.3 MVC2

MVC2 is a term invented by Sun to describe an MVC architecture for Web-based applications in which HTTP requests are passed from the client to a “Controller” servlet which updates the “Model” and then invokes the appropriate “View” renderer—for example, JSP technology, which in turn renders the View from the updated Model. Model 2 is well documented in “J2EE Blueprints.” The hallmark of the MVC2 approach is the separation of Controller code from content. Current implementations of presentation frameworks, including Struts and Sun’s iPlanet Application Framework, adhere to the MVC2 approach, which is shown in Figure 8–3.

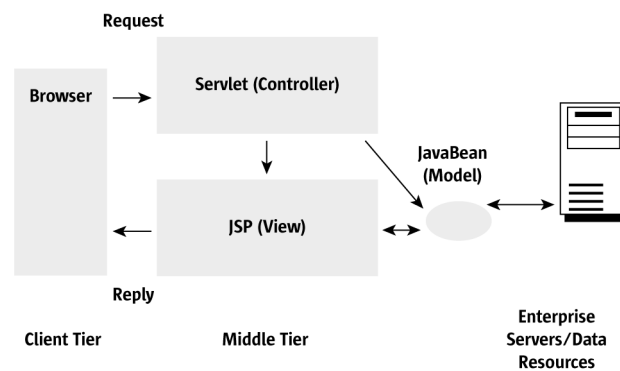


FIGURE 8–3: MVC2

## 8.3 MVC2 and the Presentation Framework

The MVC2 architecture is actually a modified MVC implementation. The major modification is that the Model no longer fires events to its Views.

The central issue is that the life cycle of the servlet (the Controller and View) is not necessarily the life cycle of the application, as it is with desktop applications. The servlet begins with a user request, typically generated by a Web browser, and ends with the response. The Model, however, may, and typically does, persist across the life of multiple servlets. Therefore, it cannot reliably notify View objects of internal state changes. This has the following consequences:

- The Model is now more “generic,” because it no longer implements the logic for registering and unregistering listeners, nor does it need to implement logic to generate events.
- The View is now responsible for capturing Model state changes.
- The Controller now notifies the View of state changes to the Model.
- The Controller must manipulate the Model before notifying the View.

These consequences, coupled with the unidirectional HTTP (client-initiated) request-response protocol, can potentially lock the developer into a coarse-grained cyclical program flow—as opposed to the fine-grained, widget-based event Model of a desktop implementation of MVC such as Swing. A major goal of Sun’s iPlanet Application Framework is to provide an MVC2 implementation that more closely approximates desktop implementations of MVC.

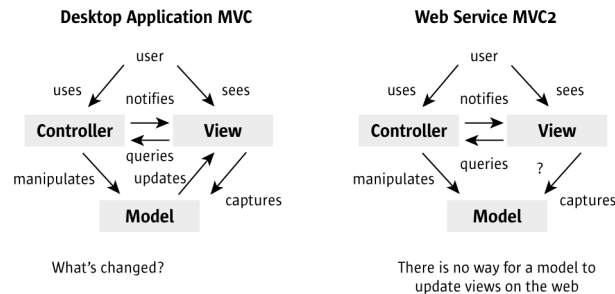


FIGURE 8–4: Desktop vs. Web-based MVC Implementations

## 8.4 Development Issues

When developing a presentation framework, it is important to consider the following issues:

**Code and content coupling** – Creating content on the server involves some degree of coupling between code and content. Depending on the technologies used, this coupling can be reduced. In some cases, it can be totally eliminated.

**Protocol limitations** – Hypertext Transfer Protocol (HTTP) is a client-initiated request-response protocol that prevents server-side code from initiating unsolicited callbacks to the client. This makes it difficult to implement realtime user interfaces such as those found in Swing.

**Coupling of events to HTTP and MVC** – Events at the granularity of an HTTP request couple the event Model to the MVC implementation. This makes it difficult for the framework to support a Swing-like hierarchical component Model.

**Component Model** – Components can be anything from a reusable piece of code to a Swing-like user interface widget.

**Flow control** – Desktop implementations of MVC focus on flow at the level of component interfaces. Hence they provide more granular approaches to application development than is typically found in MVC2 implementations that focus on flow at the application level.

## 8.5 Template and Non-Template-based MVC Architectures

The MVC model for the Web can be implemented with either a template engine or a Document Object Model (DOM)-based approach.

### 8.5.1 Template Engines

As shown in Figure 8–5, template engines take a page-based approach that allows Java technology developers and page designers to “pull” data from Java objects from within an HTML page. Templates, and more specifically taglibs, encourage but do not enforce the separation of content from code. JSP technology is the standard for generating content in this manner.

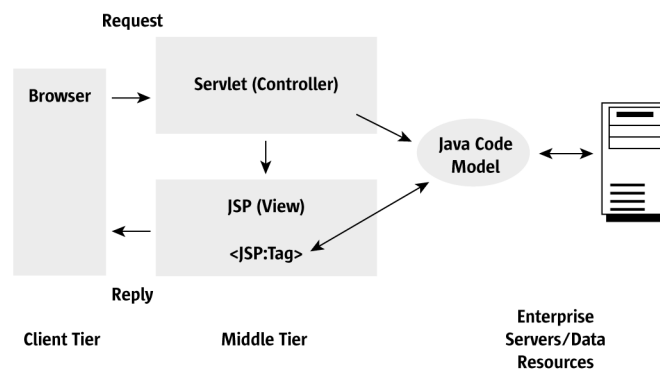


FIGURE 8–5: Template-Based Approach to Content Generation

Template approaches typically involve some degree of coupling between code and content. Implementations of the template approach use variations of software patterns, including “Mediator-View” and “Service-to-Workers” to reduce this coupling.

Events in the template approach tend to be coarse grained and nonhierarchical. This is because the event Model is typically tied to the MVC implementation, which makes it difficult to decompose the problem domain below the page level—that is, into individual components.

The combination of coarse-grained events and a page-centric view limits the complexity of flow within this type of framework. Hence, it limits the amount of functional complexity applications based on this approach can effectively handle. Sun's iPlanet Application Framework is a notable exception that provides fine-grained events coupled to widget-like UI components. As explain in Section 8.7.2, iPlanet Application Framework's implementation anticipates JSR 127 functionality, which will provide a standardized event and component Model that provides Swing-like behavior to template-based approaches.

## 8.5.2 DOM Manipulation

DOM manipulation is a standards-based, non-template approach that “pushes” data into the DOM representation of an HTML page. In other words, instead of taking the template approach of calling out to code from the HTML page, this approach uses code to manipulate the HTML page, thereby enforcing the separation of code from content. The W3C DOM specification is the standard for representing HTML pages.

The DOM approach eliminates the coupling of code and content, and results in presentation markup that is logic-free—which means that it is completely valid HTML, XML, and so forth. The resulting handshake between page designer and developer is thus simplified, which makes it easier to maintain and extend Web pages.

Pages in the DOM approach are represented as trees. Components are represented by widgets that are logically bound to various portions of that tree. The DOM approach allows fine-grained event handling. This is because the event Model is no longer tied to the MVC implementation but rather is implemented as an independent layer within the MVC framework. Typical implementations use inner classes like Swing that integrate easily with the component Model—that is, listeners can be added to the components.

Events in the DOM approach are hierarchical. This means that an event can generate other events, each of which is potentially handled by multiple listeners. Hence, program flow occurs at the component interface level, which enables this approach to have a more Swing-like behavior.

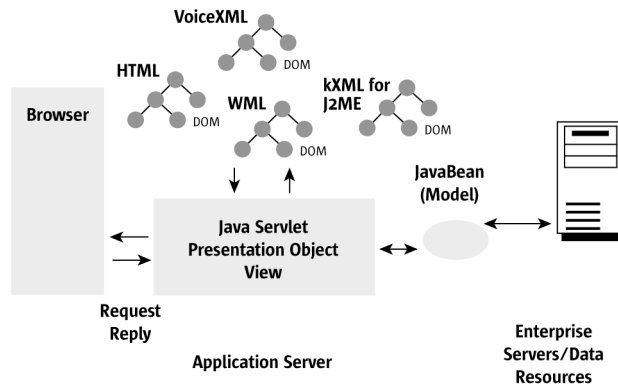


FIGURE 8-6: DOM-Based Approach to Content Generation

### 8.5.3 Advantages of DOM and Template Approaches

The template and DOM approaches to MVC implementations reflect their MVC “pull” and MVC “push” heritages respectively. Each has its advantages.

The JSP pages and taglibs provide a ready-made framework for calling code from a page in order to pull data into the markup. The DOM tree and CSS IDs provide a ready made framework for pushing data into an in-memory representation of the markup from Java code. Both approaches, however, are broad and applicable to a large range of applications. Therefore, they vary widely in their implementations. Developers are faced with having to learn the implementations of each framework that they use. This is because there are no standards for building, for example, a button, or registering an event listener to that button.

## 8.6 Java Specification Request™ (JSR) 127 Architecture

JSR 127, or JavaServer Faces™ specification, will provide an architecture and functionality that both the template and DOM approaches may standardize upon. The requirements of the architecture reflect the above issues that developers have encountered when building MVC-based frameworks that use these approaches as a foundation. Basically, the architecture must:



- Simplify development.
- Encourage/enforce separation of code from content.
- Define a model for connecting events in the UI to application behavior.
- Allow tools integration.
- Support diversity, that is, simplify usage of multiple presentation layers in the same application—for example, HyperText Markup Language (HTML) and Wireless Markup Language (WML)

## 8.6.1 JSR 127 Design Goals

The focus of JSR 127 is to create a graphical user interface (GUI) component framework that MVC-based implementations may standardize on. Its main goals are to:

- Create a standard GUI component framework which can be leveraged by development tools to make it easier for tool users to both create high quality GUIs and manage the GUI's connections to application behavior.
- Define a set of simple lightweight Java base classes for GUI components, component state, and input events. These classes will address GUI life cycle issues, notably managing a component's persistent state for the lifetime of its page.
- Provide a set of common GUI components, including the standard HTML form input elements. These components will be derived from the simple set of base classes that can be used to define new components.
- Provide a JavaBeans™ component model for dispatching events from client-side GUI controls to server-side application behavior.
- Provide a fine-grained object Model for dispatching events from client side GUI controls to server-side application behavior.
- Define APIs for input validation, including support for client-side validation.
- Specify a Model for internationalization and localization of the GUI.
- Automatic generation of appropriate output for the target client, taking account all available client configuration data, such as browser version.
- Provide for the automatic generation of output containing required hooks for supporting accessibility.

For a discussion of iPlanet Application Framework's anticipation of JSR 127, see Section 8.7.2.

## 8.7 Overview of the iPlanet™ Application Framework

The iPlanet Application Framework is Sun's implementation of a presentation framework. As a standards-based application framework for enterprise Web application development, it unites familiar concepts such as display fields, application events, component hierarchies, and a page-centric development approach with a state-of-the-art design based on both MVC and Service-to-Workers patterns. Although it is primarily intended to address the needs of J2EE technology developers building medium-sized applications, the iPlanet Application Framework also supports large- and massive-scale Web applications.

The iPlanet Application Framework provides core facilities for reusable components, making it useful to third-party developers who want to provide off-the-shelf components that can be easily integrated into Web applications. It is used as a platform for building vertical Web offerings, particularly because its extension capabilities provide a well-defined way for both end users and original developers to extend and leverage existing vertical features.

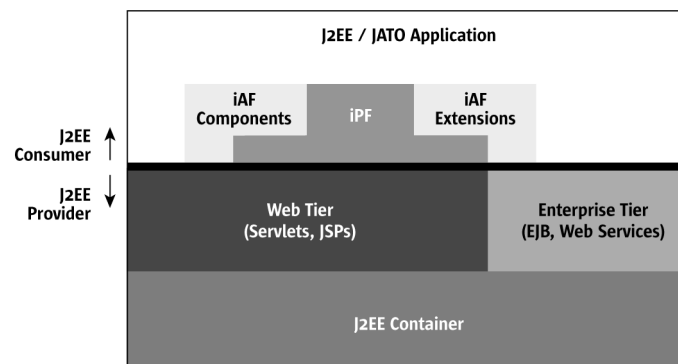


FIGURE 8-7: The iPlanet Application Framework Platform Stack

As shown in Figure 8-7, the iPlanet Application Framework provides a design-pattern-based skeleton for use by enterprise architects and Web application developers. It also provides a clearly delineated design that integrates in a well-defined way with other enterprise tiers and components.

Using the iPlanet Application Framework, developers can build reusable components by providing both low- and high-level infrastructure and design patterns. Developer-defined components are first-class objects that interact with the iPlanet Application Framework as if they were native components. Components can be arbitrarily combined and reused throughout an application, across applications, and across projects and companies.

The iPlanet Application Framework is not an enterprise tier framework, meaning that it does not directly assist developers in creating components based on the EJB specification (“EJB components”), Web services, or other types of enterprise resources. Although the iPlanet Application Framework is geared toward enterprise application development, it is properly a client of these enterprise tier resources, and thus provides a formal mechanism to access these resources.

### 8.7.1 The iPlanet Application Framework’s Implementation of MVC2

The iPlanet Application Framework currently implements the template approach to MVC, which is discussed in Section 8.3 above. The iPlanet Application Framework overcomes many of the limitations of MVC2 by formally defining View and Model entities that are independent of the technologies such as JSP technology that is used to implement them. JSP pages are, however, currently used to implement Views, and the iPlanet Application Framework makes use of the “Services-to-Work” pattern to factor code from content in this implementation.

Instead of a single servlet, the iPlanet Application Framework uses multiple servlets as entry points to a Web application. These servlets extend from a base Controller servlet which contains business and validation logic common to all Controllers.

The iPlanet Application Framework adheres to the MVC2 rule for updating Models before notifying Views. Servlets, after validating a request, pass the request to a “command” class for execution of its interface’s “perform” method—which updates the Model. Commands then specify an appropriate View which is rendered by a JSP page taglib, which in turn pulls the data from a “ViewBean” which is tied to the Model.

The iPlanet Application Framework provides a number of Model implementations that make it easy to connect to back-end data sources. The iPlanet Application Framework also addresses basic security issues inherent with their current JSP technology implementations that prevents direct access of the JSP pages by the clients. Finally, in anticipation of JSR 127, the iPlanet Application Framework provides “events” at the application level as well as the component level. The component level events provide a fine-grained component flow similar to that of Swing.

## 8.7.2 The iPlanet Application Framework and JSR 127

The iPlanet Application Framework and JSR 127 share many of the same design goals. For example, the iPlanet Application Framework's "core" provides an implementation of a set of simple lightweight Java base class files for GUI components. Included in the iPlanet Application Framework core are View-based primitives such as ContainerViews, TiledViews, and TreeViews. The iPlanet Application Framework builds on these primitives and provides a set of generic (horizontal) components that are extensible, as well as a set of vertical components that, while being less extensible, provide a particular look and feel.

JSR 127 recognizes that a severe limitation of template implementations of the MVC2 paradigm is the coupling of the event Model to its implementation. It therefore proposes to provide a fine-grained, object-based event Model for this paradigm.

The iPlanet Application Framework provides an implementation of this event Model. Typical MVC2 implementations provide only application level requests events. The iPlanet Application Framework provides three classes of events: general level events, specific requests events, and display events.

The general level events are used to respond to general application and request life cycle occurrences. Specific requests events occur based on a users action—for example, a button is pressed, and code is executed on the server corresponding to that button press. Field level display events are generated during the rendering of a page. These give the developer fine-grained hooks into the actual rendering process.

The main difference between the iPlanet Application Framework's handling of events and those of other MVC2 implementations is that the event is invoked on the component to which it pertains and can be as fine-grained as a button or a display field. Typical template approaches provide only one coarse-grained handler per page, which ties event-handling to the page and allows no decomposition of the problem domain into more finely grained components.

The iPlanet Application Framework will continue to track and incorporate into its implementation features of JSR 127. The main strength of the iPlanet Application Framework for the future is that its features have been designed around interfaces and object contracts, and are not tied to the implementation itself. Hence new capabilities, such as tool-readiness, will be easily accommodated into the framework.

### 8.7.3 The iPlanet Application Framework's Use of Design Patterns

The iPlanet Application Framework is based on state-of-the-art design patterns and techniques, including MVC, Service-to-Workers, Hierarchical View, Command, Business Delegate, and others. Furthermore, it is based on an *N*-tier JSP/Servlet architecture. The iPlanet Application Framework has been designed entirely around interfaces and object contracts that reflect these patterns. It is primarily an integrated set of cooperating design patterns and secondarily an implementation of those patterns.

Primary among these patterns is MVC. The iPlanet Application Framework addresses all three tiers of the MVC pattern. It defines formal View and Model entities with concrete relationships. It also provides an advanced logical Controller role that allows applications to scope Controller logic in appropriate ways. The iPlanet Application Framework's View tier incorporates JSP technology but is not synonymous with it. In the same way, the iPlanet Application Framework's Model tier incorporates other J2EE technologies, but is not synonymous with any of them. For these and other reasons explained below, the iPlanet Application Framework provides unprecedented extensibility for developers that other frameworks cannot match.

## 8.8 Types of iPlanet Application Framework Functionality

There are three logical groupings of iPlanet Application Framework functionality: the iPlanet Application Framework core, components, and extensions.

The iPlanet Application Framework core defines fundamental interfaces, object contracts, and primitives, as well as the minimal infrastructure required for iPlanet Application Framework applications. The iPlanet Application Framework core does not provide a component library, but provides the enabling technology for component authors.

Included in the iPlanet Application Framework core are View-based primitives like ContainerViews, TiledViews, and TreeViews, as well as Model-based primitives like DatasetModels, QueryModels, and TreeModels. The iPlanet Application Framework core also provides primitives for request dispatching and for reusable Command objects. Using these primitives, developers can easily create application-specific or reusable components that can be shared within or across projects. The iPlanet Application Framework core also includes high-level features that allow developers to immediately begin building highly functional applications.

The iPlanet Application Framework components leverage the iPlanet Application Framework core infrastructure to provide high-level, reusable components for application development. These components can come in a variety of flavors intended for different usage scopes.

For example, horizontal iPlanet Application Framework components will tend to be the most generic ones available, with their strengths being flexibility and customizability. These types of components will be usable by many different iPlanet Application Framework user populations, across projects and companies. Generally they will not be biased toward any particular look and feel.

Vertical iPlanet Application Framework components are tailored to a particular usage scenario, allowing them to provide high-level features and high ease-of-use. These types of components will be less broadly usable, but because their scope is better defined. This is because they can keep parameterization to a minimum and use a particular look and feel. All iPlanet Application Framework components can use all of the facilities provided by the iPlanet Application Framework core and can build upon its high-level features such as WebActions, SQL-based model implementations, and TreeViews.

The iPlanet Application Framework extensions provide access to facilities outside the J2EE platform in a way that is compatible with the iPlanet Application Framework. In many cases, iPlanet Application Framework extensions allow container-specific features to be used from iPlanet Application Framework applications seamlessly. Extensions differ from iPlanet Application Framework components in that they focus on technology integration rather than application development.

## 8.8.1 Technical Overview of the iPlanet Application Framework Core

The iPlanet Application Framework core is pure Java technology. Presented as an industry-standard Java Archive Software (JAR) file, it defines the following top-level packages:

- `com.iplanet.jato` – Request-handling infrastructure
- `com.iplanet.jato.command` – Command-related interfaces and implementations
- `com.iplanet.jato.taglib` – Custom JSP page tag library
- `com.iplanet.jato.model` – General Model-related interfaces and implementations
- `com.iplanet.jato.view` – General View-related interfaces and implementations

Each of these packages contains subpackages of more specific derivations, such as HTML-specific View implementations and Structured Query Language (SQL)-specific Model implementations. There are no formal packages or classes for iPlanet Application Framework components or iPlanet Application Framework extensions, which are purely logical classifications.

In writing an iPlanet Application Framework application, developers derive application-specific subclasses from existing iPlanet Application Framework classes or implement certain iPlanet Application Framework interfaces in an application-specific way. In most cases, developers will use the existing iPlanet Application Framework core implementations as superclasses, thus inheriting a great deal of useful behavior. (Component developers may be more likely to implement a set of iPlanet Application Framework interfaces directly.)

Application objects are organized around the central concept of a page. Each page consists of a rendering specification—normally a JSP page containing static content and markup plus custom iPlanet Application Framework tags—and one class comprising the root of the page’s view hierarchy. Each request to the server returns a page as the result. The page flow through an application is determined by the control logic written by the developer. There is no fixed relationship between one page and another beyond that provided by the developer.

In HTML, each rendered page generally contains one or more links or buttons which the end user can activate. Each activation of a link or button sends data back to the server, which results in invocation of a Command object specific to that activation. This Command object can take action itself or can delegate handling of the request to developer-defined event methods. Ultimately, the request is forwarded to a resource that is responsible for rendering a response to the client.

In most cases, this resource is an HTML-based JSP page that uses iPlanet Application Framework tag library to render dynamic content. The tag library uses iPlanet Application Framework View components to obtain the data it renders. These View objects are associated with one or more Model objects and draw data from them as needed. Thus, iPlanet Application Framework Views act as a hierarchical facade to any number of Models. These Views can be reused across multiple pages and with different Models. Models can generally be used by any number of Views since they have no display or View dependencies.

Once the user receives a response in a form of a page, he or she activates a link or button, which sends a request back to the iPlanet Application Framework application. The request is sent back to the same objects that rendered the page. This allows the iPlanet Application Framework infrastructure to map the submitted data back into the same Views (and thus Models) from which it originated, providing virtual persistence of this data. The developer interacts with his application objects and the submitted data as if there had never been an intervening response-request cycle. Once the data has been mapped back into the originating objects, the Command object specific to that link or button press is activated, and the cycle begins again.

The iPlanet Application Framework directly embraces J2EE technology standards like servlets and JavaServer Pages, while still allowing developers to freely use the features J2EE technology provides. The iPlanet Application Framework is not a container within a container, nor is it a layer meant to abstract the developer from J2EE technology. Instead, it adds to J2EE platform features that facilitate enterprise Web application development, while still letting developers interact with as much as or as little J2EE platform/iPlanet Application Framework as they like.

## 8.9 iPlanet Application Framework Features

The iPlanet Application Framework provides display fields, application events, component hierarchies, and a page-centric development approach, all of which will be familiar to developers using Swing, Delphi, Visual Basic, or PowerBuilder for client-side application development. iPlanet Application Framework integrates with application builders, such as Forte™ for Java software or JBuilder.

The iPlanet Application Framework provides an implicit, proven direction for both Web application architecture and application development, without precluding the use of other approaches. It does this by providing well-defined points of interaction with an application, as well as providing clearly defined ways in which to extend, augment, or override existing behavior.

Applications written in the iPlanet Application Framework will resemble one another more so than applications written using other frameworks. They will be more consistent, in their use of both high-level and low-level features, and will thus be more maintainable.



## 8.9.1 Symmetrical Display/Submit Handling

The iPlanet Application Framework assists with both the display and submit cycles in a symmetrical fashion, by virtue of its formal View tier. Whereas other frameworks loosely define their View tier as a JSP pages or some other kind of content-rendering technology, the iPlanet Application Framework makes a distinction between rendering specification (JSP pages) and View components. Only together are these considered the full View tier. An iPlanet Application Framework application defines primarily a hierarchy of View components, and then references these components from the rendering specification. The developer interacts with these View components in the same way during both display and submit cycles. The View components are the canonical View form.

