

# The Cortex Project: Reliable Messaging for Web Services

A Technical White Paper  
September 2002



# Table of Contents

<b>Chapter 1</b>	<b>Executive Summary</b> . . . . .	1
<b>Chapter 2</b>	<b>Introduction and Overview</b> . . . . .	3
	Technologies . . . . .	5
	Web Services . . . . .	8
	Messaging Overview . . . . .	9
<b>Chapter 3</b>	<b>Scenarios</b> . . . . .	11
	Deliverables . . . . .	11
	Operational Scenarios . . . . .	11
	Point-to-Point Asynchronous Without Response—Scenario 1 . . . . .	12
	Point-to-Point Asynchronous with Response—Scenario 2 . . . . .	12
	Publish-and-Subscribe Asynchronous without Response—Scenario 3 . . . . .	13
	Security . . . . .	13
	Application Level . . . . .	13
	Service Level Requirements . . . . .	13
	Key Design Decisions . . . . .	13
	Target versus URL . . . . .	13
	SOAP Header Elements . . . . .	14
	In-Order Delivery . . . . .	14
	JNDI . . . . .	14
	Message Acknowledgements . . . . .	15
	Topics . . . . .	15
	Configuration Files . . . . .	15
	Additional Design Considerations . . . . .	15
<b>Chapter 4</b>	<b>Architecture</b> . . . . .	17
	Operational Overview and Model . . . . .	17
	Inbound Bridge . . . . .	19
	Outbound Bridge . . . . .	19
	HTTP Messages . . . . .	19
	JMS Messages . . . . .	19
	Interoperability . . . . .	20

Operation . . . . .	20
Messaging Scenario 1 . . . . .	20
Messaging Scenario 2 . . . . .	21
Messaging Scenario 3 . . . . .	23
Interoperability . . . . .	24
JAXM Messaging Package . . . . .	25
Failures and Recovery . . . . .	25
Inbound Bridge Down . . . . .	25
Outbound Bridge Down . . . . .	25
JMS Message Broker Down . . . . .	26
Target Web Service Down . . . . .	26
JNDI Compatible Directory Server Down . . . . .	26
Administration . . . . .	26
Overview . . . . .	26
Inbound Bridge . . . . .	27
Outbound Bridge . . . . .	28
JNDI . . . . .	29
JMS . . . . .	29
Tuning Tools . . . . .	29
<b>Chapter 5 Conclusion . . . . .</b>	<b>31</b>
For More Information . . . . .	32
Sun Open Net Environment (Sun ONE) . . . . .	32
Sun ONE Architecture Guide . . . . .	32
Sun ONE Messaging Queue . . . . .	32
Sun ONE Integration Server . . . . .	32
Sun ONE Directory Server . . . . .	32
Sun ONE Web Server . . . . .	32
Java API for XML Messaging (JAXM) . . . . .	32
JAXM Tutorial . . . . .	32
<b>Appendix A Company Information . . . . .</b>	<b>33</b>
About Sun . . . . .	33
Sun ONE Integration Server, EAI Edition . . . . .	34
Sun ONE Message Queue . . . . .	35
Sun ONE Directory Server 5.1 . . . . .	35
Sun ONE Web Server 6 . . . . .	35
Solaris Operating Environment . . . . .	35
About ThoughtWorks . . . . .	36
<b>Appendix B Example Code: Cortex's JAXM Provider Connection API . . . . .</b>	<b>37</b>

## Chapter 1

# Executive Summary

The Cortex project is part of an ongoing Web services development effort at TransCanada PipeLines, Limited, one of North America's largest energy companies. This transnational company owns and operates a 38,000-kilometer (24,000-mile) natural gas pipeline network. As a pioneer of independent power production, TransCanada owns, controls, or is constructing a total of approximately 2250 megawatts of electrical power. TransCanada works with hundreds of energy suppliers, traders, and purchasers. Historically, paper has been the main mechanism of interaction between TransCanada and its constituents. In an effort to automate and streamline core business processes, TransCanada is creating a Web services infrastructure.

The overall project — Service Provisioning Infrastructure for a Network Environment (SPINE) — will be a system of distributed, loosely based, and process-centric Web services built on a standards-based foundation. SPINE's goal is to create a flexible, scalable, open, and reliable Web services infrastructure that can be used throughout the company to lower costs and reduce time-to-market response for business solutions.

SPINE is comprised of many components. The Cortex project provides the messaging capabilities for an integrated gas management application, codenamed Dovetail. Dovetail will provide users inside and outside the company with a single integrated application set, consisting of multiple, independently developed gas management services running in a distributed environment. Cortex provides reliable messaging facilities within this set of services.