## 8.9.2 Formal Model Entity

The iPlanet Application Framework allows the application to represent its data in a View-agnostic way. It also provides a formal mechanism for obtaining that data without implying a particular data format. Therefore, the iPlanet Application Framework provides a formal Model entity that defines a handful of standard methods that all Models must implement. Using an arbitrary, Model-specific key, Model consumers (including iPlanet Application Framework Views) can obtain Model data in a standard way, without any assumptions about how that Model internally represents its data.

For this reason, iPlanet Application Framework components can interact with any Model in the same way, allowing a different Model to be plugged into the same View. Models become interchangeable, and by this virtue, so does the data that they represent. The marshaling of data to a particular format purely for display becomes unnecessary, and the View tier need not understand the specific type of data with which it interacts. Different types of Models can coexist within an application, without the View tier being cognizant of any difference between their native data formats. XML/XPath, Java DataBase Connectivity™ (JDBC™) API, Java Data Objects (JDO), and other enterprise data all look the same to a Model consumer, and thus the iPlanet Application Framework is able to subsume the development approach of any framework concentrating on one of these data formats.

The interposition of a Model structure on an enterprise-tier resource enforces a level of abstraction that makes the application design more consistent and eases maintenance. In formally defining the data available from the enterprise tier, developers also define a formal, yet loosely coupled, contract between tiers of the application. This contract allows the application to be easily modified in the future, and in a well-defined way. The incidence of regressions is lower, and regressions are more readily apparent if they occur.

## 8.9.3 Application Events

The iPlanet Application Framework provides developers with a number of events for application-related occurrences. There are three types of events: general request events, specific request events, and display events.

General request events include events like `onBeforeRequest()`, `onSessionTimeout()`, and `onUncaughtException()`. Developers can use these events to respond to general application and request life cycle occurrences, as well as to error conditions. By default, error-related events use a consistent, localized mechanism to report errors to users, and can be overridden by developers to take application-specific action.

Specific request events occur based on user action. When a user activates a link or button (also known as a `CommandField` in the iPlanet Application Framework) on a page, the request results in the invocation of a `Command` object on the server corresponding to that activation. Although users can provide their own `Command` objects in response to such actions, the default `Command` implementation delegates handling of the request to a request-handling event method of the form `handle<name>Request()`, where `<name>` is the name of the `CommandField` the user activated. This event is invoked on the parent container of the `CommandField`, and thus is scoped to the component that originally rendered the link or button. Within this event handler, developers can take any action they like, either handling the request as they wish, or delegating the handling of the request to another object.

The main difference between this iPlanet Application Framework feature and similar request-handling features provided by other Web application frameworks is that the event is invoked on the component to which it pertains, and is fine-grained per link or button. Other frameworks generally provide only one coarse-grained event handler per HTML form, and the developer is left to conditionalize that code based on the user's action. This is both messy and hard to maintain as the set of fields changes. That approach also makes use of modular, self-contained components difficult, because the single event handler must be changed each time a new link or button is added to, or removed from, the form—regardless of whether it is contained within a component.

The iPlanet Application Framework provides fine-grained, field-level display events. Display events, which are invoked during the rendering of a page, give the developer hooks into the rendering process that simply would not otherwise be possible. From these events, developers can access the tag handlers as well as the JSP page context and output stream. Display events can be used to skip rendering of a field or to abort the currently rendering page altogether. They can also be used to tweak the outgoing content rendered by the JSP page, providing advanced content-filtering capabilities. Furthermore, display events encapsulate display logic pertinent to a component inside that component, thus providing a high degree of reusability for components even if they use advanced rendering techniques.

Display events keep Java code or program-like structures out of the JSP page. Any kind of programmatic construct in the JSP page is generally a maintenance problem, both because it exposes application functionality to the JSP page author and because parallel content must duplicate this functionality in potentially many places.

## 8.9.4 Hierarchical Views and Component Scoping

The iPlanet Application Framework provides a hierarchical namespace for HTML form fields that is not based on tightly coupled form-object concordance. Each display field view is created separately as a child of a parent container view and uses a simple local name within that container. It thus implicitly inherits a qualified, unique global name. These qualified field names are guaranteed never to conflict with other field names, even if local names are identical in other containers. Therefore, independent view components can be arbitrarily combined and will never conflict with one another. The iPlanet Application Framework automatically manages the mapping of form data associated with these qualified field names back into components during the submit cycle, so developers never have to think about how they combine components.

Developers do not use these qualified names during authoring of a JSP page. Instead, the iPlanet Application Framework provides what are called “context tags.” These tags define nested container and component scopes. Developers use local names in the JSP page within these scopes. These names are automatically and transparently translated to qualified names at runtime, using the current context. Not only can view components then be arbitrarily combined, but rendering specification fragments (JSP page fragments and pagelets) can be arbitrarily combined in a parent page. Thus, iPlanet Application Framework developers have two types of view component reuse at their disposal, and these types can be combined in several permutations.

## 8.9.5 Efficient Object Management

The iPlanet Application Framework reuses objects where it makes sense to do so but allows other objects to be allocated as needed. The common request-handling infrastructure of the iPlanet Application Framework relies on shared object instances managed by the container, but objects used by the developer during normal request handling are allocated as needed. Not only does this approach reduce complexity and eliminate an entire class of potential bugs for the iPlanet Application Framework itself, it does the same for application code.

This approach is maximally effective in production deployments, in which hundreds of requests per second are handled without significant latency or memory effects due to object allocation.

## 8.9.6 Support for Parallel Content

The iPlanet Application Framework provides full support for parallel content, which is the use of parallel sets of JSP pages, with each JSP page in the set customized to a particular language, target device, output markup (for example, XML, HTML, or WML), or any combination of these. Each of these JSP pages references the same view components, and thus contains only variations of content and markup. The application can then choose the most appropriate JSP page to render at runtime, based on user preference or any other desired criteria.

Parallel content works very well when trying to localize content for both Western and Asian languages, where page layout may diverge heavily, or when trying to render to different device types like a standard browser and an Internet-enabled cell phone. The advantage is that the business logic and View structure remain consistent across localized versions of the page, while allowing for sometimes significant rendering differences.

Some frameworks assume a static association between JSP pages and application component, or try to automate page flow using a declarative specification of the component-JSP page relationship. While this latter approach has its advantages in certain limited cases (yet many more significant drawbacks), it does not allow the flexibility needed for use of parallel content. Other frameworks that emphasize programmatic constructs in the JSP pages make the use of parallel content extremely difficult. Developers using these frameworks must copy and maintain programmatic constructs across multiple parallel JSP pages. Because the iPlanet Application Framework provides display events to keep programmatic constructs out of the JSP page, display logic never has to be replicated across parallel JSP pages in an iPlanet Application Framework application.

The iPlanet Application Framework provides full support for parallel content, making it extremely easy for applications to select a JSP page to render at runtime, based on any developer-defined criteria. The lookup for parallel JSP pages is also developer-defined, so parallel content can be organized in a way that makes sense to the application.

## 8.9.7 Ready-to-Use, High-Level Features

The iPlanet Application Framework provides high-level features that developers can use to rapidly build the following highly functional applications:

- WebActions allow developers to perform common, high-level tasks with a minimum of code. For example, developers can invoke the Next and Previous WebActions to automatically paginate through rows of data in a DatasetModel across requests. The dataset position is automatically managed across requests by the WebAction infrastructure, with no additional code necessary from the developer. Any model implementing the DatasetModel interface can be used with these WebActions.

- A set of SQL-based model implementations that automatically manage model-oriented access to JDBC technology resources. These implementations use SQL queries and stored procedures to retrieve and persist data in an RDBMS, all without the developer worrying about detailed JDBC technology use or the inconsistencies in JDBC technology-enabled driver (“JDBC driver”) usage. Of course, developers can use JDBC driver directly from within an iPlanet Application Framework application if they wish, but the presence of these value-added implementations in the iPlanet Application Framework core allows developers to very rapidly build functional, enterprise applications out of the box.
- The iPlanet Application Framework provides TreeView and TreeModel primitives that drastically simplify development of hierarchical data displays. These primitives are complemented by a set of custom tags that are agnostic about look and feel. This allows developers to structure a JSP page into portions that will be selectively rendered for a given tree node. Since these tags output no markup themselves, they can be used in JSP page fragments and pagelets to provide pluggable and customizable component look and feel.

## 8.9.8 Tool-Readiness

At the time of this writing, the iPlanet Application Framework does not yet provide tool-readiness. It has been designed around interfaces and object contracts. GUI-builder support will be added at a later stage. The result will be a framework that both provides development productivity using application builders and also supports advanced uses that make the framework ready for the enterprise.

## 8.9.9 Scalability

Because the iPlanet Application Framework has been optimized to eliminate *all* synchronization points, applications built on iPlanet Application Framework are as scalable as the J2EE platform container in which they run. iPlanet Application Framework introduces only a small, fixed amount of overhead to each application request.

## 8.10 Presentation Framework Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for presentation frameworks.

Interface Name	Level	Status	Reference *	Comments
JavaServer Faces™ (JSR 127)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/127.jsp">http://www.jcp.org/jsr/detail/127.jsp</a>	This standard will provide APIs for encapsulating and manipulating GUI components programmatically for both JSP and non-JSP software applications in a future version of the Sun ONE architecture.

### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

## The Portal Server

---

The Sun™ Open Net Environment (Sun ONE) platform Service Delivery components enable the delivery of user-facing Web applications and Web services running in the Service Container to end users using a variety of client devices—from desktop PCs to PDAs to mobile phones. One important Service Delivery component is the portal server. The portal provides a single access point to diverse types of information originating from many sources. It offers several features to end users, as well as to the Web applications and services that they are using. These features include aggregation, presentation, personalization, and security.

In order to provide these features, the portal makes use of internal services for location, presence, notification, and usage. These internal services can also be quite useful to Web services and applications outside the context of the portal. Therefore, they are exposed as public services.

The aggregation, presentation, personalization, and security features of the portal can be extended to the Web services world to control the delivery of one or more source Web services to a client Web service, rather than to a client device. Thus, as an enterprise deploys Web services throughout its organization, the portal can provide a useful way to aggregate, personalize, and deliver those services in a secure manner.

A portal server offering the interfaces described in this chapter—such as the iPlanet™ Portal Server—provides a comprehensive infrastructure for developers who are building many different types of portals. To conform to Version 1.0 of the Sun ONE architecture, a portal server must provide all the interfaces to support delivery of Web applications described in Section 9.1. Future versions of the architecture will add further requirements affecting delivery of both Web applications and Web services, as described in Section 9.2.

### 9.1 Using the Portal to Deliver Web Applications

As shown in Figure 9-1, today portal servers are primarily used to deliver content from Web applications. A portal allows multiple sources of information (typically Web applications) to be displayed within a single page or set of pages that the user can view in a browser. The page that includes the content is known as the desktop. The various sources of content appear in regions of the desktop called channels.

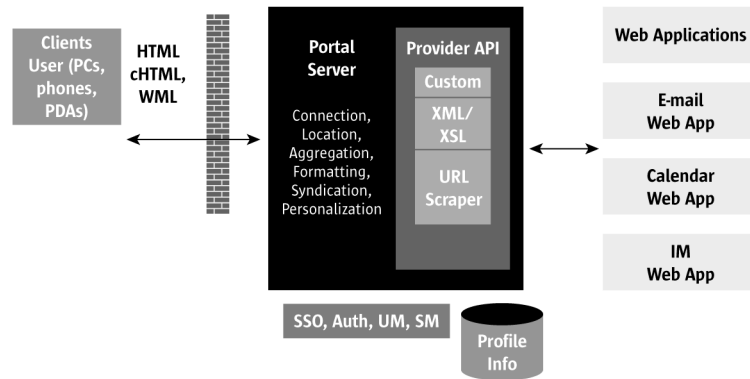


Figure 9-1: Delivering Content for Web Applications

In a portal that is part of the Sun ONE platform, a Java™ class file called a Provider is responsible for converting the content in a file or the output of a Web application into the proper format for a channel. Content Providers for the portal can be developed using the Content Provider API, and several pre-built specialized providers are included with the portal. Content is delivered to client devices in one or more markup languages such as HyperText Markup Language (HTML), HTML for cell phones and PDAs (cHTML), Wireless Markup Language (WML), or eXtensible Markup Language (XML). Authentication, authorization, and user management, along with storage of user profile information, is accomplished via Identity and Policy APIs, typically implemented on top of a directory server. Identity and Policy Services are discussed in Chapter 11, “Identity and Policy Services”.

### 9.1.1.1 Aggregation and Presentation

The portal aggregates information available from different data sources, including:

- HTML and XML content from different legacy application sources, such as mainframe, Enterprise Resource Planning (ERP), and database systems.
- Web applications, such as calendar and e-mail.

The portal must accommodate a variety of content sources aggregated from inside and outside an enterprise or from vertical or specialty portals. It provides for page layout and the creation of a customizable graphical user interface (GUI).



As briefly mentioned earlier, components responsible for translating and making content available for aggregation are called Providers. A Provider API is defined to abstract out the Provider functionality such as getting, editing, and processing content. In Version 1.0 of the Sun ONE architecture, a conforming portal server must implement the iPlanet Portal Server Content Provider API. Under the Java Community Process, the Java Specification Request (JSR) 168 Expert Group is working to define a Portlet API that will offer a standard programming interface for developing Providers. Providers that are built with this API should be able to run in portal server products from different vendors. A future version of the Sun ONE architecture will add a requirement to implement this Portlet API.

Using the iPlanet Portal Server Provider API, a developer can write a Content Provider that plugs into the portal desktop. The desktop delegates the rendering of information to a set of Providers. A Front Provider presents the user's home page and navigational links to other Providers. It uses the Provider API to invoke Content Providers to display source-specific content in the form of channels. For HTML devices such as browsers, these channels appear as table cells. On WML devices, these channels appear as links to WML cards containing the content of the channel. Content and Layout Providers present interfaces that allow the selection, ordering, and positioning of channels within the desktop. iPlanet Portal Server uses XML Stylesheet Language Transformations (XSLT) translation/formatting capability to generate appropriate format for a client device. For further information on the iPlanet Portal Server Provider API, see the interface table at the end of this chapter.

The types of clients that can access a Version 1.0-compliant portal must include desktop PC browsers, PDAs, and mobile phones and their associated markup languages: HTML, XML, WML, and cHTML.

### 9.1.1.1 Specialized Providers for Aggregation and Presentation

The portal offers the following four pre-defined Provider implementations that simplify the aggregation and presentation of common types of data sources:

- **The URLScrapper Provider** allows the creation of channels by obtaining and rendering data. The data indicated by a URL must be in a format appropriate to the client device.
- **The JavaServer Pages™ (JSP™) Technology Provider** allows the use of files provided by JavaServer Pages technology (“JSP files”) to create the content for a channel and the logic for processing requests.
- **The XML/XSL Provider** enables the creation of channels from any valid XML source using an XSL template. This Provider retrieves the XML data from the location indicated by a URL and transforms it using an XSLT engine to the specific markup language of the client device.

## 9.1.2 Personalization for Users and Applications

The portal maintains a multi-level profile (organizations, roles, and users) that allows comprehensive personalization and customization for both end users and applications. Personalization data enables context-sensitive interactions to tailor content delivered to the user according to the time, place, and type of transaction. This profile data reflects the choices made by users and applications regarding how they interact.

Users can customize their content and layout in a device-independent way, then let the portal attempt to determine how best to render the final output based on these customizations. But they can also customize according to specific devices. Similarly, services and applications can adjust their content or actions, depending on the device and its properties.

The portal allows end users and administrators to customize the desktop view layout and look and feel via a GUI that enables the editing of attributes such as colors and themes of the desktop and the selection of channels. In addition, custom Content Providers can be selected for different devices, and the selection of channels can be made on a per-device basis. In iPlanet Portal Server, the look and feel of the desktop can be changed completely by replacing appropriate JSP files.

Application developers can customize channels by implementing custom Providers. Four pre-defined specialized Provider implementations are outlined in Section 9.1.1.1.

## 9.1.3 Security for Users and Applications

The portal must support secure access for end users and applications both inside and outside the enterprise. In doing so, it must:

- Offer the flexibility to support an enterprise's existing authentication mechanisms and provide single sign-on capabilities.
- Utilize the Identity and Policy components of the Sun ONE architecture to provide single sign-on, authentication, authorization, access control, and session management for all its users and applications.
- In addition, the portal may optionally:
  - Provide a Virtual Private Network (VPN) solution for accessing intranet resources without any specialized client software in the device, other than a Web browser.
  - Include a reverse proxy that rewrites HTML documents, thereby allowing all intranet Web sites to be accessed without exposing them directly to the Internet.

## 9.1.4 Management of Users and Applications

The portal offers an administration console accessible via a Web browser that allows the customization and management of both users and applications.

## 9.2 Enhancing the Portal for Web Services

The aggregation, presentation, personalization, and security capabilities that the portal provides to Web applications can be extended to the Web services world. First, the portal can offer a single point of access to user-facing Web services. In addition, it can be used as the access point for delivering aggregated and personalized Web services to other Web services (as opposed to an end user). Furthermore, internal services of the portal such as location, presence, notification, and usage can be made available as public Web services to other Web services and applications.

This section describes portal interfaces beyond those described in Section 9.1. As shown in Figure 9-2, these interfaces enhance the portal so it can offer the above features for Web services. Future versions of the Sun ONE architecture will specify these interfaces in greater detail.

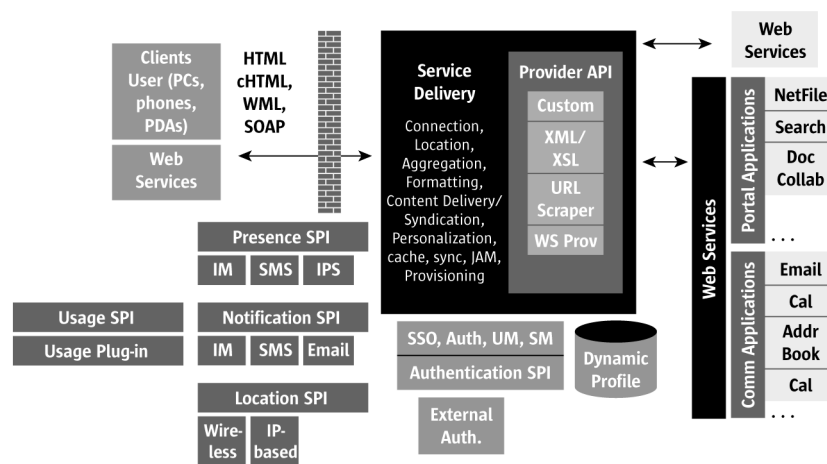


Figure 9-2: Delivering Content for Web Applications and Web services

### 9.2.1 Aggregation and Presentation of Web Services

There are two use cases for the aggregation and presentation of Web services:

- The consumer of the aggregation is an end user.
- The consumer of the aggregation is another Web service or a Web application.

These two use cases are discussed in the sections that follow.

### 9.2.1.1 Delivery to End Users

In this case, the source Web services are user-facing in the sense that they produce output that is intended for human users with simple transformations required for presentation. The Provider API must be able to handle sources that are Web services and define the allowable transformations of Web service data. The capabilities required include identifying and discovering services, formatting requests for services from the user, and translating output to the appropriate format for the user.

Web services definitions are not yet standardized for different application segments such as calendar, mail, and instant messaging. Until this happens, the portal can act as the integration point that converts different Web services interfaces for similar services to a common interface for delivery to users.

### 9.2.1.2 Delivery to Web Services and Applications

A Web services portal consolidates several Web services offered by an organization into a single access point. In this manner, it acts as a Web services hub.

Such a portal can be the edge application (Internet-intranet bridge) for partner, vendor, supplier, and customer applications and services. This function enables single sign-on (SSO) for Web services and provides a simpler way of handling security and access control. The portal is not the container running the Web services but a gateway to them.

## 9.2.2 Personalization for Web Services

A Web services portal can customize and personalize the delivery of Web services as well as Web applications. Different client users, applications, and services can have access to different Web services. Also, different clients can be connected with different back-end Web services offering similar but differentiated services depending on personalized information. The portal can manage what services are available to applications and other services based on their roles by maintaining a profile for each client application and service.

### 9.2.2.1 Supporting the Java Web Client Model

An enhanced portal could support client devices with richer interfaces than a markup language. Client devices may be provisioned with a Java application, applet, or midlet, as described in Chapter 10, “The Java Web Client Model.” The portal can use the customized information it has about the user, the device, the connection, and the application/service being accessed to provision the device appropriately. The APIs that are under development as part of JSR 124 - Java 2 Platform, Enterprise Edition (J2EE™ platform) Client Provisioning Specification (also known as the Vending Machine specification for Java) may be used by the portal to accomplish this.

### 9.2.2.2 System-Level Interfaces Supporting Personalization

The portal uses information about users, devices, applications, and services to personalize the delivery of applications and services to end users. Some of this information is stored as profile data that can be edited by users or administrators. Other information is more dynamic and needs to be tracked by the portal. A future version of the Sun ONE architecture will define system-level interfaces for the portal that allow vendors to plug in value-added modules for enhancement of the personalization of application and service delivery.

#### 9.2.2.2.1 *Location SPI*

The Location System Programming Interface (Location SPI) allows developers to provide modules that enable the portal to determine the geographic location of a user. The portal uses a collection of such modules to track users to provide personalized delivery based on location.

#### 9.2.2.2.2 *Presence SPI*

Presence refers to comprehensive information about the current context of the user, including how the user is connected to the system, the user's availability, and the types of applications in use. The Presence SPI allows the developer to provide modules that enable the portal to gather Presence information. The portal uses a collection of such modules to provide personalized delivery.

#### 9.2.2.2.3 *Notification SPI*

The portal uses user profile information and current user context to choose the method used to send messages to a user. The Notification SPI allows developers to provide modules that enable the portal to send messages via various mechanisms. The portal uses a collection of such modules to send messages in a personalized way.

#### 9.2.2.2.4 *Usage SPI*

The portal can collect utilization information about a client's use of Web services or applications. The Usage SPI allows developers to provide modules that gather usage information. The portal uses a collection of such modules to track usage on behalf of applications and services.

### 9.2.3 Security for Web Services

The portal enables single sign-on and access control for Web services and applications as well as for end users. The Web services and applications do not need to be single-sign-on aware. The portal simplifies the usage of Identity and Policy components by the Web services it consolidates.

### 9.2.4 Management of Web Services

Management interfaces for the portal (beyond simple Web browser access) will be defined in a future version of the Sun ONE architecture. The portal will allow management of Web service and Web application clients as well as end users. It will also permit the management of the Web services that it is currently aggregating.

### 9.2.5 Specialized Web Service Provider

The portal will offer a specialized Web Services Provider implementation to enable automated user interfaces (UIs) to Web services. This provider will be able to dynamically craft a UI for interactions with “simple” Web services for different types of client devices. The tables that follow provide information relevant to the portal server.

## 9.3 Portal Server Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for portal servers.

Interface Name	Level	Status	Reference *	Comments
iPlanet Portal Server Content Provider API	Application	Footnote 4	<a href="http://docs.iplanet.com/tech/ips/">http://docs.iplanet.com/tech/ips/</a>	iPlanet Portal Server PAPI
Portlet API (JSR 168)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/168.jsp">http://www.jcp.org/jsr/detail/168.jsp</a>	Standard Provider API
Presence SPI	System	Footnote 6	N/A	
Notification SPI	System	Footnote 6	N/A	
Usage SPI	System	Footnote 6	N/A	
Location SPI	System	Footnote 6	N/A	

### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

Footnote 4: This is a published proprietary interface, and support of this interface is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 5: This is an unpublished proprietary interface. A published definition for this interface will be provided in a future version of the Sun ONE architecture.

Footnote 6: This interface is not yet defined. A definition for this interface will be provided in a future version of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.





## The Java Web Client Model

This chapter describes the component that is most changed from traditional system architectures: the Web client enabled with Java™ technology (“Java Web client”).

Services on Demand can be delivered through the Sun™ Open Net Environment (Sun ONE) architecture in three ways, as shown in Figure 10–1.

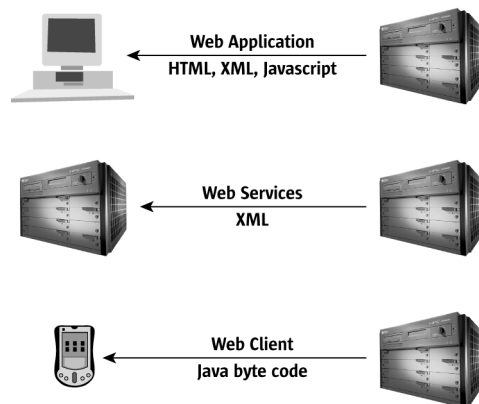


Figure 10–1: Delivering Services on Demand

The three delivery methods shown above can be defined as follows:

- **The Web application model:** HTML, XML, or JavaScript™ technology sent from a Web server to a browser. These delivery methods are designed for human interaction.
- **The Web services model:** XML-based Web services transacted between two servers. These services can be transactions completed by the servers without human interaction.
- **The Web client model:** Applications built with Java technology, delivered over the Web (provisioned) to a Web-enabled device—for example, a desktop computer, a PDA, a Web-enabled cell phone, or an appliance or machine.

This chapter explores the third option: applications written in the Java programming language that are provisioned to Web client devices and executed on those devices.

## 10.1 Design and Deployment Considerations

Although many applications built with Java technology are intended to run on desktop computers, the Web client model anticipates that many of them also will be delivered to other devices with limited processing capability; limited memory; and unreliable, intermittent Internet connectivity. The following sections describe the elements of the Sun ONE architecture that address the environmental and device-related constraints that the Web client model is designed to overcome.

### 10.1.1 Protocols and Payloads

The devices to which Java Web client applications are delivered are frequently hand-held, wireless devices that are always on the move. Connecting to the Web in different ways, these devices are typically separated by firewalls from the server delivering the Web client application. Therefore, the developer of the Java Web client application cannot depend on the availability of proprietary protocols when delivering services to client devices that can connect anytime, anywhere.

Because the availability of proprietary protocols cannot be assumed, the Sun ONE architecture provides the means to use a very few, very pervasive protocols to handle all aspects of Web client transactions, including provisioning, data transfer, and asynchronous reliable messaging.

Furthermore, the developer of the Java Web client application must plan for the application to be delivered to a range of client devices with a wide variety of operating system versions, physical capacities, and access protocols. Web client applications that demand an exact fit between provider and consumer of a payload will succeed far less often than those that can tolerate an approximate fit.

The Sun ONE architecture Java Web client payload is defined by an exposed XML message schema that can be accessed at design time, during deployment, and at runtime. Because these schemas allow for an inexact match, it is easier to provide robust support for a variety of client device types, or for different versions of the same client device.

## 10.1.2 Environment

The Java Web client operates in an environment that is distinctly different from that of the traditional client. Often access must be provided with intermittently connected, low bandwidth, high latency, indifferently secured, and relatively expensively tariffed communication. However, in spite of such limitations in the underlying communication fabric, the Java Web client application must provide secure, cost-effective access to services. It also must remain responsive to user input despite intermittent connectivity.

Devices must be able to operate anywhere, including in store franchises, branch offices, homes, and moving vehicles. Furthermore, this operability must be sustained where local support is lacking, which is almost always the case. For example, even if the user is a direct employee of an enterprise providing a service, first-level support for that user may be outsourced to communication carriers with no knowledge of, and a limited stake in, enterprise back-office operations.

Devices used for access are in a state of constant advancement, and the scale of services is undergoing steady expansion. Therefore, users need to be readily provisioned with a set of services whose implementation and deployment details can vary as new devices and services become available.

## 10.1.3 MVC Design for Java Web Clients

Unreliable connectivity is a fact of life for Web client devices. For this reason, a robust Java Web client application must:

- Anticipate the loss of connectivity at any time.
- Provide a graceful way to continue operating until the connection is restored.
- Be capable of completing transactions that were interrupted by the loss of the connection.

The Model-View-Controller (MVC) design model, discussed at greater length in Section 8.2, “The Model-View-Controller,” has important implications for Java Web client applications. Hence, it also has major impact on the Sun ONE architecture that supports their creation, assembly, and deployment.

In a typical client-server transaction using a reliable network connection, the client application can be exclusively concerned with the presentation of information—or, in MVC terms, with the View. A Web client application, on the other hand, must be prepared to manage some aspects of both the Model and Controller. The Sun ONE architecture supports an MVC approach to Java Web client design.

The Sun ONE architecture model for Java Web clients assumes that the client device has the inherent capability to display text, graphics, and controls such as buttons and fields. The View can therefore be managed declaratively by using XML, rather than procedurally by declaring what is to be displayed and leaving the client device to determine how to make that happen.

As discussed earlier in Section 10.1.1, the information payload must be defined as loosely as possible to support as wide a range of client devices as possible. Therefore, the Model, like the View, is managed declaratively, with both the schema and the information content delivered as XML.

If intermittent connectivity is the rule, some business logic must always be present on the client device to maintain functionality when the connection is lost. Therefore, the third component of MVC, the Controller, must also be present in a Java Web client application. The Controller is, of course, procedural in nature, rather than declarative. Furthermore, given the demands of the Web client environment, the vehicle for delivering procedural functionality must be both platform-independent and scalable. Proprietary languages are platform-specific, and the JavaScript programming language is unmanageable for anything except very small applications. Therefore, the Sun ONE architecture specifies that the Controller function be managed using Java technology byte codes. Java Web client applications use Java technology byte codes to deliver the Controller elements of the application, and XML to deliver the Model and View elements of it.

## 10.1.4 Supporting Architectural Elements

The Sun ONE architecture helps developers deal with the environmental challenges that often confront Java Web client applications. For example:

- Asynchronous reliable messaging, cache control, and synchronization facilities for local stores help Web client applications deal with unreliable connectivity in the client environment.
- Encapsulation of signing and encryption services in devices provide secure, validated communication. These same architectural elements reduce the risk of customers repudiating transactions that have been completed using Web client applications. Repudiation occurs when a customer attempts to deny having completed an online transaction. The Java Web client infrastructure must be capable of providing proof of transactions.
- Abstraction of the client application development and deployment model makes it easier to support a variety of client devices, even through semi-automatic interface generators. Declarative configuration of runtimes provides predictable operation tuned to device requirements by device-specific runtimes, rather than requiring separate applications written for each device.

- The developer of a Web client application must frequently anticipate delivering it to any of dozens—possibly even hundreds—of different client device types and versions. Customization of runtime configurations using predefined device profiles provides a scalable way to provision many different types of client devices over the Web.
- Services that are accessed over the Web can be discontinued, moved to another URL, or temporarily rendered unavailable by design or mishap. If this happens, the Java Web client that cannot find another way to access the same or an equivalent service will fail. Moreover, a change in the user’s personal profile might require a change in access points. For example, when a user moves a stock portfolio from one brokerage firm to another, the access point for finding the current value of the user’s portfolio probably changes, too. By deferring the binding of access points until access is needed, the application developer can help forestall the temporary unavailability of services.

## 10.1.5 XML Information Services and Device Interaction

Local storage in message queues, caches, and device-specific stores is implicit in a Java Web client architecture. When the architecture for Java Web clients is considered through the MVC design model, the store or “Model” is XML. Driving XML into the device means that disconnected clients can present information collected while online and then update server-side elements when reconnected.

However, the footprint limitations of high-volume Java Web client devices such as cell phones and PDAs limit the capabilities of client device XML-processing to handle, for example, only valid XML structures. Properly written Web client applications conserve limited device memory by instantiating objects from XML structures only when they are needed. Another way to save memory is to make the XML structures breadth-first cacheable by pieces rather than the whole—that is, to keep only the top nodes of a tree in memory, instantiating lower nodes only when necessary.

Another recommended practice is to use token-based compression of XML structures on the server to conserve bandwidth, or on the client device to conserve storage. This practice also requires that the portal customize compression schemes on the fly to accommodate dynamically discovered device limitations. A lightweight XML parsing engine and document object navigation interface support these design considerations, as discussed in more detail in Section 3.2, “J2EE Platform Container Provided Services.”

## 10.1.6 Client Device

One of the most important considerations when developing Java Web client applications is the range of client devices that will attempt to download the application. The following sections discuss device-related issues.

### 10.1.6.1 Java™ Virtual Machine (JVM™) and KVM Operating Environments

Full-scale Java™ Virtual Machine (JVM™)<sup>1</sup> software and reduced-capability Kernel Virtual Machines (KVMs) are available on both traditional workstation-like machines and, particularly for KVMs, on more limited and portable client devices.

JVM software and KVMs provide a common application and application library execution environment that has been adopted by a diverse set of Web services client device manufacturers. They offer secure operations for accessing underlying device-specific capabilities in a way that insulates the developer from those interfaces while providing device-specific integration. Section 10.2, “Java Web Client Model Interfaces,” provides information relevant to the Java 2 Platform, Standard Edition (J2SE™ platform) release that is aimed at workstation JVM operating environments.

### 10.1.6.2 Mobile Devices

Mobile devices represent a class of edge-node devices that are both limited in their capability and potentially much more pervasive than traditional workstations. Their limitations and their more personal character lead to trade-offs that require new sensitivities and different architectural approaches. The user of a mobile device expects to log into the Internet and access personal configurations both from multiple devices and from a single device that can be carried along and used at various locations. Accommodating this dual expectation in the limited storage, computation, and communications environment of mobile devices is the architectural challenge.

As outlined earlier, mobile environments include two classes of operating environments—one based on the JVM software and the other on the KVM. The JVM software environment supports the Connected Device Configuration (CDC). In turn, the CDC supports the Foundation Profile. This is extended by the Personal Basis Profile (used, for example by AutoJava and JavaTV) and the Personal Profile (available, for instance, on iPAQ devices). The KVM supports the Connected Limited Device Configuration (CLDC). In turn, the CLDC supports the Mobile Information Device Profile, variants of which are provided by the harmonized HL7 Reference Model (RIM) and NTT DoCoMo.

## 10.1.7 Extended Services

When written for mobile devices, Java Web client applications execute on top of the device's profile. They use the data access, presentation, user interaction, and communication services provided by the profile.

1. The terms “Java Virtual Machine” and “JVM” mean a virtual machine for the Java platform.

Mobile environments provide a different kind of user interface environment that is limited to interaction with documents and very simple document creation. Without the more capable document creation facilities characteristic of desktops, access to services must be engineered using facilities for the service provision point for the device. This type of access is discussed within the context of server-side provisioning in Section 10.1.9, “Server-Side Provisioning.” However, the device must have the basic infrastructure—reliable messaging, security, synchronization, and access to local storage—required to interact with the server-side provisioning point.

Queuing interactions for intermittent connection are required for reasonable operation with mobile devices. Java Message Service (JMS) provides this capability for the programming interface. Furthermore, the JMS API allows for transacted operations on queues, including those identified by name rather than location.

With JMS technology, multiple source and targets can be identified through a common name, thus allowing information to be published without possessing lists of users who subscribe to that information. The Electronic Business XML (ebXML) Trap (ebMS) Simple Object Access Protocol (SOAP) extensions for reliable messaging create a Web service interface that provides this capability.

Device stores are used both to allow disconnected operation and to provide a common form of application intercommunication. Access to these stores, especially for application intercommunication, must be controlled through keyed access permissions. Because a mobile device must meet the user’s need to log on both from multiple devices and from a single device that can be turned on and off in various locations, it must have device stores that are bi-directionally synchronized with matching information held on servers. Changing from one device to another requires that information on devices be viewed as caches. Operating while disconnected requires that the caches be populated with current information. Meeting both types of requirements leads to the need for compensating transactions. These can be as simple as mailed or SMS notifications asking for resolution. They may also involve more automated means requiring integration servers running long-lived transactions.

Offering the basic means for nonrepudiation of user transactions and the installation and operation of trusted applications requires security support in the devices. Because such support cannot be compromised, it must be isolated in its own environment with carefully engineered access points to the device.

Note that the storage and communication mechanisms on devices are still evolving. New mechanisms may introduce new capabilities.

## 10.1.8 Telephony Access Mechanisms

The convergence of communications and computation happens on the Web Service access client device. Notifications arrive from a short message service (SMS) providing a Uniform Resource Identifier (URI) for a service access point. In the other direction, a service may provide an address book entry for a voice call and offer to dial it.

## 10.1.9 Server-Side Provisioning

Almost as a matter of course, Web pages are becoming customized for client support. portal servers and gateways provide client customization of the data that is exchanged with the client. They also customize the client-side behavior that provides local interaction with the data that is stored and queued on the client. Therefore, the use of caches to scale Web services requires the caching of fragments of data composed in all the variety that a client might need, rather than sending the final page to some particular client.

Compiling JavaScript technology to Java technology byte codes is an easy way to reduce device footprint and deal with legacy environments. In the Wireless Application Protocol (WAP) environment, controllers are described by URLs. These de-reference as Java technology byte codes if the WAP portal compiles JavaScript technology to them. The advantage is that already-generated WAP pages already can be supported on devices without dedicating footprint to a JavaScript programming language interpreter. Over time, as the burden of maintaining some collection of JavaScript programming language files surpasses the value that it provides, its behavior can instead be represented as Java class files for ease of maintenance.

In a more developed provisioning strategy, devices can obtain the Model-View-Controller (MVC)-factored resources they need for local interaction with data and access privileges for data supplied by Web clients, as described in Section 10.1.3, “MVC Design for Java Web Clients.” They also get registration with identity, profile, and preferences for the services they access and the way in which they access them.

Provisioning a device in this manner has implications for service billing. The “vending machine” outlined in Java Specification Request (JSR) 124 can be connected, for example, to the OSS/J systems of a carrier or directly to the billing systems of the appropriate service providers.

## 10.2 Java Web Client Model Interfaces

Interface Name	Level	Status	Reference *	Comments
J2SE 1.4 Release (JSR 59)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/59.jsp">http://www.jcp.org/jsr/detail/59.jsp</a>	Aimed at workstation JVMs Section 10.1.6.1
J2ME Platform Specification (JSR 68)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/68.jsp">http://www.jcp.org/jsr/detail/68.jsp</a>	Aimed at mobile devices Section 10.1.6.2
J2ME CDC (JSR 36)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/36.jsp">http://www.jcp.org/jsr/detail/36.jsp</a>	Alternative configuration Section 10.1.6.2
J2ME Foundation Profile (JSR 46)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/46.jsp">http://www.jcp.org/jsr/detail/46.jsp</a>	Alternative profile Section 10.1.6.2
Personal Basis Profile (JSR 129)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/129.jsp">http://www.jcp.org/jsr/detail/129.jsp</a>	Alternative profile Section 10.1.6.2



Interface Name	Level	Status	Reference *	Comments
Personal Profile (JSR 62)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/62.jsp">http://www.jcp.org/jsr/detail/62.jsp</a>	Alternative profile Section 10.1.6.2
MidP (JSR 37, 118)	Application	Footnote 1	<a href="http://www.jcp.org/jsr/detail/37.jsp">http://www.jcp.org/jsr/detail/37.jsp</a> and <a href="http://www.jcp.org/jsr/detail/118.jsp">http://www.jcp.org/jsr/detail/118.jsp</a>	Alternative profile Section 10.1.6.2
CLDC Next Generation (JSR 139)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/139.jsp">http://www.jcp.org/jsr/detail/139.jsp</a>	Alternative configuration Section 10.1.6.2
PDA Profile for the J2ME Platform (JSR 75)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/75.jsp">http://www.jcp.org/jsr/detail/75.jsp</a>	Alternative profile Section 10.1.6.2
J2ME Widget Sets, User Events, and Communication	Application	Footnote 5		Several ISVs have products with this capability. Section 10.1.7
Device JMS and SOAP	Application	Footnote 5		Several ISVs have products with this capability. Section 10.1.7
JDBC (Native Stores, Application Interconnection)	Application	Footnote 5		Several ISVs have products with this capability. Section 10.1.7
SyncML	Application	Footnote 2	<a href="http://www.syncml.org/technology.html">http://www.syncml.org/technology.html</a>	Synchronization Section 10.1.7
Device Cache Management	Application	Footnote 2	<a href="http://www.jcp.org/jsr/detail/107.jsp">http://www.jcp.org/jsr/detail/107.jsp</a>	Section 10.1.7
Java APIs for Bluetooth (JSR 82)	Application	Footnote 2	<a href="http://www.jcp.org/jsr/detail/82.jsp">http://www.jcp.org/jsr/detail/82.jsp</a>	Section 10.1.7
Java APIs for Security, Signing, Trust, Signature, Encryption, and the Javacard	Application	Footnote 2	<a href="http://java.sun.com/products/javacard/javacard21.html">http://java.sun.com/products/javacard/javacard21.html</a>	Section 10.1.7
Javacard Integration into Phones, PDAs, XML, Trust, Signature, Encryption, (JSRs 105-107)	Application	Footnote 2	<a href="http://www.jcp.org/jsr/detail/105.jsp">http://www.jcp.org/jsr/detail/105.jsp</a> and <a href="http://www.jcp.org/jsr/detail/106.jsp">http://www.jcp.org/jsr/detail/106.jsp</a> and <a href="http://www.jcp.org/jsr/detail/107.jsp">http://www.jcp.org/jsr/detail/107.jsp</a>	Section 10.1.7
Wireless Telephony on J2ME (JSR 120)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/120.jsp">http://www.jcp.org/jsr/detail/120.jsp</a>	Wireless Telephony Communication APIs (WTCA) Section 10.1.8
Phonelets (JSR 61)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/61.jsp">http://www.jcp.org/jsr/detail/61.jsp</a>	API to package, deploy, and run Computer Telephony Integration (CTI) applications Section 10.1.8

Interface Name	Level	Status	Reference *	Comments
JAIN 3G MAP Mobile Application Intercommunication (JSR 123)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/123.jsp">http://www.jcp.org/jsr/detail/123.jsp</a>	Service Provider Presence and Availability Management API Section 10.1.9
Service Location Protocol (JSR 140)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/140.jsp">http://www.jcp.org/jsr/detail/140.jsp</a>	Service Location Protocol Application Programmer Interface Section 10.1.9
Service Provider Presence (JSR 123)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/123.jsp">http://www.jcp.org/jsr/detail/123.jsp</a>	J2ME application environment for network-connected devices Section 10.1.9
J2EE Client Provisioning Specification (JSR 124)	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/124.jsp">http://www.jcp.org/jsr/detail/124.jsp</a>	Abstracts details of the provisioning model Section 10.1.9
JAIN Service Creation Environment	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/100.jsp">http://www.jcp.org/jsr/detail/100.jsp</a>	API to support and simplify the creation of portable telecommunication services Section 10.1.9

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

Footnote 4: This is a published proprietary interface, and support of this interface is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 5: This is an unpublished proprietary interface. A published definition for this interface will be provided in a future version of the Sun ONE architecture.

Footnote 6: This interface is not yet defined. A definition for this interface will be provided in a future version of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

## Part 6. Fundamental Services

---



## Identity and Policy Services

---

Identity and Policy Services include three broad categories of services, all of which generally have a dependency on directory services:

- Identities, roles, and security for users, groups of users, and other system objects.
- Federated identity systems such as the Liberty Alliance Project.
- Management Services, which include both systems and applications management.

As shown in Figure 11–1, Identity and Policy Services are located at a lower level in the Sun™ Open Net Environment (Sun ONE) platform. They are used by most of the higher-level components.

The first category of services is directly related to existing directory server interfaces such as those offered by the iPlanet™ Directory Server Access Management Edition and Integration Edition products from Sun. Federated identity interfaces based on work underway in the Liberty Alliance Project will extend services in this first category to operate across the Web. The Sun ONE architecture also defines a common management infrastructure for all components. Identity and Policy services leverage platform functionality whenever possible—for example, in the incorporation of Kerberos security.

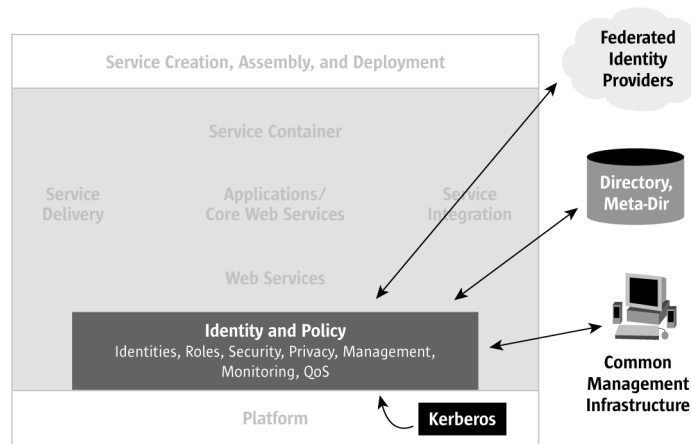


Figure 11-1: Identity and Policy Architecture

## 11.1 Identity, Roles, and Security

Identity and Policy Services provide the infrastructure for managing user identity, roles, and security—as well as services and policies. They also provide fundamental basic services such as discovery of Web services using Universal Description, Discovery, and Integration (UDDI,) authentication, single sign-on (SSO), policy evaluation, and security. These fundamental Core Services can be leveraged by other Web Services and applications to perform the basic operations. For more information, see Chapter 13, “Core Web Services.”

Figure 11-2 shows the overall architecture of the Identity and Policy services. In that figure, the architecture is broadly divided into:

- Infrastructure management components
- Core components
- Web services

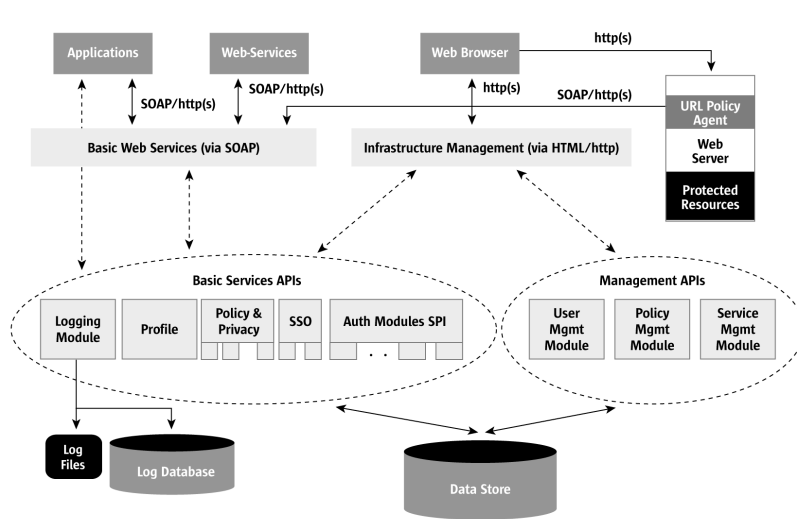


Figure 11-2: Identity and Policy Architecture

As shown Figure 11-2, the infrastructure management modules provide interfaces to manage users, policy, and services via a Web browser. The User Management component creates, deletes, and modifies user identities. It also provisions users for various services.

The Policy Management component provides interfaces to create, delete, and modify policy rules that define the service and access privileges for users. For example, the protection of a Web resource can be based on a user's identity as well as on conditions such as "can be accessed only from 9AM to 5PM, Monday through Friday."

The Service Management component provides interfaces to manage Web services using UDDI and interfaces to manage service configuration parameters. Additionally, administrative operations can be performed on service instances such as starting and stopping the service using Common Information Model/Web-Based Enterprise Management (CIM/WBEM).

Figure 11-2 also shows Core Services that provide the following two types of interfaces:

- APIs that can be accessed directly by applications written in the Java™ programming language.
- Web services that can be accessed via SOAP over http(s).

The following Core Services are provided:

- Authentication
- Single sign-on (SSO)
- User personalization or preferences

- Policy Evaluation and Privacy
- Security through PKI and Kerberos
- Logging or audit

### 11.1.1 iPlanet™ Directory Server Products

From the perspective of the Sun ONE platform, the Identity and Policy components are provided by iPlanet Directory Server Access Management Edition and Integration Edition.

These two editions of the directory server are infrastructure products designed to meet the complex and varying needs of Intranets, extranets, and Internet Service Providers (ISPs). They provide the means for unifying identity and service management, as well as for consolidating policy management for increased enterprise and Web security. Their components and services integrate with iPlanet Directory Server, the MetaDirectory, UDDI, Light Weight Directory Access Protocol (LDAP) Proxy, Web Server, Proxy Server, and Certificate Management System to form a platform for managing the extranets, Intranets, and portals. The UDDI registry is built using iPlanet Directory Server and will initially serve the purpose of being a private UDDI registry. It will later be extended to become a public registry.

Both the management and integration editions of the server provide interfaces that will be compliant with the requirements of the Sun ONE architecture and any existing standards. Additionally, the management edition provides proprietary interfaces either in the absence of standard interfaces or when simple and easy-to-use convenience interfaces are needed. Note that these additional interfaces are not mandatory for Sun ONE architecture compliance.

### 11.1.2 Identity: Authentication

The Authentication component provides interfaces that can be accessed by:

- Users via their browser to obtain authentication.
- Web services and applications to authenticate users.

Figure 11-3 shows the architecture of the Authentication component. It uses the Java Authentication and Authorization Service (JAAS) software as its framework, which is part of the Java 2 Platform, Standard Edition (J2SE™) 1.4 specification. Refer to <http://java.sun.com/products/jaas> for more information on JAAS interfaces and their authentication mechanism.



Java Authentication and Authorization Service provides the client authentication APIs, the authentication framework, and plug-in Service Provider Interfaces (SPIs). The authentication APIs can be used to authenticate users by programmers using the Java programming language. The Authentication component also provides the Simple Object Access Protocol (SOAP)-based Web services to authenticate users in a client-service environment and a HyperText Markup Language (HTML)-based user interface (UI) that can be used by Web applications.

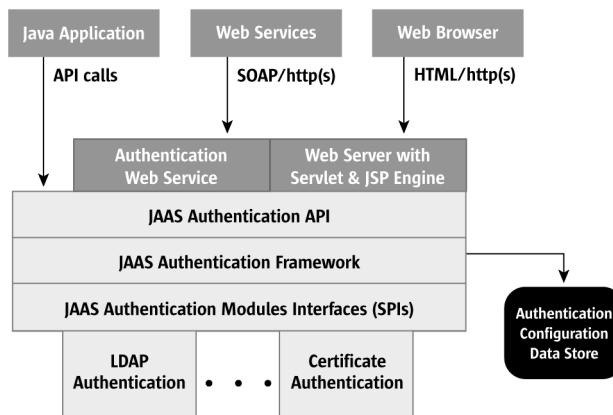


Figure 11-3: Authentication Architecture

The Authentication service, which can be accessed via SOAP, provides constructs to gather and submit user credentials. Hence it provides a means for the client to have a conversation with the Authentication Service. This is a requirement for authentication protocols that require challenge-responses.

At the time of this writing, iPlanet Directory Server and iPlanet Directory Server Access Management Edition support centralized authentication, either through the directory or through the addition of more authentication modules to the management edition's server. The additional authentication modules provided by the management edition are RADIUS, Certificate Based Authentication (Public Key Infrastructure [PKI]), SafeWord, SecureID, UNIX®, NT Domain, and LDAP. Furthermore, iPlanet Directory Server Access Management Edition will provide convenience authentication APIs over Java Authentication and Authorization Service interfaces that will include a synchronous mechanism to obtain callback objects to gather user credentials. Java Authentication and Authorization Service provides an asynchronous mechanism.

iPlanet Directory Server Access Management Edition stores authentication configuration information in iPlanet Directory Server (compared to the files implementation provided by Java Authentication and Authorization Service). Therefore, if there are multiple instances of Authentication Service, the configuration must be changed only once in the directory. For more details on the management edition of the server, refer to <http://www.iplanet.com>.

### 11.1.3 Identity: Web Single Sign-On

The SSO component provides interfaces that can be used by applications and Web services to maintain a user's authenticated session across multiple applications and Web services without having to re-authenticate the user. Figure 11-4 shows the architecture of the SSO component, which basically contains SSO Java technology APIs and a Session Manager. The SSO APIs provide methods for Java programming language applications to establish a SSO session for a user and to build a SSO service that can be accessed by Web services. Additionally, the Authentication service calls the SSO APIs to initially construct a SSO session for an authenticated user. The Session Manager handles SSO session information such as time of authentication, principal name, idle time, and maximum session time.



Figure 11-4: Web SSO Solution Architecture

The SSO component also provides a Session Service that can be accessed by Web services to validate SSO tokens and to obtain session-related information such as idle timeouts and maximum timeout.

As shown in the figure, the architecture enhances the integration between the Authentication component and the SSO component by providing a SSO token to an authenticated user.

From the perspective of the Sun ONE platform, iPlanet Directory Server Access Management Edition provides a secure Web SSO framework by extending *Javax.servlet.http.HttpSession* to allow the user access different Web resources, processes, applications, and services in a single user session without having to authenticate to each one of them. Furthermore, the management edition supports signed exchange of Web SSO tokens and other user credentials.

The Session Manager in the iPlanet Directory Server Access Management Edition provides a pluggable architecture to support different implementations of session managers. The default implementation provided by the management edition is based on *Javax.servlet.http.HttpSession*.

## 11.1.4 Identity: Cross-Domain Single Sign-On

In the previous section, the SSO solution was discussed with respect to a single domain in which the participating Web services and applications have a trust relationship. To enable SSO across multiple domains, a trust relationship across these domains must be negotiated. The Liberty Alliance Project, which is described in Section 11.2.1, will address this issue. It will also provide a protocol specification to accomplish cross-domain SSO. Furthermore, Liberty will address federation of user accounts across multiple domains.

The security services technical committee within the Organization for the Advancement of Structured Information Standards (OASIS) is developing Security Assertion Markup Language (SAML), which also will address cross-domain SSO. The Liberty Alliance Project is expected to endorse the SAML protocol specification. In fact, it may enhance it to support federation of user accounts. For further information regarding SAML, refer to the interface table.

From the perspective of the Sun ONE platform, iPlanet Directory Server Access Management Edition currently supports a proprietary cross-domain Web SSO solution. Future releases, however, will be based on SAML and Liberty specification.

## 11.1.5 Identity Management: User Account Management and Provisioning

Identity management controls the flow of user information between the different systems in an enterprise and extranet. It provides a common way for the administrator to manage user information. Furthermore, it provides interfaces for users to self-register and configure their preferences.

### 11.1.5.1 User Provisioning and Self-Registration

Administrators and account managers can create, delete, and modify user information through Java technology APIs and XML interfaces. The XML interfaces follow the recommendations of the OASIS Provisioning Working Group. For further information regarding these recommendations, see the interface table.

The self-registration feature allows users to provision themselves with user attributes. Some of these attributes are mandatory, while others are optional. For example, the user must provide his or her first name and last name, but the phone number and e-mail attributes may be optional. Also, the user can set up privacy rules regarding his or her information. iPlanet Directory Server Access Management Edition provides GUI and command line tools for the provisioning of users.

Note that the self-registration APIs are optional. At the time of this writing, a new Java Specification Request (JSR) is being submitted to address self-registration. This JSR will be incorporated in a future version of the Sun ONE architecture.

### 11.1.5.2 Profile API

Identity Management provides services that allow a user to customize such attributes as his or her home address and preferred telephone number. Once the user has been authenticated, he or she should be able to add, delete, and modify these attributes, which are collectively called the user's profile. Additionally, administrators should be able to configure the preference attributes for the users and their default values. Java technology APIs will be specified in a future edition of this book to allow applications and services to set and get user preferences on behalf of the user.

### 11.1.5.3 User Organization and Integration

iPlanet Directory Server Access Management Edition provides APIs to set up organizations (domains), sub-organizations (subdomains), and groups, then assign users to them.

iPlanet Directory Server Integration Edition provides a rich framework for the synchronization of user information between disparate systems and the Sun ONE architecture's Identity and Policy Services. Connectors for Windows 2000, Windows NT, Oracle, and other platforms are provided through the MetaDirectory service. This service supports both scheduled and on-change synchronization of user and group data.

iPlanet Directory Server Integration Edition also includes the LDAP proxy service, which supports fine-grained access control to permit and manage controlled access to directory information from Web services and applications located across the enterprise and between organizations. In conjunction with the multi-master replication provided by iPlanet Directory Server, this ensures a secure and highly available directory service supporting user, group, policy, business, and service information.

### 11.1.5.4 Delegated Management

Most aspects of Identity Management are regarded as privileged operations that must be executed by what are generally called "administrator" users. iPlanet Directory Server Access Management Edition supports delegated administration whereby administration of identities can be delegated to appropriate administrators based on their roles or groups. The granularity can extend from entities such as organization, domain, sub-organizations and subdomains to finer distinctions such as roles, groups, and specific services. Note that delegated management is optional for compliance with the Sun ONE architecture.

## 11.1.6 Policy Management and Evaluation

This section describes the Policy component of the Identity and Policy architecture shown in Figure 11-5.

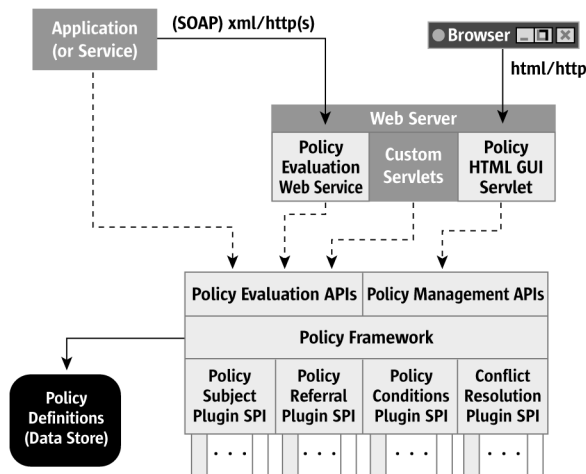


Figure 11-5: Policy Component Architecture

The policy component broadly provides two kinds of interfaces:

- An interface to obtain and set policy definitions as to who can access which resource and the conditions for accessing it.
- An interface to evaluate policies such as whether or not user A can perform action X on resource Y.

The policy component follows the OASIS standards XACML and SAML for policy definition and evaluation respectively. For more information on these standards, see the interface table.

### 11.1.6.1 Policy Framework

The Policy Framework maintains a data store such as a directory server to persistently store policies. It also provides the necessary infrastructure components that allow:

- Applications and services to register policy schema.
- Administrators to setup policies for users and for applications.
- Services to obtain policy evaluation results.

Apart from a rule that specifies the actual authorization assertion on a resource, a policy comprises a subject representing the principal(s) accessing the resource and conditions that describe the circumstances under which the rule operates (for example, only from 1:00 PM to 6:00 PM on Fridays). A policy can be optionally be referred to an external policy evaluation entity, in which case the framework delegates the policy evaluation function to that entity and forwards the policy evaluation response to the client that requested it. Multiple policies for a resource can cause conflicts that can be configured to be resolved based on relative policy priorities and/or by specifying, for example, that a Deny result always will override an Allow result.

Note that iPlanet Directory Server Access Management Edition provides GUIs and command line interfaces (CLI) to administer the policy framework.

### 11.1.6.2 Plug-in SPIs

To accommodate the range of requirements that cannot be addressed via the default providers, the policy framework provides SPIs for each of the policy aspects described in Section 11.1.6.1. This allows service, application, and enterprise security designers to plug in their own implementations by either entirely replacing the default providers or by specifying them on a per resource and/or user characteristics basis.

As depicted in Figure 11–5, the pluggable components are as follows:

- Policy Subject – A subject that identifies the user.
- Policy Referral – Referral to forward the policy evaluation to another component or to an external policy server.
- Policy Conditions – Conditions to enforce constraints on the policy.
- Conflict resolution.

Java technology API interfaces are provided for plug-in SPIs.

### 11.1.6.3 Management and Evaluation APIs

In the area of management and evaluation, both Java technology APIs and Web services are provided.

The Java technology APIs include:

- Java APIs for the creation and management of policies (Policy Definition APIs) based on OASIS XACML standards.
- Java APIs for policy evaluation based on OASIS SAML. Future APIs will be based on JSR 155 specifications.
- J2SE platform support via implementations of the `java.security.Policy` and `javax.security.auth.Policy` abstract classes.
- `java.security.PermissionCollection` objects or the given subject and/or code base package. This is an optional API.

- APIs to construct `java.security.Permission` objects, including permission to enable applications written in the Java programming language to obtain an instance of the `Permission` object and to check permissions. This is an optional convenience API to provide a concrete implementation of the `java.security.Permission` class for managed entities.
- Java 2 Platform, Enterprise Edition (J2EE™ platform) support via JSR 115 “Authorization Service Provider Contract for Containers,” the SPI for supporting J2EE platform roles-based authorization.

The Web service interfaces include:

- Policy Evaluation Web service in line with the SAML specification.
- Policy Definition Schema in line with the OASIS XACML specification.

#### 11.1.6.4 Privacy

Privacy is a user-defined policy that determines who can access information and what can be done with it. Specifically, it defines the policies for collection, dissemination, disclosure, forwarding, storage, and selling of personal information on a per-vendor/merchant basis.

##### 11.1.6.4.1 *Difference Between Policy and Privacy*

In a typical use-case flow, privacy is controlled and managed by the end user. Policy and authorization, on the other hand, are controlled and managed by the service, resource provider, or manager.

This distinction leads to different designs or architectures for not only the delivery of services, but also for their implementation. Typically, in an authorization framework, roles are used to define access control conditions or policy rules. Server/resource providers or managers define these policy rules. End users are then assigned to the appropriate roles for accomplishing roles-based access control or authorization. For scalability and manageability reasons, the use of roles rather than users is encouraged when defining the access control and policy rules.

In a privacy framework, the end user is presented with a list of potential vendors, organizations, and other parties that would like to have access to his or her personal data. Typically, roles are not used in such a context, although categories or groups of involved third parties can be defined by the service provider to make it easier for the end user to define rules for them. Furthermore, support for Policy Enforcement Points (PEP) can be built into client systems such as the IE6.0 browser.

The other concern in privacy is more prevalent. Basically, it involves the ability of the end user to control what a business can do with the personal information that he or she has provided. This includes whether the business can store, forward, disclose, or sell it to other third parties, which may in turn have policies on their ability to further store, forward, disclose, or sell it.

The Liberty Alliance Project, which is described in Section 11.2.1, addresses the important issue of privacy under the federated identity model in which a user's personal data spans across distributed enterprises.

iPlanet Directory Server Access Management Edition supports privacy to protect the public and nonpublic personal information of individuals browsing or transacting over the Internet.

## 11.1.7 Security

The Security services defined in this section are used throughout the infrastructure of the Sun ONE architecture and by developers of Web services and Web applications. The definitions of the relevant APIs, schemas, and protocols are collected here for simplicity of presentation. However, the actual implementations of the services are distributed in various places such as a directory server, the J2SE platform class libraries, or the Sun ONE platform.

Security is key to Sun ONE architecture. Therefore, it must be addressed at all tiers in the Sun ONE platform in the following manner:

- Tier 1 security should be addressed by firewalls, Web Server, Proxy Server, Load Balance, and optionally by IP level security (like IPSec).
- Tier 2 security should be addressed by Application Servers and the network operating system (NOS).
- Tier 3 security should be addressed by applications.

The Sun ONE architecture addresses security at various tiers by adopting current industry standards and practices. It also supports evolving and emerging standards.

### 11.1.7.1 Public Key Infrastructure

A public-key infrastructure (PKI) consists of protocols, services, and standards supporting applications of asymmetric and symmetric key cryptography. PKI provides mechanisms for:

- Encryption and decryption of data.
- Digital signing and verification of the documents.
- Building trust hierarchy based on public keys with the use of Certification Authority (CA) and Registration Authority (RA).

The Sun ONE architecture adopts industry standards such as PKIX and X.509 certificates. It also adopts PKCS, TLS, XML Signature, XML Encryption, and XML Key Managements. For full information on these standards, see the interface table.



The Sun ONE architecture will provide the following security APIs: Java Cryptography present in the J2SE platform, Java Crypto Extensions (JCE) and Generic Security Services (GSS) API present in the J2SE 1.4 specification, and Java Secure Sockets Extension (JSSE) software. Additionally, the Sun ONE architecture will adopt Java technology interfaces for Public Key Cryptography Standards, XML Digital Signature APIs, and XML Digital Encryption APIs. For full information on these security APIs, see the interface table.

From the perspective of the Sun ONE platform, iPlanet Directory Server Access Management Edition provides a Certificate Management System (CMS) that implements a certificate infrastructure for issuing X.509 v3 certificates (CA), registration authority (RA), key recovery manager (KRM), on-line certificate status protocol (OCSP), and support for certificate revocation lists (CRLs). Furthermore, the management edition of the server provides Java Security Services (JSS) (reference the JSS information at <http://www.mozilla.org>) that include Java technology APIs for symmetric and asymmetric key cryptography such as encryption, authentication, tamper detection, and digital signatures.

### 11.1.7.2 Kerberos

The Sun ONE platform provides support for Kerberos, a network authentication protocol that is designed to provide strong authentication for client/server application by using secret-key cryptography.

To authenticate users using Kerberos, the J2SE 1.4 specification provides a Kerberos authentication module that can be used within Java Authentication and Authorization Service, which is discussed in Section 11.1.2. The Solaris™ Operating Environment bundles Kerberos Key Distribution Service (KDC) and issues a Ticket Granting Ticket (TGT).

The iPlanet Directory Server Access Management Edition uses the Java Authentication and Authorization Service Kerberos authentication module to support Kerberos.

### 11.1.8 UDDI

The UDDI specification is an integral part of the Sun ONE architecture. UDDI enables an enterprise to describe its business and services, discover other enterprises that offer desired services, and integrate with these other enterprises.

The iPlanet software implementation uses iPlanet Directory Server for UDDI data storage and access. These functions enable logging and audit.

## 11.1.9 Logging and Audit

Logging facilities allow administrators to track resource usage and errors as well as to diagnose problems. Audit trails are key to system security for tracking such activities as failed and successful user authentications to help diagnose and resolve security threats and to assure nonrepudiation. Logging is done as per the Extended Log format standard defined by the W3C. Java technology APIs for logging are based on J2SE platform Logging JSR 47 specifications. For full information on these resources, see the interface table.

iPlanet Directory Server Access Management Edition supports secure flat file and database logging. Logging can be done on a per-service basis. The default is to log in Extended Log format mode; however, the log format can be customized.

## 11.1.10 Identity and Policy Services Interfaces

This following table summarizes the interfaces that are related to Identity and Policy Services.

Interface Name	Level	Status	Reference *	Comments
JAAS Authentication API	Application	Footnote 1	<a href="http://java.sun.com/products/jaas/index-14.html">http://java.sun.com/products/jaas/index-14.html</a>	Section 11.1
JAAS Authentication SPI	System	Footnote 1	<a href="http://java.sun.com/products/jaas/index-14.html">http://java.sun.com/products/jaas/index-14.html</a>	Section 11.1
Authentication Web Service	Application	Footnote 5		Section 11.1
SSO API	Application	Footnote 4	<a href="http://docs.iplanet.com/docs/manuals/dsame/50/html/prog/contents.htm">http://docs.iplanet.com/docs/manuals/dsame/50/html/prog/contents.htm</a>	Section 11.1.3
SSO Web Service	Application	Footnote 5	<a href="http://www.oasis-open.org/committees/security">http://www.oasis-open.org/committees/security</a>	Section 11.1.3
User Provisioning Schema	Application	Footnote 5	<a href="http://www.oasis-open.org/committees/provision/">http://www.oasis-open.org/committees/provision/</a>	Section 11.1.5.1
Self-Registration API	Application	Footnote 5		Section 11.1.5.1
Profile API	Application	Footnote 5		Section 11.1.5.1
User Organization API	Application	Footnote 5		Section 11.1.5.2
Policy Evaluation API	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/jsr155.jsp">http://www.jcp.org/jsr/detail/jsr155.jsp</a>	OASIS-SAML JSR 155 Section 11.1.5.3
iPlanet Directory Server Access Management Edition Policy Evaluation API	Application	Footnote 5		Section 11.1.6.3
Policy Evaluation Web service	Application	Footnote 3	<a href="http://www.oasis-open.org/committees/security/">http://www.oasis-open.org/committees/security/</a>	OASIS-SAML based. Section 11.1.6.3
Policy Management API	Application	Footnote 5		Section 11.1.6.3

Interface Name	Level	Status	Reference *	Comments
Policy Definition Schema	System	Footnote 3	<a href="http://www.oasis-open.org/committees/xacml">http://www.oasis-open.org/committees/xacml</a>	XACML is the information model for defining policies. Section 11.1.6.3
Policy Subject SPI	System	Footnote 5		Section 11.1.6.2
Policy Referral SPI	System	Footnote 5		Section 11.1.6.2
Policy Conditions SPI	System	Footnote 5		Section 11.1.6.2
Policy Conflict Resolution SPI	System	Footnote 5		Section 11.1.6.2
Security: PKIX and X.509	Application	Footnote 1	<a href="http://www.ietf.org/html.charters/pkix-charter.html">http://www.ietf.org/html.charters/pkix-charter.html</a>	Section 11.1.7.1
Security: PKCS	Application	Footnote 1	<a href="http://www.rsasecurity.com/rsalabs/pkcs">http://www.rsasecurity.com/rsalabs/pkcs</a>	Section 11.1.7.1
Security: TLS	Application	Footnote 1	<a href="http://www.ietf.org/html.charters/tls-charter.html">http://www.ietf.org/html.charters/tls-charter.html</a>	Section 11.1.7.1
Security: XML Signature Schema	Application	Footnote 1	<a href="http://www.w3.org/TR/2002/REC-xmlsig-core-20020212">http://www.w3.org/TR/2002/REC-xmlsig-core-20020212</a>	Section 11.1.7.1
Security: XML Encryption Schema	Application	Footnote 3	<a href="http://www.w3.org/Encryption/2001">http://www.w3.org/Encryption/2001</a>	Section 11.1.7.1
Security: XML Key Management Schema	Application	Footnote 3	<a href="http://www.w3.org/2001/XKMS">http://www.w3.org/2001/XKMS</a>	Section 11.1.7.1
Java Crypto Architecture (JCA) API	Application	Footnote 1	J2SE 1.3+ specification	Section 11.1.7.1
Java Crypto Extension (JCE) API	Application	Footnote 1	J2SE 1.4 specification	Section 11.1.7.1
Generic Security Services (GSS) API	Application	Footnote 1	2SE 1.4 specification	IETF RFC 2854, JSR 72 Section 11.1.7.1
Java Secure Sockets Extension (JSSE) API	Application	Footnote 1	2SE 1.4 specification	Section 11.1.7.1
Public Key Cryptography Standards (PKCS) API	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/74.jsp">http://www.jcp.org/jsr/detail/74.jsp</a>	JSR 74 Section 11.1.7.1
XML Digital Signature API	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/105.jsp">http://www.jcp.org/jsr/detail/105.jsp</a>	JSR 105 Section 11.1.7.1
XML Digital Encryption API	Application	Footnote 3	<a href="http://www.jcp.org/jsr/detail/106.jsp">http://www.jcp.org/jsr/detail/106.jsp</a>	JSR 106 Section 11.1.7.1
UDDI	System	Footnote 1	<a href="http://www.uddi.org/">http://www.uddi.org/</a>	Section 11.1.8
Logging API	System	Footnote 1	<a href="http://www.w3.org/TR/WD-logfile.html">http://www.w3.org/TR/WD-logfile.html</a> <a href="http://java.sun.com/j2se/1.4/docs/guide/util/logging/overview.html">http://java.sun.com/j2se/1.4/docs/guide/util/logging/overview.html</a>	Section 11.1.9

---

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

Footnote 4: This is a published proprietary interface, and support of this interface is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 5: This is an unpublished proprietary interface. A published definition for this interface will be provided in a future version of the Sun ONE architecture.

---

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

## 11.2 Federated Identity Systems

Federated identity systems ensure that the use of critical personal information is managed and distributed by the appropriate parties, rather than a central authority. Federated identity enables the development of federated commerce.

Federated e-commerce allows businesses to maintain ownership of their customer directory, leverage their directory more effectively to conduct affinity marketing, and partner with other Internet properties who extend the overall value proposition to the customer. This also gives businesses the ability to provide their customers or end users with more convenience, choice, and control of their identity. Furthermore, a federated identity model allows businesses or users to manage their own data.

### 11.2.1 Liberty Alliance Project

Many of the core Identity and Policy Services, including authentication, single sign-on, and user personalization, are the subject of standardization efforts being undertaken by the Liberty Alliance Project in order to enable federated identity systems. Specifications for these standards are still under development and are not described in this version of the Sun ONE architecture. Because these standard interfaces will be required in future versions of the architecture, this section provides some background information on Liberty.

The Liberty Alliance Project is a business alliance formed to deliver and support an identity solution for the Internet that enables single sign-on for both consumers and business users in an open, federated way.

The Liberty Alliance has three main objectives:

- Allow individual consumers and businesses to maintain personal information securely.

- Provide a universal open standard for single sign-on with decentralized authentication and open authorization from multiple providers.
- Provide an open standard for network identity spanning all network devices.

For more information, refer to <http://www.projectliberty.org>.

## 11.3 Management Services

The purpose of Management Services is to provide both the architectural framework and the collection of programmatic interfaces needed to manage Web services and system resources throughout the Sun ONE platform. The Management Services included within the Sun ONE architecture allow developers of Web services that integrate into the Sun ONE platform to manage their components using the same methodology and interfaces as that used by Sun in its integrated stack. Management Services provide both the system programming interfaces and the application programming interfaces necessary to implement the management of Web services and hardware resources operating within the Sun ONE platform.

Management interfaces provide a means for implementing both system management *applications* such as management consoles, and system management *services* that can monitor and/or control the hardware and software resources constituting a Sun ONE services environment. Management interfaces are specifically intended to provide:

- A means to instrument a managed component (whether a software service or a hardware resource) so that it can be monitored by a management application or service.
- A means to establish controls related to the managed component that allow a management application or service to affect its behavior. It is important to reiterate that the Sun ONE platform management conception applies both to software resources (that is, services) as well as to hardware resources.

### 11.3.1 The Sun ONE Platform Management Architecture

The Sun ONE platform management architecture, illustrated in Figure 11-6, has a three-tiered model which structures the management implementation into the following levels:

- The Presentation Tier
- The Management Services Tier
- The Agent Tier

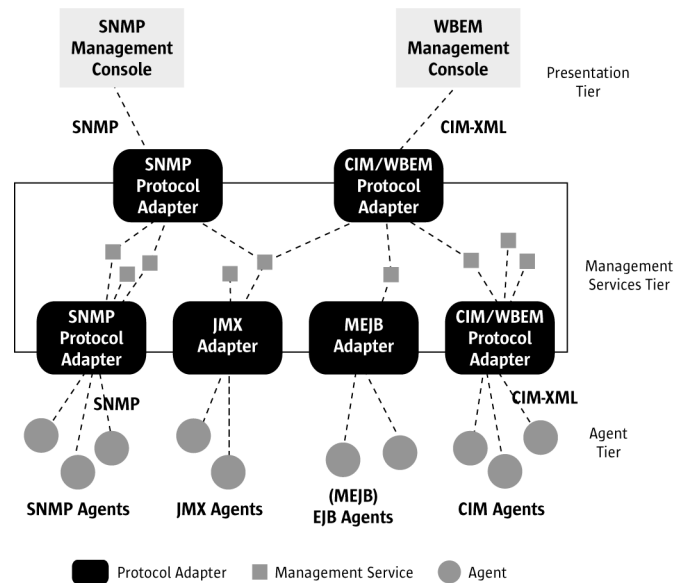


Figure 11-6: System Management Architecture

**The Presentation Tier** – Management applications that interact directly with users run at this highest-level tier. These provide user interfaces and allow user access to the remotely managed objects and/or lower level management services.

**The Management Services Tier** – The primary business logic of a management service can be constructed at this mid-level tier. Management services at this level implement management functionality at a higher level than that offered by individual agents. They may for example, implement a single logical service resulting from the orchestrated management of a number of physically separate resources at the agent level. Alternatively, they may provide for the aggregation and coordinated management of a diverse collection of resources needed to implement the management a given logical service. In other words, services at this level can import the management controls offered by a number of resources at the Agent Tier. Furthermore, they can export the set of higher-level management instrumentation and controls needed to monitor and manage the service abstracted from these individual components.

**The Agent Tier** – Management instrumentation for the remotely manageable components (Sun ONE platform hardware resources and services) runs at this lowest-level tier. An agent is a programmatically accessible entity (whether accessed directly by procedure-call invocation or indirectly via management protocol) associated with a manageable resource. It provides the instrumentation and control interfaces for a single resource, allowing it to be monitored and controlled. At this level, there is one agent for each managed resource.

## 11.3.2 Sun ONE Platform Management Information and CIM Models

The Management Services tier employs the Common Information Model (CIM) from the Distributed Management Taskforce (DMTF) as its single unifying representation and native description methodology for management information. Within the Sun ONE platform management services tier, all management information is represented using CIM models.

One of the goals of the CIM model was to consolidate and extend existing management standards and instrumentation such as that provided by Simple Network Management Protocol (SNMP), Common Management Information Protocol (CMIP), and Java Management Extensions (JMX™). This is accomplished through standard schema, schema-mapping, and protocol adapters.

A collection of adapters is provided by the Sun ONE platform management services tier for mapping to and from other management information representations and protocols at both the Agent Tier and the Presentation Tier. This is described further in Section 11.3.2.3, “Provision of CIM-Based APIs.”

### 11.3.2.1 Use of the CIM Model for All Sun ONE Platform Components

CIM is an implementation-neutral, object-oriented data model for describing overall management information in an enterprise environment. The CIM model, which is comprised of both the CIM Specification and the CIM Schema, is used to model the management information for all Sun ONE platform components.

The CIM Specification defines the meta model, high-level concepts, rules, and a definition language. The CIM definition language is called Managed Object Formation (MOF).

The CIM Schema provide the following set of models for managed resources:

- **Core Model** – This information model incorporates notions applicable to all management domains.

- **Common Model** – This information model incorporates notions common to specific management domains. It does this independently of any particular implementation or technology. At present, the following domains taxonomize the CIM Schema constituting the DMTF Common Model: applications, devices, events, interoperability, metrics, physical, policy, support, system, user, and networks. The information models provided within these domains are the basis for the development of management applications. They include a set of base classes for extension into application- or technology-specific areas.

### 11.3.2.2 Use of Standard Schema to Describe All Sun ONE Platform Components

While the DMTF has defined Core Schema and Common Schema that cover many of the base platform underpinnings of the Sun ONE architecture, new components that are specific to the Sun ONE platform will be defined as compliant extensions to these Core and Common Schema. At present, more than 900 classes and 1800 properties are represented in the DMTF schema. This scope provides a very rich foundation for developers who wish to model additional manageable resources such as new Web services within the Sun ONE platform.

The use of CIM-compliant extensions to model new Sun ONE platform components provides developers with the basis of management for Services on Demand. It also enables the deployment, monitoring, metering, and management of all Sun ONE platform services in a uniform and consistent manner.

Sun ONE architecture schema are a compliant extension of the same DMTF standard used by other base platform vendors. Thus developers who rely on the Sun ONE architecture schema and management service model when developing management, configuration, or performance application tools can expect these tools to interoperate with similarly modeled components on those other platforms such as Microsoft's .NET environment.

### 11.3.2.3 Provision of CIM-Based APIs

Because CIM/XML is the *lingua franca* for the Sun ONE platform management information and operation, a set of native CIM-based APIs enables the development of both management applications and instrumentation providers (agents) for the Sun ONE architecture. These APIs define the standard Java interfaces for communicating with CIM-compliant systems such as the Solaris Operating Environment and the Sun ONE platform over a number of protocols. For a specification that provides full information on these Java interfaces, refer to the interface table.

In the Sun ONE architecture version 1.0, supported protocols include CIM/XML and CIM/Remote Method Invocation (RMI). Note also that these APIs, along with a reference implementation of them, have been placed in Open Source. A C++ version of these APIs has also been developed to enable non-Java management application developers to write applications that can manage CIM-compliant systems.



## 11.3.3 Integration of Existing Management Schemes

Sun ONE describes an architecture for multiplatform deployment. Since there are a number of widely used historical schemes for representing the properties of remotely manageable resources (which will be seen when Sun ONE platform services interact with services running on other platforms), the Sun ONE platform management architecture provides integration with the following predominant existing management technologies:

- Simple Network Management Protocol (SNMP) (Internet Engineering Taskforce [IETF] RFCs)
- Java Management Extensions (JMX)—JSR-3 and JSR-160
- J2EE Platform Management (Java Enterprise Management, MEJBs)—JSR-77
- CIM objects on platforms other than Sun ONE
- Directory-based management information schemes such as Light Weight Directory Access Protocol (LDAP), Domain Name System (DNS), and Network Information Service (NIS)

### 11.3.3.1 Interoperation via Multiple, Pluggable Protocol Adapters

Within the Sun ONE architecture for management, pluggable protocol adapters are the connection components that provide the model and protocol mapping between:

- Applications in the Presentation Tier and services in the Management Services Tier.
- Agents in the Agent Tier and the services in the Management Services Tier.

A Presentation Tier adapter accepts client management requests or queries, translates them into the appropriate CIM-related request, and routes them correctly using the Sun ONE Platform Management Services Tier to the appropriate agent(s). Information from the agent(s) is translated correspondingly by the adapter when returned to the initiating caller. Any necessary modelling and protocol translations also occur in the appropriate Agent Tier adapter.

The Sun ONE platform systems management architecture permits the use of as many protocol adapters as needed. Those defined in the Sun ONE architecture version 1.0 at the Presentation Tier (as illustrated at the top of Figure 11-6) are SNMP, LDAP, CIM/XML (Sun ONE platform *lingua franca*), CIM/SOAP, and CIM/Remote Method Invocation (RMI). At the Agent Tier (as illustrated at the bottom of Figure 11-6), the adapters provided are SNMP, JMX, J2EE Management and CIM-XML (Sun ONE platform *lingua franca*). Note that the Agent Tier also provides the J2EE Platform Management adapter, but this may be supported via JSR-77's defined mapping to CIM-XML rather than a separate specific adapter in the Management Services Tier.

All the CIM adapters fall under the DMTF umbrella name of Web-Based Enterprise Management (WBEM). Each of these protocols has a standard DMTF specification and a test suite to verify its compliance.

The Sun ONE platform management architecture enables other protocol adapters to be added dynamically. The management SDK gives examples of how to develop and register new protocol adapters. Information relevant to the WBEM architecture and services specification, which encompass Sun ONE platform adapters, is given in the interface table in Section 11.3.4, “Management Interfaces.”

### 11.3.3.2 Use of Standard CIM Mappings to Existing Schema

The richness of the CIM model allows it to be used as a base model from which and to which the developer can map existing schema from other management information description methods. At present, such information is modeled using directory services (LDAP, NIS, and DNS), SNMP, and Java technology.

At the time of this writing, the DMTF has defined explicit mappings to LDAP for the Core Schema, as well as to some of the Common Schema. Eventually all the DMTF schema will be mapped into LDAP. This will enable interoperability with existing configuration and management tools that are currently directory-focused.

Similar mappings have been done for many SNMP Management Information Bases (MIBs). A standard is being defined for many of the important ones in popular use. These mappings will be registered with the DMTF to ensure interoperability between existing SNMP client applications and new CIM-modeled environments such as the Sun ONE architecture. This enables existing management application and/or agent vendors to transition their LDAP- and SNMP-based applications over time to XML applications using the CIM model.

### 11.3.3.3 SNMP

Many existing management applications and agents operate on management information described using the methodology described in the IETF’s System and Network Management Protocol (SNMP). The Sun ONE management architecture provides two protocol translators for SNMP—one at the Agent Tier and one at the Presentation Tier—as the means to map management information modeled in SNMP MIBs and transmitted using the SNMP message protocol to information modeled with CIM. The relevant standards and interfaces are identified in the interface table in Section 11.3.4, “Management Interfaces.”

### 11.3.3.4 Java™ Management Extensions (JMX™)

The Java Management Extensions define an architecture, the design patterns, the APIs, and the services for application and network management in the Java programming language.

The JMX specification provides Java technology developers across all industries with the means to instrument Java code, create smart Java agents, implement distributed management middleware and managers, and smoothly integrate these solutions into existing management systems. In addition, the JMX specification is referenced by a number of Java APIs for existing standard management technologies.

The relevant standards and interfaces are identified in the interface table in Section 11.3.4, “Management Interfaces.”

### 11.3.3.5 J2EE Platform Management

The J2EE Platform Management Specification defines a management information model for the J2EE platform, the J2EE Platform Management Model. The specification also includes standard mappings of the model to:

- The CIM model
- An SNMP MIB
- A Java API as a server-resident Enterprise JavaBeans™ (EJB™) component
- The J2EE Platform Management EJB (MEJB) component

The MEJB component provides interoperable remote access to the model from any standard J2EE application. The relevant standards and interfaces are identified in the interface table in Section 11.3.4, “Management Interfaces.”

## 11.3.4 Management Interfaces

The following table lists the requirements for the Sun ONE architecture conformance for management interfaces.

Interface Name	Level	Status	Reference *	Comments
WBEM Architecture	System	Footnote 3	<a href="http://www.dmtf.org">www.dmtf.org</a>	DMTF Section 11.3.3.1 See “Web-Based Enterprise Management” at the dmtf web site
WBEM Services Spec	System	Footnote 3	<a href="http://www.jcp.org/jsr/detail/48.jsp">www.jcp.org/jsr/detail/48.jsp</a>	JCP Section 11.3.3.1
SNMP Architecture	System	Footnote 1	RFC 2571	IETF Section 11.3.3.3
SNMP v1	Protocol	Footnote 1	Standard 15: RFC 1157	IETF Section 11.3.3.3
SNMP Operations v1		Footnote 1	Standard 15: RFC 1157	IETF Section 11.3.3.3
Structure of Management information (SMI v1)	System	Footnote 1	Standard 16: RFC 1155, RFC 1212, RFC 1215,	IETF Section 11.3.3.3
Structure of Management information (SMI v2)	System	Footnote 1	Standard 58: RFC 2578, RFC 2579, RFC 2580	IETF Section 11.3.3.3
SNMP v3	Protocol	Footnote 1	RFC 1906, RFC 2572, RFC 2574	IETF Section 11.3.3.3
SNMP Operations v2		Footnote 3	RFC 1905	IETF Section 11.3.3.3
Fundamental Mgmt Apps		Footnote 1	RFC 2573, RFC 2575	IETF Section 11.3.3.3

Interface Name	Level	Status	Reference *	Comments
JMX 1.0	System	Footnote 1	<a href="http://www.jcp.org/jsr/detail./3.jsp">www.jcp.org/jsr/detail./3.jsp</a>	JCP Section 11.3.3.4
JMX 1.5	System	Footnote 3	<a href="http://www.jcp.org/jsr/detail./60.jsp">www.jcp.org/jsr/detail./60.jsp</a>	JCP Section 11.3.3.4
J2EE Management 1.0	System	Footnote 3	<a href="http://www.jcp.org/jsr/detail./77.jsp">www.jcp.org/jsr/detail./77.jsp</a>	JCP Section 11.3.3.5

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.

## Platform Services

---

The purpose of Platform Services—the interface at the lowest level of the Sun™ Open Net Environment (Sun ONE) architecture—is to provide the functions needed to allocate and manage the resources of the underlying network and hardware platform required to host the higher-level service layers in the Sun ONE platform.

Platform Services' primary focus is to provide the system programming interfaces to the platform's basic functionality that can be used by Web service execution middleware and other lower-level Web services. It is important to note, however, that a product implementing the Platform Services interface may also offer certain facilities—such as printing directly—as Web services for use by higher-level Web services and Web applications.

Platform Services provide the following basic functions:

- Management of the hardware platform and resources.
- Network connection and data transport interfaces for Internet access, such as Internet Protocol versions 4 and 6 (IPv4 and IPv6), and Transmission Control Protocol (TCP).
- Filing and data access through interfaces such as Network File System (NFS), File Transfer Protocol (FTP), and Simple Mail Transfer Protocol (SMTP, or Sendmail).
- Low-level registry, directory, and name services such as Light Weight Directory Access Protocol (LDAP), and Domain Name System/Berkeley Internet Name Daemon (DNS/BIND).
- Security facilities such as Kerberos, ssh (secure shell), and IPsec (a collection of IP security measures that comprise an optional tunneling protocol for IPv6).
- Printing services.
- System management and monitoring interfaces, as described in Chapter 11, “Identity and Policy Services.”

One important aspect of the Sun ONE architecture is that the key layers can be hosted on a variety of operating systems and network platforms. These key layers include Service Delivery, Core Web Services, the Service Container, and Service Integration, as shown in Figure 2–1 in Chapter 2. As one such complete implementation of the Sun ONE architecture, a fully integrated Sun ONE platform is hosted on Sun’s Solaris™ Operating Environment (“Solaris™ OE”) and SPARC® technology platform. This provides both Web service developers and end-user customers the opportunity to buy a complete implementation from Sun if they desire. It also provides an implementation of the Sun ONE architecture that is particularly well suited to enterprise-scale and commercial-grade deployments. As a Platform Services Layer implementation for the Sun ONE platform, Solaris OE provides many security, and reliability, availability, and serviceability (RAS) as well as scalability features, as well as various performance-focused capabilities that can be exploited by the Service Container and other layers in the Sun ONE platform that it supports.

## 12.1 Hardware Platform and Resource Management

Developers of higher-level services in the Sun ONE platform have the following goals:

- Produce a service that scales to hardware platforms suitable for enterprise or commercial deployment.
- Develop highly available Web services.
- Increase the reliability of the service and its execution environment.
- Ensure known performance levels for the higher-level services.

The following sections discuss some specific platform-level interfaces that the Solaris Operating Environment provides to help developers meet these goals.

### 12.1.1 Scaling and RAS

To allow developers to offer enterprise-level commercial scalability and make services highly available, the Solaris Operating Environment provides a number of interfaces that allow Web services and Web-service execution environments to run on configurations that exploit multiple hardware nodes. This “horizontal” scalability can be exploited by a Web service container implementation and by higher level Web service implementations to achieve both greater total platform capacity and performance-range scaling of the underlying hardware supporting them. Likewise, this type of scalability can be used to obtain greater overall system reliability in the face of hardware or software failures on the platform.

The horizontal-scalability function is partially provided by interfaces that support cluster and grid computing hardware configurations. These interfaces are described in the sections that follow.

### 12.1.1.1 Interfaces to Exploit Cluster Configurations

The Solaris ClusterOS software enables Web-services system software (such as a portal server, application server, directory server, messaging service, or integration server) to run on and exploit a hardware cluster configuration. In addition to providing the means to make Web services highly available, the Cluster Interface Set offers the ability to increase overall system performance via scaling the platform to run on multiple hardware nodes. Services implemented to exploit these facilities allow the service to remain available for both planned and unplanned downtime. Access to back-end data used by Web services (such as those facilities afforded by the Sun ONE Architecture Service Integration Layer) can also be made highly available as a side effect of such configurations.

In addition, the Cluster Interface Set provides a means to establish the configuration of system software. This configuration includes the dependencies between components and the issue of how, when, and where they get started—both initially and at times when applications or services must be restarted during a failover event.

#### 12.1.1.1.1 *Service and Application Failover*

The basic consideration in implementing highly available services on a cluster is the failover of applications or services running on the platform—that is, the ability of the platform software to restart these applications or services and/or other components of system software when either a hardware or software failure occurs on the cluster.

The Cluster Interface Set allows system software running on a cluster to register with the cluster framework and then receive callbacks that reflect any major life-cycle event on the cluster. These events allow the system software to recognize and respond to cluster change circumstances.

The Solaris ClusterOS™ software Resource Group Manager (RGM) provides the environment to establish highly available and scalable software services. The benefits of the cluster RGM include:

- Easy management and monitoring of “agents” (applications or software services implemented to be highly available).
- End-to-end control of service or application failover and scalable agents.
- Ease of use.

The elements that form the programming interface to the The Solaris ClusterOS software RGM facility include:

- A set of callback methods that the cluster RGM uses to control an application on the cluster.
- APIs, commands, and functions that callback methods use to access information about the elements in the cluster.
- Process management facilities for monitoring and restarting processes on the cluster.

### 12.1.1.1.2 *High-speed Communication, Synchronization, and Checkpointing*

The Cluster Interface Set includes interfaces that allow a software service or application running across multiple nodes to commit session-level or other persistent state to a reliable, fast-shared store such as shared memory accessible over high-speed interconnects. A set of low-level interfaces called Remote Shared Memory (RSM) allows system software to exploit remote shared memory. This facility also may be used to perform high-performance checkpointing, inter-process, or inter-service synchronization.

RSM provides specific support for those cluster operations that require shared or persistent state. The RSM library functions support the following:

- Memory segment operations, including both segment management and data access
- Interconnect controller operations
- Cluster topology operations
- Cluster event operations

## 12.1.2 Solaris™ Resource Manager

In order to address certain quality-of-service considerations for services running at higher levels of the Sun ONE architecture, the Solaris Operating Environment provides some basic facilities for the commitment of hardware and software resources. One of these facilities is Solaris™ Resource Manager software.

The primary goal of the resource manager, available in the Solaris 9 Operating Environment, is to improve the ability of both system administrators, and of the individual services running on the platform, to manage the fundamental resources that the platform makes available to them. These resources include CPU, physical and virtual memory, network bandwidth, i/o bandwidth, shared memory and synchronization objects, and interrupt bindings.

To manage these resources, Solaris Resource Manager software provides a number of both programming and administrative interfaces that enable server consolidation—that is, the ability to run multiple workloads with different resource needs and priorities on a single server while still controlling the resources allocated to the individual applications that constitute those workloads. SRM offers a means to commit performance- or capacity-related resources to each workload at a finer granularity than the commitment of the full resources of a hardware server.

Solaris Resource Manager software's focus on performance and the scalability of the platform is somewhat analogous to that of the Cluster Interface Set. However, SRM concentrates more specifically on the problem of *provisioning* the services or applications running on a shared system with the necessary resources. SRM uses the concepts of *processes*, *tasks*, and *projects* as the basis for structured aggregation of the executing software that represents a *workload* such as a service or application. These concepts are also the basis of SRM's resource control.



Solaris Resource Manager software introduces the concept of a resource pool as the collection of resources reserved for the exclusive use of a given workload (service or application). Pools allow a server platform to be partitioned into a number of virtual machines. Each virtual machine provides the resources for a single workload consisting of one or more executables. The resource partitions represented by separate virtual machines provide fixed boundaries between workloads, giving each workload the resources it requires, regardless of resource contention on the rest of the physical machine.

Solaris Resource Manager software functionality complements that of the Clustering functionality insofar as it provides a mechanism to allocate the resources on the network and hardware platform. The resource management concepts offered by the SRM model are independent of whether the underlying system configuration consists of one or multiple hardware nodes.

## 12.2 Networking

The Sun ONE Platform Services offer a number of facilities for basic transport-level access to the Internet, including:

- HyperText Transfer Protocol (HTTP)
- Internet Protocol (IP): IPv4 and IPv6, and the Internet protocols stack such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP)

### 12.2.1 Networking in the Solaris Operating Environment

The basic interfaces for network data transport are standard parts of most contemporary operating system products that implement the hardware and network platform. However, certain aspects of the operating environment's implementation afford greater performance scalability and reliability to Web services and Web-service- execution environments.

One example of network reliability and scalability in the Solaris Operating Environment is IP-multipathing. This implementation permits data transport using IP to be run transparently over multiple network hardware interfaces to gain greater reliability. IP-multipathing provides for failover in the event of a network interface outage or failure.

A second example of performance acceleration in Solaris OE networking is the Network Cache Accelerator (NCA). NCA is a Solaris OE kernel module designed to provide improved Web server performance.

The NCA kernel module, which services all HTTP requests, maintains an in-kernel cache of Web pages that it examines when servicing an HTTP request. If the NCA kernel module cannot service the request itself (directly from the cache), it then passes the request to an appropriate user-level HTTP server such as `httpd`. The cache can improve Web server performance significantly. An associated logging facility, `ncalogd(1)`, logs all requests and may be used to monitor, analyze, and tune system performance.

## 12.3 Storage, Filing, and Data Access

The Sun ONE Platform Services may offer a number of core facilities for basic access to data, as well as to directly attached and network-attached storage and filing systems. These are optional, since access to storage, files, and data by Web services executing on a given platform can be achieved through the access to Web services that provide those facilities but run on another platform. However, since at least *some* platforms must provide such facilities, the optional interfaces for these facilities in Platform Services for this version of the Sun ONE architecture include FTP, SMTP/Sendmail, and NFS.

### 12.3.1 Solaris Operating Environment Storage, Filing, and Data Access

As with networking, many of the basic interfaces for filing, data access and storage are standard parts of most contemporary operating system products that implement the hardware and network platform. In this area too, however, certain aspects of the Solaris Operating Environment implementation afford greater performance scalability and reliability to Web services and Web services execution environments.

The Solaris Volume Manager software provides a means to make storage more scalable and reliable, and to increase data-access performance. Solaris Volume Manager software implements facilities for disk concatenation (logical volumes), striping, and mirroring. This supports the implementation of storage fault tolerance and performance enhancement via hot-spare diskpools, multi-homed disks, and various RAID storage configurations, as well as the scaling of storage capacity via the attachment of additional disks to a logical volume.

## 12.4 Name Service, Directory, and Registry Functions

The Sun ONE architecture's focus on Web services means that registries and directories to advertise and locate services (particularly UDDI as addressed in earlier chapters), are a primary requirement. However, some implementations of such directories, as well as implementations of other Web services may want to use lower-level directory facilities.

The Sun ONE architecture therefore allows certain interfaces for basic name service, directory, and registry functions to be provided by the Platform Services Layer. In version 1.0 of the Sun ONE architecture these optional facilities include LDAP and DNS/BIND.

## 12.4.1 Naming, Registry, and Directory Services in the Solaris Operating Environment

The Solaris Operating Environment provides an implementation of both version 1.0 of the Sun ONE architecture optional low-level directory services discussed above, as well as NIS (Network Information Service), which is another widely used de-facto standard.

As for several other basic platform facilities, Solaris OE provides cache-based acceleration for several of its name-service, directory and registry-service facilities. One example of this is the Name Service Cache daemon (nscd), which accelerates many of the fundamental name service lookups used for network-based operation such as hostname to IP-address mapping and username to user-id.

## 12.5 Security

Security considerations must be taken into account when developing both Web services and the broader execution environment for Web services such as the Service Container, Directory servers, and Web servers. Developers who work in these areas must ask questions such as:

- How will users of Web applications and individual lower-level Web services be authenticated?
- How will services ensure that they are operating over known, secure channels of communication?
- How will the privacy of transmitted information be realized?

Comprehensive security in the Sun ONE platform requires attention at each level of the architecture. However, Platform Services provides a set of basic facilities that serve as the foundation for security. Using these facilities, developers of Web services execution middleware and implementors of Liberty services (to enable open single sign-on) can:

- Provide the ability to implement secure communication over the network.
- Ensure authenticity of peer servers and services.
- Encrypt and sign data for privacy and integrity.

Several of the security facilities provided by the Solaris Operating Environment's implementation of Platform Services are described in the sections that follow.

### 12.5.1 Authentication

The Solaris Operating Environment provides the following two standard interface sets that may be used by Web-service system software and Web services to implement security:

**The GSS-API** – This Generic Security Services interface consists of a security framework and a set of interfaces that enable services or applications to protect the data that they transmit.

**Kerberos** – A client/server architecture that offers strong user authentication, along with data integrity and privacy, in order to provide secure transactions over networks.

In the Solaris Operating Environment, the Sun Enterprise Authentication Mechanism (SEAM) is the Kerberos implementation. More specifically, SEAM implements the Kerberos version 5 network authentication protocol described below. It also provides an administrative framework for it.

The Kerberos network-based authentication protocol is designed to provide strong authentication for client/server applications by using secret-key cryptography. The protocol implementation is available free from Massachusetts Institute of Technology, as well as from other vendors. Services and applications can be “Kerberized” to take advantage of a system running the Kerberos server, a security framework that enables applications to protect the data that they transmit.

In the Solaris Operating Environment, a number of pre-Kerberized services already exist, including utilities such as file transfer protocol (ftp), remote copy (rcp), remote login (rlogin), remote shell (rsh), and telnet. An introduction to the Kerberos system is described in [krb\_man] Kerberos API: the Solaris OE manual pages: kerberos(1) and manual pages in section (3krb). Sun Microsystems Inc., which are available via <http://docs.sun.com>.

## 12.5.2 Secure Internet Transport

Developers of Web-service-execution middleware and Liberty services require secure transport of data over the Internet. IPsec provides a standard low-level means of ensuring authenticity and confidentiality (privacy) of data transmitted using the IP as a result of the encryption interface and certificate management framework that it provides. Internet Key Exchange (IKE) is an automated key management standard for IPsec.

## 12.5.3 Strong Random Numbers

Secure protocols and services provided in the Sun ONE platform all ultimately rely upon high-strength random numbers in order to provide effective cryptographic strength to the security feature they offer. The random-number-generation facility is a particularly low-level system programming interface. A Web service developer leveraging the Sun ONE platform will most often rely indirectly on the random number generation facility by using other Web services that provide a secure facility. However, in cases in which it is appropriate, the Solaris random number interface may be used directly by a Web service system software developer.

Sun's Solaris OE includes the implementation of a strong-random-number generator. Several of the other Solaris OE software services, including Kerberos, IPsec, Secure Socket Layer (SSL), ssh, and Public Key Infrastructure (PKI) take advantage of this facility.

## 12.6 Interfaces

Interface Name	Level	Status	Reference *	Comments
Cluster Resource Group Management Interfaces	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.572.9">http://docs.sun.com/ab2/coll.572.9</a>	"Sun Cluster 3.0 Concepts", "Sun Cluster 3.0 Data Services Developer's Guide", and "Cluster Resource Management API" - man pages in sections (1HA) and (3HA).
Cluster RGM Admin Cmds & Interfaces	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.573.9">http://docs.sun.com/ab2/coll.573.9</a>	"Sun Cluster 3.0 Data Services Installation and Configuration Guide"
Remote Shared Memory	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pages: RSM API
Solaris Resource Management (SRM): Extended Accounting	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pages: libexacct(3lib)
SRM: Tasks and Projects	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pgs: libproject(3lib)
SRM: Resource Pools	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pgs: llibpool(3lib)
SRM: Admin Intfcs	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.47.13">http://docs.sun.com/ab2/coll.47.13</a>	Solaris 9 Operating Environment System Admin Guide: Resource Mgmt and Netwk Svcs
IPv4	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc0791.txt">http://www.ietf.org/rfc/rfc0791.txt</a>	IETF RFC 791
IPv6	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc2460.txt">http://www.ietf.org/rfc/rfc2460.txt</a>	IETF RFC 2460

Interface Name	Level	Status	Reference *	Comments
HTTP 1.1	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc2616.txt">http://www.ietf.org/rfc/rfc2616.txt</a>	IETF RFC 2616
Solaris Volume Manager	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.260.2">http://docs.sun.com/ab2/coll.260.2</a>	DiskSuite 4.1.2
FTP	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc0959.txt">http://www.ietf.org/rfc/rfc0959.txt</a>	IETF RFC 959
SMTP	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc0821.txt">http://www.ietf.org/rfc/rfc0821.txt</a>	IETF RFC 821
NFS v.4	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc3010.txt">http://www.ietf.org/rfc/rfc3010.txt</a>	IETF RFC 3010
LDAP v.3	Protocol	Footnote 2	<a href="http://www.ietf.org/rfc/rfc2251.txt">http://www.ietf.org/rfc/rfc2251.txt</a>	IETF RFC 2251 .. RFC 2256
LDAP: APIs	Protocol	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pgs: llibdapl(3lib)
LDAP: Admin Interfaces	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.47.13">http://docs.sun.com/ab2/coll.47.13</a>	Solaris 9 Operating Environment System Admin Guide: Naming and Directory Svcs (DNS, NIS and LDAP)
GSS-API	System	Footnote 1	<a href="http://www.ietf.org/rfc/rfc2744.txt">http://www.ietf.org/rfc/rfc2744.txt</a>	IETF RFC 2744
Kerberos v.5	Protocol	Footnote 1	<a href="http://www.rfc-editor.org">http://www.rfc-editor.org</a>	IETF RFC 1510
Solaris Enterprise Security	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.384.2/SEAM">http://docs.sun.com/ab2/coll.384.2/SEAM</a>	Sun Enterprise Authentication Mechanism 1.0.1
IPsec	Protocol	Footnote 1	<a href="http://www.ietf.org/rfc/rfc2401.txt">http://www.ietf.org/rfc/rfc2401.txt</a>	IETF RFC 2401
IKE	Protocol	Footnote 2	<a href="http://www.ietf.org/rfc/rfc2409.txt">http://www.ietf.org/rfc/rfc2409.txt</a>	IETF RFC 2409
Random numbers	Protocol	Footnote 1	<a href="http://www.rfc-editor.org">http://www.rfc-editor.org</a>	IETF RFC 1750
Random number API	System	Footnote 4	<a href="http://docs.sun.com/ab2/coll.40.7">http://docs.sun.com/ab2/coll.40.7</a>	Solaris 9 Operating Environment man pgs: random(3c)

#### Table Footnote Legend

Footnote 1: This interface is a standard, and support of this standard is required for products conforming to v1.0 of the Sun ONE architecture.

Footnote 2: This interface is a standard, but support of this standard is not required for products conforming to v1.0 of the Sun ONE architecture. Support of this standard will be required in a future version of the architecture.

Footnote 3: A standard interface is being developed for this component, and that standard will be required in a future version of the Sun ONE architecture.

Footnote 4: This is a published proprietary interface, and support of this interface is required for products conforming to v1.0 of the Sun ONE architecture.

\*This table contains url's to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.





## Part 7. Core Web Services

---



## Core Web Services

---

The Sun™ Open Net Environment (Sun ONE) architecture defines several Web services called Core Web Services. After they are developed, these services will make the functionality included in the Sun ONE platform and in existing Sun Web applications available for use by other Web services and Web applications.

These Core Web Services can be thought of as built-in Web services that will offer developers a set of useful system-level and application-level services. At the time of this writing, however, they have not been fully defined. Therefore, no interface specifications are cited in this version of the Sun ONE architecture, and no interface tables are contained in this chapter. Instead this chapter provides an indication of the kinds of Core Web Services that will be specified in a future version of the architecture.

Note that, with the emergence of federated identity services such as the Liberty Alliance, many of the Core Web Services outlined here are likely to form the basis for future Federated Services. Such Federated Services will be offered by large service providers to furnish ubiquitous access to these communications and information services, thus increasing the utility of Services on Demand, particularly for users of mobile computing devices.

### 13.1 Portal-based Web Services

The following definitions of Web services are based on the functionality of the iPlanet™ Portal Server, which is described in Chapter 9, “The Portal Server.” In that chapter, the Location, Presence, Notification, and Usage Web services described below are briefly discussed as System Level Interfaces (SPIs) to support personalization. The following sections provide more detailed descriptions of how these SPIs will perform when they are fully developed as portal-based Web services.

Note that the portal-based Web services described below will be defined in more detail in a future edition of this book.

### 13.1.1 Location Web Service

The Location Web service will provide geographic location information to the client. To locate a user, the service will interact with a wide range of applications and services from the portal infrastructure—for example, a wireless module, an instant-messaging module, a badge reader system, or a desktop module.

Location-based services such as driving directions will be prime users of the Location Web service. Many future providers and services will be location-sensitive.

### 13.1.2 Presence Web Service

The Presence Web service will provide comprehensive information on the current context of the user, including the manner in which he or she is connected to the system. This service will also indicate the user's geographical location, availability, and type of applications in use. It will use location, user, calendar, and other data to infer all Presence information, including location, status of the device being used, capabilities of the device, availability of the user, activity of the user, communication address of the user, and device preference.

Presence information will be categorized into different access levels. Client applications using the Presence Web service will have access only to Presence attributes authorized for their access level.

### 13.1.3 Notification Web Service

The Notification Web service will provide a notification mechanism that takes advantage of the user profile information and current user context available to the portal. Using this information, the service will deliver messages to users in the most efficient and convenient manner possible.

For example, applications and services such as a calendar can use the Notification Web service to notify a user about events that have been added to his or her schedule or that have been set by other applications. The service—using the user profile information, the urgency of the message, and the data obtained from the Location and Presence Web services—will notify the user by voice mail, e-mail, pager, notification channel, and other means.

Applications and services must have knowledge only of the Notification Web service and its relatively simple interface. The service determines the best way to reach the user and uses the appropriate mechanism to do so—for example, notification via instant messaging, a wireless module, or a channel in the user's desktop.

## 13.1.4 Usage Web Service

The Usage Web service will collect utilization information from the portal. This information will be available to other services for planning, statistical, and billing purposes.

The portal will provide, as part of its infrastructure, a mechanism to plug in Usage Collectors and associate Usage Collectors to the Web services being aggregated in the portal. A Usage Collector will pass usage records to the Usage Web service, which will then pass them to subscribed applications and services.

This model will allow the implementation of Usage Collectors that can introspect the payload of a message to gather usage information or any other type of usage monitoring without having to modify the portal. Out-of-the-box Usage Collectors may be provided to meter the number of hits, determine the beginning and end of a session, and collect common information for logging and auditing purposes.

## 13.1.5 Search Web Service

A future edition of this book may define a Search Web service based on the iPlanet Portal Server Search Engine. This service would allow a client to specify search criteria based on which references to matching query documents are returned.

## 13.1.6 File Web Service

A future edition of this book might define a File Web service based on the iPlanet Portal Server Netfile component. This service would allow a client to upload, download, and move files among different file systems.

## 13.2 Communications Applications Web Services

This section describes a set of Web services that will expose the main functionality of many of the iPlanet software communications applications, including the main applications of mail, calendar, address book, and conferencing. Developers can easily construct a Web application or Web service that incorporates communications functionality using these Web services.

Note that the iPlanet communications applications Web services described below will be defined in more detail in a future edition of this book. For further information, see <http://www.iplanet.com>.

## 13.2.1 Mail Web Service

The ability to read and send e-mail is traditionally implemented via the Post Office Protocol (POP), Internet Message Access Protocol (IMAP), and Simple Mail Transfer Protocol (SMTP) standards. In the Web services model, these protocols may be layered with a Simple Object Access Protocol/eXtensible Markup Language (SOAP/XML) interface that will allow a general client, not just a dedicated e-mail client, to access information that will be provided as XML data. This will allow the exposure of the following functions as Web services:

- Return of a counter of unread messages
- Provision of a list of unread messages
- Reading of an individual message
- Reading, processing, and saving of message attachments
- Creation of a new message

Beyond these basic services, folder and message management features may also be provided. Using these features, an application would be able to list, add, remove, and move folders, as well as move and delete messages.

In addition, a message store facility may be provided. Beyond its capability an e-mail message store, it could be used as a general-purpose data store. Using this facility, an application would be able to create, delete, copy, or move items.

Finally, a mailing list facility may be provided to deal with distribution lists. This facility would allow an application to create and delete a list, then add, remove, or query a specific list.

## 13.2.2 Calendar Web Service

The iPlanet Calendar Server provides key services relating to scheduling people, resources, and events. Web services will be defined to expose calendar functionality for creating, listing, and removing appointments.

## 13.2.3 Contacts Web Service

The iPlanet Contacts Server provides access to an end user's contact information, which is stored in a repository. This information includes the user's name, address, phone number, and e-mail address. The contacts core functionality will be exposed as a Web service to allow an application to add, remove, modify, and query both contacts and groups.

## 13.2.4 Conferencing Web Services

Conferencing Web services will provide the key functionalities for initiating real-time communications. Three functional components—chat, news, and conference—will be exposed as Web services. The Chat Web service will allow applications to post and read messages. The News Web service will allow applications to list and create news channels, as well as to add and retrieve messages. The Conference Web service will allow applications to list and create conferences, as well as to add messages to and retrieve messages from a conference.

---

For definitions of the acronyms and technical terms used in this and other chapters, see the *Glossary* at the end of this book.

For supporting references regarding the topics discussed in this and other chapters, see the *Bibliography* that follows the *Glossary*.





# Glossary

---

TERM	DEFINITION
asynchron-ous reliable messaging	A messaging system that allows an application to interact with other applications using a local message queue, regardless of whether the remote application is actually available when the first application initiates the interaction. The message is delivered by the reliable messaging provider when the service at the other end is receiving. The first application does not need to maintain an open thread waiting for the remote application to receive the message.
BIND	BIND (Berkeley Internet Name Daemon) is software that allows a user to type site names such as <a href="http://www.sun.com">www.sun.com</a> , to connect to an IP address, instead of a string of numbers.
BPSS	BPSS (Business Process Specification Schema) specification defines Business Signals as application level documents that 'signal' the current state of the business transaction. These business signals have specific business purpose and are separate from lower protocol and transport signals. Business signals include receipt acknowledgements, acceptance acknowledgements and exceptions.
cHTML	A mark-up language used in cell phones and PDAs.
CIM	CIM (Common Information Model) is the DMTF (Desktop Management Task Force) model for describing management information to work with disparate systems.
CIM-SOAP	A management protocol based on using SOAP to transmit CIM information.
CORBA	CORBA (Common Object Request Broker Architecture) provides for standard, object-oriented interfaces between ORBs, allowing for the interoperability of object-oriented software systems residing on disparate platforms.
Core Web Services	A set of built-in system-level and application-level Web services that Sun will offer developers. When Core Web services are fully defined, they will include Portal-based services such as Presence, Notification, and Usage. They will also include communications-application-based services such as Mail, Contacts, and Conferencing.
DHCP	DHCP (Dynamic Host Configuration Protocol) is a TCP/IP protocol that enables PCs and workstations to get temporary or permanent IP addresses from centrally administered servers.
DOM	Document Object Model is a platform- and language-neutral interface that allows programs and scripts to access and update the content, structure and style of documents dynamically.
DSML	DSML is a markup language for representing directory services in XML. DSML helps XML-based applications make better use of directories. With a recognized standard, applications can be written to make use of DSML and capture the scalability, replication, security and management strengths of directory services.
ebXML	A complete B2B framework, Electronic Business eXtensible Markup Language ebXML enables business collaboration through the sharing of Web-based business services. B2B business processes re expressed as a sequence of business service exchanges. Included are specifications for Message Service, Collaborative Partner Agreements, Core Components, Business Process Methodology and Registry and Repository.

TERM	DEFINITION
EDI	EDI (Electronic Data Interchange) is a series of standards that provide computer-to-computer exchange of business documents between different companies' computers over phone lines and the Internet.
EJB	EJB (Enterprise JavaBeans) provides a standard component architecture for building distributed, object oriented business applications. EJBs allow a programmer to focus on business logic without having to manage the details of transaction processing, security, load balancing, connection pooling, and other performance concerns in an application server system.
ESMTP	Extended Simple Mail Transport Protocol used primarily in the Unix community. Refer to SMTP.
Fibre Channel	Fibre Channel refers to a set of standards developed by ANSI that are intended to provide a practical and inexpensive means of rapidly transferring data between workstations, mainframes, supercomputers, desktop computers, storage devices, displays and other peripherals.
Forte Development tools	Based on the NetBeans platform, Sun's Forte IDE provides the tools necessary to create, assemble, and deploy Sun ONE applications and Services on Demand that target the Sun ONE architecture and stack. Forte Tools provide services-centric functionality to assemble solutions from services, construct services from components, and build modern client access to services.
FTP	FTP (File Transfer Protocol) allows users to quickly transfer text and binary files to and from a distant or local PC, list directories, delete and rename files on the foreign host, and perform wildcard transfers between hosts.
GSS-API	The Generic Security Service API (GSS-API) is a CAPI for distributed security services. It has the capacity to handle session communication securely, including authentication, data integrity, and data confidentiality. The GSS-API is designed to insulate its users from the specifics of underlying mechanisms. GSS-API implementations have been constructed atop a range of secret-key and public-key technologies.
HTML	HTML (HyperText Markup Language) is the authoring software language used on the World Wide Web.
HTTP	HTTP (HyperText Transfer Protocol) is the basis of Internet browsers and Web servers. A client makes an HTTP request to a server, and HTML HyperText is returned via HTTP.
HTTPS	HTTPS (HyperText Transfer Protocol Secure) is a type of server software which provides the ability for secure transactions over the World Wide Web.
iAF	The iPlanet Application Framework (iAF) is iPlanet's implementation of a presentation framework. As a standards-based application framework for enterprise Web application development, it unites familiar concepts such as display fields, application events, component hierarchies, and a page-centric development approach with a state-of-the-art design based on both MVC and Service-to-Workers patterns. Although it is primarily intended to address the needs of J2EE developers building medium-sized applications, iAF also supports large- and massive-scale Web applications. Also refer to Presentation frameworks and Model-View-Controller.
iCAL	iCAL (Internet Calendaring and Scheduling Core Object Specification) is a specification from the Internet Engineering Task Force (IETF) designed to allow people to share and coordinate their appointment calendars over the Internet.
Identity and Policy Services	These services provide the infrastructure for managing user identities, services, and policies. They also provides the fundamental basic services such as discovery of Web services (using UDDI), authentication, single-sign-on, policy evaluations, and security. The core services can be leveraged by other Web services and applications to perform the basic operations.
IFX	IFX version 1.2 provides an XML-based communication protocol that enables the exchange of information among financial institutions, financial institutions and their customers, and financial institutions and their service providers. This latest version features a wide range of functions that allow financial institutions and associated service providers to access account information, download credit card statements, transfer funds, process consumer and business payments, enable bill presentment, and improve customer service. The IFX specification supports a broad range of client devices, such as any standard Web browser software, personal computers with personal financial manager (PFM) software, voice response units (VRUs) that provide bank by phone services, automated teller machines (ATMs), consumer handheld devices, or mobile telephones with data capabilities.
IKE	IKE (Internet Key Exchange) is an automated key management standard for IPsec.

TERM	DEFINITION
IMAP	IMAP (Internet Messaging Access Protocol) allows users to create and manage mail folders over the WAN, as well as scan message headers and then download only selected messages. The ability to read and send e-mail is traditionally implemented via the POP, IMAP, and SMTP standards. In the web services model these protocols may be layered with a SOAP interface that will allow a general client, not just a dedicated e-mail client, to access information that will be provided as XML data.
iMQ message system	The iPlanet Message Queue (iMQ) is Sun's asynchronous reliable messaging product. It includes iMQ Administered Objects, the iMQ Client Runtime, and iMQ Message Service. Also refer to asynchronous reliable messaging.
InfiniBand	Intel's InfiniBand Architecture InfiniBand Technology will be used to connect servers with remote storage and networking devices, and other servers. It will also be used inside servers for inter-processor communication (IPC) in parallel clusters. InfiniBand features small form factors, greater performance, lower latency, easier and faster sharing of data, built in security and quality of service, improved usability (the new form factor will be far easier to add/remove/upgrade than today's shared-bus I/O cards).
IPsec	A collection of IP security measures that comprise an optional tunneling protocol for IPv6.
IPv4	Refer to TCP/IP.
IPv6	IPv6 (Internet Protocol Version 6) is designed to replace and enhance TCP/IP (IPv4). IPv6 has 128-bit addressing, auto configuration, new security features and support for real-time communications and multi-casting.
J2EE	J2EE (Java 2 Platform, Enterprise Edition) offers a standard, extensible, component-based application programming model. Access to many critical services, such as transaction management and persistence, are automated. In addition, J2EE provides reliable access to a range of additional services, thereby reducing the complexity of developing distributed applications. J2EE enables Web Services applications and simplifying their development.
J2EE Connector architecture	This J2EE connector architecture defines a standard way to extend the Sun ONE Service Container to include the functionality of an Enterprise Information System (EIS). With the appropriate connector installed, a Sun ONE application is able to use the functionality of the EIS without having to deal with the complexity of integrating EIS remote access, transactions, and security. The functionality of the EIS appears to the Sun ONE developer as a new service provided by the Service Container.
J2ME	Java technology for mobile and embedded devices is covered by the J2ME—Java 2 Micro Edition. Because of different devices with different capabilities, J2ME is based on <i>configurations</i> and <i>profiles</i> . A <i>configuration</i> defines the minimum set of class libraries available for a range of devices. A <i>profile</i> defines the set of APIs available for a particular family of devices.
JAAS	The Java Authentication and Authorization Service (JAAS) is a Java package that enables services to authenticate and enforce access controls upon users. It implements a Java version of the standard Pluggable Authentication Module (PAM) framework, and supports user-based authorization.
Java	Invented in 1995 by Sun, Java is a programming language that enables software to run on any machine (write once, run anywhere).
JavaCard	Java Card API is an ISO 7816-4 compliant application environment focused on smart cards.
Java Connectors	Java connectors provide standardized access to existing mainframe-based transactions from environments compatible with J2EE. Also refer to J2EE Connector architecture.
JavaMail	Java Mail is one of several J2EE supported standard communication technologies, including RMI-IIOP, JavaIDL and JMS, that are used as a means to communicate on a network, sending messages or invoking services.
Java Server Pages	Refer to JSP.
Java Web client model	This Services-on-Demand category allows applications to be downloaded to desktop computers, handheld devices, home gateway computers, or audiovisual devices and set-top boxes. Included are desktop Java standards and the use of Java Webstart to download Java applications that, by extension, are being used to create the Java Vending Machine (JVM) concept.

TERM	DEFINITION
JAX*	<p>Web Services J2EE's APIs for XML allow web applications to be written entirely in the Java programming language. These APIs fall into two broad categories: those that deal directly with XML documents and those that deal with procedures.</p> <p>Document-oriented Java API for XML Processing (JAXP) — processes XML documents using various parsers.</p> <p>Java Architecture for XML Binding (JAXB) — maps XML elements to classes in the Java programming language.</p> <p>Java API for XML Messaging (JAXM) — sends SOAP messages over the Internet in a standard way.</p> <p>Java API for XML Registries (JAXR) — provides a standard way to access business registries and share information.</p> <p>Java API for XML-based RPC (JAX-RPC) — sends SOAP method calls to remote parties over the Internet and receives the results.</p>
JCA	Java Cryptography Architecture
JDBC	JDBC (Java Database Connectors) provides J2EE's database connectivity. Structured Query Language (SQL) commands or queries can be issued to a relational database, and the results returned to any Java application. The JDBC API supports stored procedures, transactions, connections, and user authentication. JDBC drivers may support connection pooling, distributed transactions, and caching of rows from the database.
JMS	The Java Message Service (JMS) provides an API for asynchronous messaging. Rather than invoke a service and wait for a response, a JMS message is queued for delivery, and control returns to the invoker. In addition to supporting specific message queues—for example, for a specific EJB, JMS supports publish-and-subscribe messaging in which any number of clients can subscribe to (request messages on) well-known topics in a hierarchy of topics, and any number of clients can publish to (send messages to subscribers of) a specific topic. JMS supports reliable, guaranteed delivery of messages.
JNDI	Java Naming and Directory Services (JNDI) provides access to a naming environment. It provides methods for performing directory operations, such as associating attributes with objects and searching for objects using their attributes. JNDI is used for a variety of purposes. JDBC data sources and JTA transaction objects can be stored in a JNDI naming environment. A container provides an environment to its components via a JNDI naming context. JNDI can be used by components in a distributed application to locate one another and initiate communications. Existing corporate directory services can be accessed via JNDI.
JSP	The Java Server Pages (JSP) technology builds on Java Servlet technology to simplify the development of dynamic Web content. JSP supports a page-based metaphor that separates dynamic and static Web content; the JSP page defines a static HTML template, with embedded calls to code written in the Java programming language to fill in dynamic portions of the page.
JVM	Java Virtual Machines (JVMs) provide a common application and application library execution environment that has been adopted by a diverse set of Web services client device manufacturers. They offer secure operations for accessing underlying device-specific capabilities in a way that insulates the developer from those interfaces while providing device-specific integration.
Kerberos	Kerberos is a network authentication protocol designed to provide strong authentication for client/server applications by using secret-key cryptography.
KVM	Kernel Virtual Machines (KVMs) are a reduced capacity versions of Java Virtual Machines for use on limited-footprint portable client devices. For a more complete definition, refer to Java Virtual Machine (JVM).
LDAP	LDAP (Lightweight Directory Access Protocol) is a protocol for accessing directories.
Liberty	The Liberty Alliance Project is a business alliance formed to deliver and support an identity solution for the Internet that enables single sign-on for consumers as well as business users, in an open, federated way. The role of the Liberty Alliance Project is to support the development, deployment and evolution of an open, interoperable standard for network identity. Included is collaboration on standards so that privacy, security, and trust are maintained.

TERM	DEFINITION
Metadata Repository	This component of the Netbeans architecture makes it easier to build modules that support other programming languages, in addition to enhancing performance and features that are related to refactoring. By supporting standardized models for metadata in a language-neutral way, the Metadata Repository also makes it easier to integrate third-party products, such as UML tools. Metadata refers to data that describes the structure and characteristics of program elements or data—for example, the structure and method signatures of a Java class file.
MIDP	The Mobile Information Device Profile (MIDP) is a set of Java APIs which, together with the Connected, Limited Device Configuration (CLDC), provides a complete J2ME application runtime environment targeted at mobile information devices, such as cellular phones and two-way pagers.
Mobile IP	Mobile IP is a set of extensions to the Internet Protocol for packet data transmission that serves nomadic users connecting on a wireline, rather than a wireless, basis. The protocol is being developed by the IETF.
MVC	The Model-View-Controller (MVC) design model holds that an application consists a Model, some Views of the Model, and some Controllers. The Model is the part of the application that contains the actual application logic. The Model does the database access, computes numbers, and manipulates data structures. The View and Controller represent the user interface of the application. The user interface is conceptually split into input and output components. The Controller is an input component that supplies information to the Model. The View is an output component which displays information from the Model. The View typically communicates with the Model by registering itself as a callback and responding to events generated by the Model.
NetBeans IDE	Sun's integrated development environment (IDE) for creating, assembling, and deploying Sun ONE applications and Web services. NetBeans is open source, modular, and standards-based. Because it is written in the Java language, it can run on any platform with a Java Virtual Machine that is compliant with the Java 2 platform.
NFS	NFS (Network File System) is a distributed-file-system protocol that allows a computer on a network to use the files and peripherals of another networked computer as if they were local. The protocol was developed by Sun.
OBI	OBI (Open Buying on the Internet) is a standard that provides a generic set of requirements, architecture, and a technical specification for Internet purchasing solutions in the general context of e-commerce.
OCSP	OCSP is a client-server based solution allowing a client to send a status request to the server for obtaining revocation information about some particular certificate(s) in mind. In this way, OCSP may be used for providing more timely revocation information than is possible with CRLs (certification revocation lists).
P3P	Platform for Privacy Preferences Project (P3P) standard establishes a complex computerized negotiation that extends the information-gathering potential of the Web site and the privacy-protection possibilities inherent in each user visit to the Web site.
Platform Services	This interface is located at the lowest level of the Sun ONE architecture. It provides the functions needed to allocate and manage the resources of the underlying network and hardware platform required to host the higher-level services in the Sun ONE stack. Platform Services' primary focus is to provide the system programming interfaces to the platform's basic functionality that can be used by Web service execution middleware. It is important to note, however, that Platform Services may also offer certain application interfaces directly in the form of Web Services for use by both Web applications and higher level Web Services.
POP	POP is the Point of Presence at which ISPs exchange traffic and routes at Layer 2 (Link Layer) of the OSI model.
Portal Server	This Service Delivery component of the Sun ONE architecture provides a single access point to diverse types of information originating from many sources. It offers several features to end users, as well as to the Web applications and services that they are using. These features include aggregation, presentation, personalization, and security.
POSIX	POSIX (Portable Operating System Interface) is an IEEE standard, based on historical UNIX systems, for an application interface that will run on a variety of operating systems, such as Solaris.

TERM	DEFINITION
Presentation frameworks	These frameworks gather information from both end users and the business layer of an application. They then generate the user interface and process the user's interaction with it. The iPlanet Application Framework (iAF) is iPlanet's implementation of a presentation framework.
RDF	The Resource Description Framework (RDF) integrates a variety of applications from library catalogs and world-wide directories to syndication and aggregation of news, software, and content to personal collections of music, photos, and events using XML as an interchange syntax. The RDF specifications provide a lightweight ontology system to support the exchange of knowledge on the Web.
PKCS	Public-Key Cryptography Standards (PKCS) are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography.
PKIX	The goal of the Internet Public Key Infrastructure (PKI) standards effort is to meet the needs of deterministic, automated identification, authentication, access control, and authorization functions. Support for these services determines the attributes contained in the certificate as well as the ancillary control information in the certificate such as policy data and certification path constraints.
PPP	PPP (Point-to-point protocol) is a protocol that allows a computer to connect to the Internet with a standard dial-up telephone line and a high-speed modem.
RSS	RDF Site Summary (RSS) is a lightweight multipurpose extensible metadata description and syndication format. RSS is an XML application, conforms to the W3C's RDF Specification and is extensible via XML-namespace and/or RDF based modularization.
SAML	SAML (Security Assertion Markup Language) is an XML-based security standard for exchanging authentication and authorization information.
SASL	SASL (Simple Authentication and Security Layer) is a method for adding authentication support to connection-based protocols. To use SASL, a protocol includes a command for identifying and authenticating a user to a server and for optionally negotiating protection of subsequent protocol interactions. If its use is negotiated, a security layer is inserted between the protocol and the connection.
SAX	SAX, the Simple API for XML, is a standard interface for event-based XML parsing.
SCSI	SCSI (Small Computer Systems Interface) allows devices such as hard disks, optical disk drives, tape drives, CD-ROM drives, printers and scanners to communicate with the computer's main processor. SCSI is both a bus and an interface standard.
Sendmail	Sendmail is the UNIX software that delivers electronic mail.
Services on Demand	This umbrella category encompasses past Web initiatives, today's Web applications, Web services, and Java Web clients. It also includes tomorrow's new services, including contextually enhanced Web services that are aware of user context and identity to create a superior interactive, online experience. Potentially, Services on Demand will also include new technologies such as peer-to-peer (JXTA) and dynamic configuration infrastructure (Jini). Services on Demand have the flexibility to encompass new protocols and methods of operation in order to deliver customized, personalized services whenever and wherever they are needed.
Servlets	Java Servlet technology provides a basic mechanism for generating dynamic web content. Servlets were developed as an improvement over CGI scripts, which are generally platform-specific, and are limited in their ability to support rich interaction. Like all J2EE components, servlets run in a container implemented by the J2EE platform provider. The container manages a servlet's interaction with its client and provides a rich environment for the servlet to use to access various services based on Java technology. A servlet container implements the entire Java 2 Platform, Standard Edition APIs. This makes a variety of technologies based on the Java programming language available to servlets, including JDBC, Java Naming and Directory Interface, RMI, JavaBeans, and others.
S/MIME	S/MIME (Secure Multipurpose Internet Mail Extensions) is a specification designed to be easily integrated into e-mail and messaging products. S/MIME builds security on top of the industry standard MIME protocol according to the Public Key Cryptography Standards (PKCS).

TERM	DEFINITION
SMS	SMS (Short Message Service) is a means to send or receive short alphanumeric messages to or from mobile telephones.
SMTP	Simple Mail Transport Protocol is a TCP/IP protocol for sending e-mail between servers.
SOAP	SOAP (Simple Object Access Protocol) provides an extensible XML messaging protocol and also supports an RPC programming model. A number of SOAP implementations are available.
SQL	SQL (Structured Query Language) is a standard database language used for creating, maintaining and viewing database data.
SSL	SSL (Secure Socket Layer) provides a secure mechanism for clients to access hosts on the Internet, without someone eavesdropping or tampering with the messages.
SSL/TLS	SSL/TLS (Secure Socket Layer/Transparent LAN Service) is a secure high-speed VPN (Virtual Private Network) that hides the complexity associated with wide area networks. SSL was invented by Netscape Communications; TLS is the equivalent standard from the IETF.
Sun ONE	Sun ONE (Open Net Environment) is Sun Microsystems' standards-based software vision, architecture, platform, and expertise for building and deploying Services on Demand.
SyncML	SyncML is a standard, based on XML, used to synchronize the data between clients and servers.
TCP/IP	TCP/IP (Transport Control Protocol/Internet Protocol) provides a mechanism to establish connections and reliably deliver streams of data between Internet hosts.
UBL	Universal Business Language (UBL) will be used by vertical industry segments to describe the semantics associated with e-business XML documents. The language is being developed by OASIS, the industry's XML standards organization.
UDDI	UDDI (Universal Description Discovery and Integration) is a project to create a platform-independent, open framework for describing services, discovering businesses, and integrating business services using the Internet, as well as an operational registry that is available public ally. The UDDI initiative is an industry consortium lead by Accenture, Ariba, Commerce One, Compaq, Edifecs, Fujitsu, HP, I2, IBM, Intel, Microsoft, Oracle, SAP, Sun Microsystems, and VeriSign. More than 130 companies have joined the UDDI initiative.
vLIP	A protocol developed by Sun, designed to support tighter integration of the directory with other Sun/iPlanet applications.
VoiceXML	VXML, or VoiceXML, technology allows a user to interact with the Internet through voice-recognition technology by using a voice browser and/or the telephone. Using VXML, the user interacts with voice browser by listening to audio output that is either pre-recorded or computer-synthesized and submitting audio input through the user's natural speaking voice or through a keypad, such as a telephone.
WBEM	Web-Based Enterprise Management (WBEM) is a set of management and Internet standard technologies developed to unify the management of enterprise computing environments. WBEM provides the ability for the industry to deliver a well-integrated set of standard-based management tools leveraging the emerging Web technologies.
Web applications	This category includes most of the Services on Demand products delivered prior to the year 2002. Web applications deliver dynamic content via Web browsers for human interaction. The Sun ONE architecture includes a variety of tools for developing, deploying, and dynamically updating Web applications.
Web client model	Refer to Java Web Client Model.
WebDAV	WebDAV (Web Distributed Authoring and Versioning) provides a network protocol for creating interoperable, collaborative applications. The stated goal of the WebDAV working group is to define the HTTP extensions necessary to enable distributed Web authoring tools to be broadly interoperable, while supporting user needs. Features already developed include: Locking (concurrency control), Properties (using SML properties to provide storage for arbitrary metadata; and Namespace Management (copying or moving Web site resources).

TERM	DEFINITION
Web services	These self-describing, modular, encapsulated functions can discover and engage other Web services to complete complex tasks over the Internet. Unlike traditional, hard-wired client/server applications, Web services are loosely coupled. They can dynamically locate and interact with other components on the Internet to provide services, and can themselves be dynamically located and used by other Web services. In other words, Web services transform services into clearly defined components and allow those services to be easily interconnected. A Web service is usually invoked by a program, not directly by a human user. It is used to integrate applications—either within the enterprise or over the Internet—between the enterprise and its customers and business partners.
WSDL	Web Services Description Language is a technology developed by Ariba, IBM, and Microsoft that specifies a common XML framework for describing the interfaces to a Web service.
WML	Wireless Markup Language is the browser language used with WAP (Wireless Application Protocol).
X.509	A cryptography term, X.509 is part of the ITU-T X.500 Recommendation that deals with Authentication Frameworks for Directories. Within X.509 is a specification for a certificate that binds an entity's distinguished name to its public key through the use of a digital signature.
XACML	XACML (eXtensible Access Markup Language), an XML-based standard for expressing policies for information access over the Internet.
XHTML	XHTML is a reformulation of HTML 4.0 as an XML 1.0 application. The hybrid language allows users to migrate from HTML to XML as users can create documents in HTML while mixing in XML functions.
XKMS	XKMS (XML Key Management Specification) for distribution and registration of public keys.
XML	eXtensible Markup Language is a standard developed by the World Wide Web Consortium (W3C). XML permits structured exchanges of data between machines attached to the Web, allowing heterogeneous Web servers to communicate.
XML DSIG	XML digital signatures provide integrity, message authentication, and/or signer authentication services for data of any type, whether located within the XML that includes the signature or elsewhere.
XML Encrypt	TBD
XSLT	JAXP supports the XSLT (XML Stylesheet Language Transformations) standard, providing control over the presentation of the data and enabling developers to convert the data to other XML documents or to other formats, such as HTML.



# Bibliography

---

This Bibliography\* provides a chapter-by-chapter list of references. Note that it does not include all of the references that appear in the interface tables. For a complete listing of references for each chapter, refer to the interface tables as a supplement to this Bibliography.

## *Preface: About This Book*

<http://www.sun.com>

<http://www.sun.com/sunone>

## *Chapter 1. Delivering Services on Demand*

<http://dcb.sun.com/practices/webservices>

## *Chapter 2. The Sun ONE Architecture*

<http://www.sun.com>

<http://www.sun.com/sunone>

## *Chapter 3. J2EE Components and Containers*

<http://www.java.sun.com/j2ee>

## *Chapter 4. J2EE Connector Architecture and Web-Service-Based Integration*

<http://www.jcp.org/jsr/detail/58.jsp>

## *Chapter 5. Asynchronous, Reliable Messaging*

### *Web Pages*

<http://www.jcp.org/aboutjava/communityprocess/maintenance/JMS/index.html>

### *Printed Documents*

iPlanet Message Queue for Java Developers' Guide

iPlanet Message Queue for Java Release Notes - Technical Notes section

## *Chapter 6. Business Process Integration*

UMM - Published by UN/CEFACT, July 2001 at

[http://www.unece.org/cefact/docum/download/01bp\\_n090.zip](http://www.unece.org/cefact/docum/download/01bp_n090.zip)

Wf-XML - Published by the Workflow Management Collation (WfMC). Published January 2002 at <http://www.wfmc.org/standards/docs/Wf-XML-11.pdf>

<http://www.oasis-open.org/committees/reqrep/documents/2.0/specs/ebrs.pdf>

<http://www.jcp.org/jsr/detail/93.jsp>

<http://www.w3.org/TR/xmlsig-core/>

<http://www.jcp.org/jsr/detail/93.jsp>

<http://www.jcp.org/jsr/detail/101.jsp>

## *Chapter 7. Development Tools*

### *Web Pages*

<http://www.netbeans.org>

<http://apisupport.netbeans.org/>

<http://form.netbeans.org/>

<http://mdr.netbeans.org/>

<http://www.netbeans.org/modules.html>

<http://www.corba.netbeans.org/>

<http://www.db.netbeans.org/>

<http://www.jini.netbeans.org/>

<http://www.projects.netbeans.org/>

<http://www.autoupdate.netbeans.org/>

<http://www.j2eeserver.netbeans.org/>

#### *Printed Documents*

JSR 30 - Connected, Limited Device Configuration  
JSR 36 - Connected Device Configuration  
JSR 36 - Mobile Information Device Profile for the J2ME Platform  
JSR-40, the Java Metadata Interface (JMI) specification  
JSR 46 - J2ME Foundation Profile  
JSR 66 - J2ME RMI Profile  
JSR 75 - PDA Profile for the J2ME Platform  
JSR 118 - Mobile Information Device Next Generation  
JSR 120 - Wireless Telephony Communication APIs (WTCA)  
JSR 134 - Java Game Profile  
JSR 139 - CLDC Next Generation

### *Chapter 8. Presentation Frameworks*

#### *Web Pages*

<http://www.jcp.org/jsr/detail/127.jsp>

#### *Printed Documents*

JSP specification and the J2EE Blueprint

### *Chapter 9. The Portal Server*

<http://developer.iplanet.com/tech/ips/>

<http://www.jcp.org/jsr/detail/168.jsp>

### *Chapter 10. The Java Web Client Model*

#### *Web Pages*

Device Cache Management, available via <http://www.jcp.org/jsr/detail/107.jsp>

J2EE Client Provisioning Specification (JSR 124), available via  
<http://www.jcp.org/jsr/detail/124.jsp>

J2ME Platform Specification (JSR 68), available via <http://www.jcp.org/jsr/detail/68.jsp>

J2ME CDC (JSR 36): <http://www.jcp.org/sr/detail/36.jsp>

J2ME Foundation Profile (JSR 46), available via <http://www.jcp.org/jsr/detail/46.jsp>

J2SE Merlin Release (JSR 59), available via <http://www.jcp.org/jsr/detail/59.jsp>

JAIN 3G MAP Mobile Application Intercommunication (JSR 123), available via <http://www.jcp.org/jsr/detail/123.jsp>

JAIN Service Creation Environment, available via <http://www.jcp.org/jsr/detail/100.jsp>

Java APIs for Bluetooth (JSR 82), available via <http://www.jcp.org/jsr/detail/82.jsp>

Java APIs for Security, Signing, Trust, Signature, Encryption, and the Javacard, available via <http://www.java.sun.com/products/javacard/javacard21.html>

Javacard Integration into Phones, PDAs, XML, Trust, Signature, Encryption (JSRs 105 - 107), available via <http://www.jcp.org/jsr/detail/105.jsp> and <http://www.jcp.org/jsr/detail/106.jsp> and <http://www.jcp.org/jsr/detail/107.jsp>.

Personal Basis Profile (JSR 129), available via <http://www.jcp.org/jsr/detail/129.jsp>

Personal Profile (JSR 62): <http://www.jcp.org/jsr/detail/162.jsp>

P01honelets (JSR 61), available via <http://www.jcp.org/jsr/detail/61.jsp>

MidP (JSR 37, 118), available via <http://www.jcp.org/jsr/detail/37.jsp> and <http://www.jcp.org/jsr/detail/118.jsp>

CLDC Next Generation (JSR 139), available via <http://www.jcp.org/jsr/detail/139.jsp>

PDA Profile for the J2ME platform (JSR 75), available via <http://www.jcp.org/jsr/detail/75.jsp>

Service Location Protocol (JSR 140), available via <http://www.jcp.org/jsr/detail/140.jsp>

Service Provide Presence (JSR 123), available via <http://www.jcp.org/jsr/detail/123.jsp>

SyncML Information, available via <http://www.synchml.org/technology.html>

Wireless Telephony on J2ME (JSR 120); <http://www.jcp.org/jsr/detail/120.jsp>

#### *Printed Documents*

StarOffice and StarPortal User Documentation

J2ME Widget Sets, User Events, and Communications

## *Chapter 11. Identity and Policy Services*

#### *Web Pages*

<http://java.sun.com/products/jaas>

<http://www.iplanet.com>  
<http://www.oasis-open.org/security-services>  
<http://www.oasis-open.org/committees/security>  
<http://www.oasis-open.org/committees/provision>  
<http://www.oasis-open.org/committees/xacml>  
<http://www.oasis-open.org/committees/security>  
<http://www.projectliberty.org>  
[dmtf.org/standards/standard\\_webem.php](http://dmtf.org/standards/standard_webem.php)  
<http://www.jcp.org/jsr/detail/155.jsp>  
[www.jcp.org/jsr/detail/48.jsp](http://www.jcp.org/jsr/detail/48.jsp)  
[www.jcp.org/jsr/detail/3.jsp](http://www.jcp.org/jsr/detail/3.jsp)  
[www.jcp.org/jsr/detail/60.jsp](http://www.jcp.org/jsr/detail/60.jsp)  
<http://www.ietf.org/html.charters/pkix-charter.html>  
<http://www.rsasecurity.com/rsalabs/pkcs/>  
<http://www.ietf.org/html.charters/tls-charter.html>  
<http://www.w3.org/TR/2002/REC-xmlsig-core-20020212/>  
<http://www.w3.org/Encryption/2001/>  
<http://www.w3.org/2001/XKMS/>  
<http://java.sun.com/products/jsse/>  
<http://www.w3.org/TR/WD-logfile.html>  
<http://java.sun.com/j2se/1.4/docs/guide/util/logging/overview.html>  
<http://www.projectliberty.org>

*Printed Documents*

JETF RFC 2853

JSR 72 JSR 74

JSR 105

JSR 106

All of the following documents are from the Internet Engineering Taskforce (IETF).

RFC 2571

Standard 15: RFC 1157

Standard 16: RFC 115, RFC 1212, RFC 1215

Standard 58: RFC 2578, RFC 2579, RFC 2580

RFC 1906, RFC 2572, RFC 2574

RFC 1905

RFC 2573, RFC 2575

## *Chapter 12. Platform Services*

### *Web Pages*

<http://docs.sun.com>

[FTP] “File Transfer Protocol,” J. Postel, J.K. Reynolds, IETF RFC 959. Oct. 1985, available via <http://www.ietf.org/rfc/rfc0959.txt>

[gss\_doc] “GSS-API Programming Guide” (from Solaris 9 Software Developer Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[gss\_man] Generic Security Service API: Solaris 9 manual pages: libgss(3LIB), and manual pages in section (3gss). Sun Microsystems Inc., available via <http://docs.sun.com>

[ike] “The Internet Key Exchange,” IETF RFC 2409, available via <http://www.ietf.org/rfc/rfc2409.txt>

[ipsec] “Security Architecture for the Internet Protocol,” IETF RFC 2401, available via <http://www.ietf.org/rfc/rfc2401.txt>

[ipsec\_man] “IPsec protocol,” Solaris manual pages: ipsec(7p). Sun Microsystems Inc., available via <http://docs.sun.com>

[IP] “Internet Protocol, DARPA Internet Program Protocol Specification,” IETF RFC 791, Sept. 1981, available via <http://www.ietf.org/rfc/rfc0791.txt>

[IP\_man] “IP protocol,” Solaris manual pages: ip(7p), Sun Microsystems Inc., available via <http://docs.sun.com>

[IPv6\_man] “IP protocol,” Solaris manual pages: ip(7p), Sun Microsystems Inc., available via <http://docs.sun.com>

[IPv6] “Internet Protocol, Version 6 (IPv6) Specification,” IETF RFC 2460, December 1998, available via <http://www.ietf.org/rfc/rfc2460.txt>

[krb\_impl] Kerberos version 5 reference implementation, available via <http://Web.mit.edu>

[krb\_man] Kerberos API: Solaris manual pages: kerberos(1) and manual pages in section (3krb). Sun Microsystems Inc., available via <http://docs.sun.com>

[krbv5] “Kerberos version 5”, RFC 1510 available via <http://www.rfc-editor.org>

[gss] “Generic Security Service API specification,” IETF RFC 2744, available via <http://www.ietf.org/rfc/rfc2744.txt>

[ldap\_doc] “System Administration Guide: Naming and Directory Services (DNS, NIS and LDAP),” (part of Solaris 9 System Administrator Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[ldap\_man] “LDAP programming interfaces” Solaris manual pages: libldap(3lib) and manual pages in section (3ldap), Sun Microsystems Inc., available via <http://docs.sun.com>

[random] “Randomness Recommendations for Security,” IETF RFC 1750, December 1994, available via <http://www.rfc-editor.org>

[random\_man] Random number API: Solaris manual pages: random(3c) and random(7d). Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_admin1] “Sun Cluster 3.0 12/01 Data Services Installation and Configuration Guide” (from Sun Cluster 3.0 12/01 Data Services Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_admin2] “Sun Cluster 3.0 12/01 Software Installation Guide” (from Sun Cluster 3.0 12/01 Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_admin3] “Sun Cluster 3.0 12/01 System Administration Guide” (from Sun Cluster 3.0 12/01 Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_admin4] “Sun Cluster 3.0 12/01 Hardware Guide” (from Sun Cluster 3.0 12/01 Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

m\_doc1] “Sun Cluster 3.0 Data Services Developer’s Guide” (from Sun Cluster 3.0 12/01 Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_doc2] “Sun Cluster 3.0 Concepts” (from Sun Cluster 3.0 12/01 Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[rgm\_man] “Cluster Resource Management API” manual pages, sections (1HA) and (3HA), Sun Microsystems Inc., available via <http://docs.sun.com>

[rsm\_man] Remote Shared Memory API: Solaris 9 manual pages: librsm(3LIB), and manual pages in section (3RSM). Sun Microsystems Inc., available via <http://docs.sun.com>

[seam] “Sun Enterprise Authentication Mechanism 1.0.1”, Programmer’s Guide, Sun Microsystems Inc., available via <http://docs.sun.com>

[srm1.2] “Solaris Resource Manager 1.2 System Administration Guide,” Sun Microsystems Inc., available via <http://docs.sun.com>

[srm9] “System Administration Guide: Resource Management and Network Services” (part of Solaris 9 System Administrator Collection), Sun Microsystems Inc., available via <http://docs.sun.com>

[srm\_exacct] “Extended Accounting API,” Solaris 9 manual pages: libexacct(3lib) and manual pages in section (3exacct), Sun Microsystems Inc., available via <http://docs.sun.com>

[srm\_project] “Project Database API,” Solaris 9 manual pages: libproject(3lib), project(4), and manual pages in section (3project), Sun Microsystems Inc., available via <http://docs.sun.com>

[SMTP] “NFS version 4 Protocol,” Shepler et.al., Dec. 2000, IETF RFC 3010, available via <http://www.ietf.org/rfc/rfc3010.txt>

[SMTP] “Simple Mail Transfer Protocol,” J. Postel, Aug. 1982, IETF RFC 821, available via <http://www.ietf.org/rfc/rfc0821.txt>

[socket\_man] “Socket API,” Solaris manual pages: libsocket(3lib) and manual pages in section (3socket), Sun Microsystems Inc., available via <http://docs.sun.com>.

## *Chapter 13. Core Web Services*

<http://www.iplanet.com>

\*This Bibliography contains url’s to third party sites. Sun has no responsibility, and makes no representation or warranties, regarding information on these third party sites.



# Index

---

- A
- account management and provisioning. See identity management.
- adapters
  - pluggable 185
- API Support module 106
- application events
  - iPlanet Application Framework (iAF) 138
- application server 39
  - J2EE standards 40
- applications
  - integration challenges 81
  - integration via J2EE Connector architecture 47
  - interoperability with Sun ONE 20
  - J2EE
    - integration 27
  - personal productivity 21
  - types of business 3
  - Web services 27
  - See also Web applications
- asynchronous reliable messaging 48
  - requirements beyond JMS 59–61
    - multiple distributed transaction support 61
    - multiple protocol support 60
    - multiple queue delivery styles 60
    - object management 60
    - pluggable persistence 61
    - security 60
  - requirements of 56
  - use of JMS 57–59
  - via ebXML 83–87
- audit
  - via ebXML messaging 89
- authentication 168–171, 178
  - Liberty Alliance Project 180

- Solaris 195
  - via ebXML messaging 89
- B
- browser compatibility 21
- C
- calendar Core Web service 206
- chat Core Web service 207
- CIM model 183–186
- client/server computing model 5
- Cluster Interface Set, Solaris 191, 191–192
- collaboration among businesses 79–91
- conference Core Web service 207
- connection management
  - via J2EEConnector architecture 50
- contacts Core Web service 206
- container-provided services, J2EE 33
- contracts of J2EE Connector architecture 49–51
- CORBA
  - interoperability with J2EE 37
- Core Web Services 203–207
  - communications applications 205–207
    - Calendar 206
    - Conferencing 207
    - Contacts 206
    - Mail 206
  - defined 203
  - Portal-based 203–205
    - File 205
    - Location Web service 204
    - Presence Web service 204
    - Search 205
    - Usage 205
- customization
  - via Portal Server 146
- D
- database servers
  - access through J2EE 37
- development tools. See tools, development.
- DMTF
  - schema 184
- DOM manipulation
  - with MVC 127
- E
- ebXML 11, 21, 82–91
  - exchange processes 90
  - functional overview 87
  - messaging 83

- reliable electronic business exchange 88
  - objectives and architecture ii, 82
- efficient object management
  - iPlanet Application Framework (iAF) 139
- e-mail
  - JavaMail API 36
- e-mail Core Web service 206
- Enterprise Information Systems (EIS)
  - integration with Sun ONE applications 47
- Enterprise JavaBeans (EJB) 30–32
- Enterprise JavaBeans (EJB) 2.0 specification 56
- F
- federated identity systems 180
- federated identity systems. See also Liberty Alliance Project. 180
- File Web service 205
- Form Editor 107
- Forte IDE 110–117
  - extensible architecture design 115
  - integrated architecture capabilities 114
  - interfaces 117
  - primary components 110
  - service assembly 116
  - service creation 115
  - service deployment 117
  - service development 113
  - services-centric functionality 113
  - third-party tools 113
- G
- Generic Security Services (GSS) API 196
- GUI
  - customization via Portal Server 121, 146
  - relation to Portal Server 144
  - See also UI, presentation frameworks.
- H
- hardware
  - Platform Services 189–197
- history of networked computing 4–6
- host-based computing model, 5
- I
- identity
  - authentication 168
  - cross-domain single sign-on 171
- Identity and Policy Services
  - interface table 178
- Identity and Policy services 146, 165–187
  - federated identity systems 180
  - identity, roles, and security 166–178

- interface tables 178
  - Management Services 181–187
- identity management 171–172
  - delegated management 172
  - profile API 172
  - user organization and integration 172
  - user provisioning and self-registration 171
- iDSAME 165
- iDSIE 165
- integration 77–96
  - challenges 78
  - See also process integration.
- Internet Key Exchange (IKE) 196
- iPlanet Application Framework (iAF) 30, 42, 130–141
  - features
    - application events 138
    - efficient object management 139
    - formal model entity 137
    - hierarchical views and component scoping 138
    - ready-to-use, high-level features 140
    - scalability 141
    - support for parallel content 140
    - symmetrical display/submit handling 137
    - tool readiness 141
  - implementation of MVC2 131
  - relationship to JSR 127 132
  - technical overview of iAF core 135
  - types of functionality 133
  - use of design patterns 133
- iPlanet Application Server 40–43
  - component life cycle optimizations 42
  - deployment options 41
  - high availability 41
  - interfaces 42
  - management 41
  - platform integration 42
  - scalability 41
  - tools integration 42
- iPlanet Calender Server 206
- iPlanet Contacts Server 206
- iPlanet Integration Server 91–96
  - Controller/Coordination Layer 92
  - Data Transformation and Translation Layer 94
  - Document Exchange Process Engine 94
  - Message Routing Table 94
  - Messaging Interface Layer 95
- iPlanet Message Queue for Java 61–75

- iPlanet Portal Server 203
- iPlanet Portal Server (iPS) Provider API 145
- J
- J2EE
  - relationship to presentation frameworks 141
  - See also J2EE Connector architecture.
- J2EE components and containers 27–43
  - access to database servers 37
  - access to name and directory servers 38
  - advantages 27
  - container-provided services 33
  - cross-platform and CORBA interoperability 36
  - Enterprise JavaBeans (EJB) 30
  - interface tables 38
  - Java Server Pages (JSP) 29
  - Java Servlet technology 29
  - JavaMail API 36
  - JAXB 35
  - JAXM 35
  - JAXP 34
  - JAXR 35
  - JAX-RPC 35
  - Web services 34
- J2EE Connector architecture
  - advantages 49
  - application-level contracts 51
  - packaging and deployment 51
  - system-level contracts 50
  - use in integration of EIS systems 47–53
- Java 2 Platform, Enterprise Edition. See J2EE.
- Java Message Service (JMS)
  - use in integrating MOM environments 48
- Java Messaging Service. See JMS.
- Java Server Pages (JSP) 29
- Java Servlet technology 29
- Java Specification Request 127. See JSR 127
- Java Vending Machine (JVM) 13
- Java Web client
  - support from Portal Server 149
- Java Web client model 153–160
  - client device 157
  - defined 13
  - environment 155
  - extended services 158
  - interface table 160
  - mobile devices 158
  - MVC design 155

- protocols and payloads 154
  - server-side provisioning 160
  - supporting architectural elements 156
  - telephony access mechanisms 159
  - XML information services and device interaction 157
- JavaMail API 36
- JAXB 35
- JAXM 27, 35
- JAXP 28, 34
- JAXR 27, 35
- JAX-RPC 27, 35
- JAX-RPC tools 52–53
- JDBC 37
- JMS 57–59
  - defined 57
  - functions not included 58
  - JMS domains 58
  - JMS messages 58
  - JMS provider 57
  - objectives 57
  - portability 58
  - requirements 59
- JMX specification 186
- JNDI 38
- JSR 127
  - iAF's implementation 132
  - use with MVC-based presentation frameworks 128
- JVM and KVM operating environments 158
- K
- Kerberos 177
  - Sun Enterprise Authentication Mechanism (SEAM) 196
- KVM 158
- L
- Liberty Alliance Project 24, 180, 203
- location monitoring
  - via Portal Server 149
- Location Web service 204
- logging and audit 177
- M
- management of applications
  - via Portal Server 147
- management of users
  - via Portal Server 147
- management of Web services
  - via Portal Server 150
- Management Services 181–187
  - architecture 183

- integration of existing management schemes 185
  - interface table 187
  - use of CIM model 183–184
  - use of SNMP 186
- messaging
  - JMS 33
  - to user via Portal Server 149
  - via ebXML 83–87
  - via EJB 32
  - via SOAP messaging API (JAXM) 36
- Metadata Repository (MDR) 108
- Microsoft .NET 20, 21
  - interoperation with Sun ONE 20
- mobile computing environments
  - challenges 155
  - JVM and KVM operating environments 158
  - telephony access mechanisms 159
  - XML information services and device interaction 157
- mobile devices
  - challenges 158
- Model-View-Controller (MVC) design model
  - DOM manipulation 127
  - JSR 127 architecture 128
  - template and non-template based architectures 126–128
  - template engines 126
  - use by Java Web clients 155
  - use by presentation frameworks 122–128
  - See also MCV1, MCV2.
- MVC1 123
- MVC2
  - iAF's implementation 131–133
  - use by presentation frameworks 124
- N
- name and directory servers
  - access through JNDI 38
- NetBeans IDE
  - Core and APIs 103
- Netbeans IDE 102
  - interface table 109
- network platform
  - Platform Services 189–197
- networked computing
  - evolution of 4–6
- news Core Web service 207
- nonrepudiation 159, 178
- Notification Web service 204

- O
- OASIS UBL 21
- P
- packaging and deployment
  - resource adapters 51
- partnerships, business. See Collaboration among businesses
- personalization
  - via Portal Server 146
  - Web services
    - via Portal Server 148, 149
- Platform Services 13, 189–196
  - hardware platform and resource management 190
  - purpose and functions 189
- policy management and evaluation 172–175
  - management and evaluation APIs 174
  - plug-in SPIs 174
  - Policy Framework 173
- Portal Server 143
  - aggregation and presentation of Web applications 144
  - enhancement for Web services 147–150
  - interface tables 151
- presence monitoring
  - via Portal Server 149
- Presence Web service 204
- presentation frameworks 121–141
  - development issues 125
  - interface tables 142
  - overview 121–122
  - See also iPlanet application framework, Model-View-Controller (MVC) design model
- privacy 175–176
- process integration 77–96
  - via ebXML ii, 82–91
- Providers
  - used by Portal Server 145
- Public Key Infrastructure (PKI) 176
- R
- resource adapters
  - packaging and deployment 51
  - use in integration of EIS systems 52
  - use in J2EE connector architecture 51
- RMI-IIOP 37
- S
- scalability
  - horizontal hardware 190
  - iPlanet Application Framework (iAF) 141
- security
  - all types



- via Identity and Policy Services 165–178
  - application
    - via J2EE Connector architecture 51
    - via Portal Server 146
  - Kerberos 177
  - Liberty Alliance Project 180
  - logging and audit 177
  - place in Sun ONE architecture 176
  - privacy 175–176
  - Public Key Infrastructure (PKI) 176
  - strong random numbers 196
  - user
    - via Portal Server 146
  - Web services
    - via Portal Server 150
- Service Container 16
  - architecture as defined by J2EE 27–43
  - components 53
- service delivery
  - Portal Server 143–150
  - via J2EE Connector architecture 47–51
- Service Delivery box 16, 39
- Service Stack 15
  - product mappings to 18
- Services on Demand
  - defined 4, 6
  - delivery methods 153
  - J2EE APIs 27
  - phases of adoption 21
- single sign-on 170
  - cross-domain 171
  - Liberty Alliance Project 180
- SNMP 186
- SOAP 22, 52
  - messaging API (JAXM) 35
- Solaris 190–197
  - Cluster Interface Set 191–192
  - Forte IDE support 113
  - naming, registry, and directory services 195
  - networking 193
  - resource management 192
  - security 195–197
  - storage, filing, and data access 194
- Solaris Resource Management (SRM) 192
- Solaris Volume Manager (SVM) 194
- strong random numbers 196
- Sun ONE architecture

- development tools 101–117
- elements that support the Java Web client model 156
- hardware platform and resource management 190
- integration and interoperability 18
- iPlanet Application Framework (iAF) 130–141
- J2EE components and containers 27–43
- J2EE Connector architecture 47–53
- Java Web client 153–160
- layers hosted on variety of OS and network platforms 190
- Management Services 181–187
- overview 15–24
- overview of delivery of Services on Demand 3–13
- phases of adoption 21
- Platform Services 189–197
- Portal Server 143–150
- Service Stack 15
- support for Web services 52–53
- Swing 121, 125
- synchronous messaging
  - history 56
  - limitations 55
- T
- telephony access mechanisms 159
- template engines
  - with MVC 126
- tools, development 13, 101–117
  - API Support module 106
  - Form Editor 107
  - Forte IDE 110–117
  - integration via iPlanet Application Server 42
  - J2EE 34–36
  - Metadata Repository (MDR) 108
  - NetBeans IDE 102–106
  - requirements for Sun ONE tools 101
  - third party 113
  - third-party 101
- transaction management
  - via J2EE Connector architecture 51
- U
- UDDI 35, 177
- UI 121
  - automated via Portal Server 150
  - See also GUI.
- Universal Business Language (UBL) 96
- usage monitoring
  - via Portal Server 150
- Usage Web service 205

- W
- Web applications
  - delivery
    - Portal Server 143–150
  - EIS
    - integration via J2EE Connector architecture 47
    - integration via Web services 51
    - model defined 7
- Web client model. See Java Web client.
- Web services
  - aggregation
    - via Portal Server 143, 144, 148
  - as integration mechanism 51
  - availability through Cluster Interface set 191–192
  - communications applications Core Web services 205–207
  - defined 5, 8–12
  - delivery
    - via Portal Server 143
  - differences from human-oriented services 9
  - dynamic 22
  - implementing with Sun ONE 53
  - integration simplicity 9
  - interactions with services and registries 10
  - internally focused 22
  - J2EE APIs 27
  - management
    - via Portal Server 147, 150
  - personalization
    - via Portal Server 143, 146, 148
  - phases of adoption 21
  - Portal-based Core Web services 203
  - presentation
    - via Portal Server 144, 148
  - relationship to J2EE Connector architecture 51
  - security 53
    - via Portal Server 146, 150
  - Sun ONE support 52–53
  - using from Sun ONE 52
- Web services. See also Core Web Services.
- Web single sign-on 170
- WebActions
  - iPlanet Application Framework (iAF) 140
- X
- XML
  - Java APIs 27
- XML information services and device interaction 157