The goal of Cortex is to offer an interoperable and reliable enterprise messaging capability. This will be delivered as a utility service that can be leveraged by other applications and services. At TransCanada, Cortex deliverables will be used by many applications to provide document-oriented messaging in a consistent, platform-independent manner.

Cortex enables disparate applications to communicate by passing XML documents. It is capable of reliable message delivery in synchronous and asynchronous messaging, as well as publish-and-subscribe messaging environments. Cortex is interoperable with other applications and systems through a Simple Object Access Protocol (SOAP) over HTTP protocol, using an XML message format.

Cortex delivers a rich set of capabilities. Software products from the Sun™ Open Net Environment (Sun™ ONE) platform provide underlying functionality for both messaging and workflow. Using the Sun ONE Message Queue software product, Cortex offers:

- A reliable messaging service that adapts messages from different clients, such as those developed with Java™ technology, Web applications, and Microsoft .Net, adapting them to a Java Message Service (JMS) API using the Cortex bridge. This enables disparate applications to use a common messaging environment.

- Guaranteed delivery using either asynchronous call back or synchronous messaging modes. This functionality enables Cortex to be used in all Dovetail applications. In addition, it is capable of working through firewalls, allowing Web services to be used inside and outside the enterprise.

The Sun ONE Integration Server provides process integration and workflow capabilities. It can be used with Cortex to provide:

- Workflow functionality. Messages can be routed according to a set of predetermined rules; for example, a transaction report can be routed through a credit department for review before being compiled into a monthly report by accounting.
- Notification and exception capabilities. If a message matches a certain set of rules, users or process can be notified, or the message can be rerouted.

Cortex offers a number of advantages:

- It offers increased responsiveness to TransCanada and its suppliers and partners, enabling a rapid approach to integrating new services. Its open architecture offers direct access points to TransCanada's systems. Customers and partners can integrate their systems and design new applications without vendor lock-in.
- Once fully operational, Cortex is expected to reduce costs by offering a consistent, standards-based approach. A comprehensive, integrated, and reliable messaging system will minimize the use of "one-off" messaging systems.
- Cortex is scalable to meet growing business needs. The Cortex team expects wide adoption beyond Dovetail.
- Cortex is designed to meet the growing demand for availability. In an effort to reduce costs and increase responsiveness, TransCanada is opening more of its systems and applications to customers and partners. These users want 24x7x365 availability, and Cortex was designed to address this environment.

Cortex is a joint development project between TransCanada and Sun Microsystems, with integration services provided by ThoughtWorks. The project leverages the principles of the Sun ONE architecture, which helps businesses create Web services while leveraging existing IT applications, data, and systems. TransCanada's primary goal is to deliver a production-ready implementation of a messaging service. Additionally, Sun's goal is to deliver a reference implementation and portable demonstration for its iForce<sup>SM</sup> Ready Centers.

This paper contains a technical description of how the Cortex reliable messaging service works, including an architectural overview and a description of each of the Cortex subsystems.

An overview of the technologies and design criteria used in creating the Cortex reliable messaging service, as well as a technical discussion of the operation, products, and technologies used in this project, are also included.

- The first section offers a technical overview of the standards used in Web services, as well as an overview of the Sun ONE platform.
- The second section outlines the various scenarios in which the Cortex message service will operate, as well as key design considerations in developing Cortex.
- The third section provides an operational overview of Cortex. This includes details on each of the major subsystems, as well as a discussion of failure capabilities, interoperability, and administrative functions.
- The appendix contains additional information about the companies that supplied resources to this project, plus sample code.

## Chapter 2

# Introduction and Overview

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, and be available across an organization's many communities—employees, partners, customers—as well as a 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 that can 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:

- **Local applications:** These run on dedicated PCs and include popular office applications. Some are LAN-based, as well.
- **Client-server applications:** These are hosted by a large server and are often business-critical—accounting, human resources, manufacturing—requiring large database back ends. Client-server applications use both proprietary and Web-based front ends.
- **Web applications:** Dedicated, single-function applications that run over the Web, such as e-mail and calendar, these often require dedicated client software or are written with a Java technology front end so they can be accessed from any browser. Web applications can be used in house, outsourced from an application service provider (ASP), or a combination of the two.
- **Web services:** These run over the Web and can combine with other services to create a more useful or powerful solution. Web services offer modular, well-defined, and encapsulated functions, used for loosely coupled integration between applications or systems.
- **Operating environment services:** These provide scalability and reliability for clustering, dynamic reconfiguration, and high availability capabilities. They are typically provided by an operating system such as Sun's Solaris™ Operating Environment (OE).

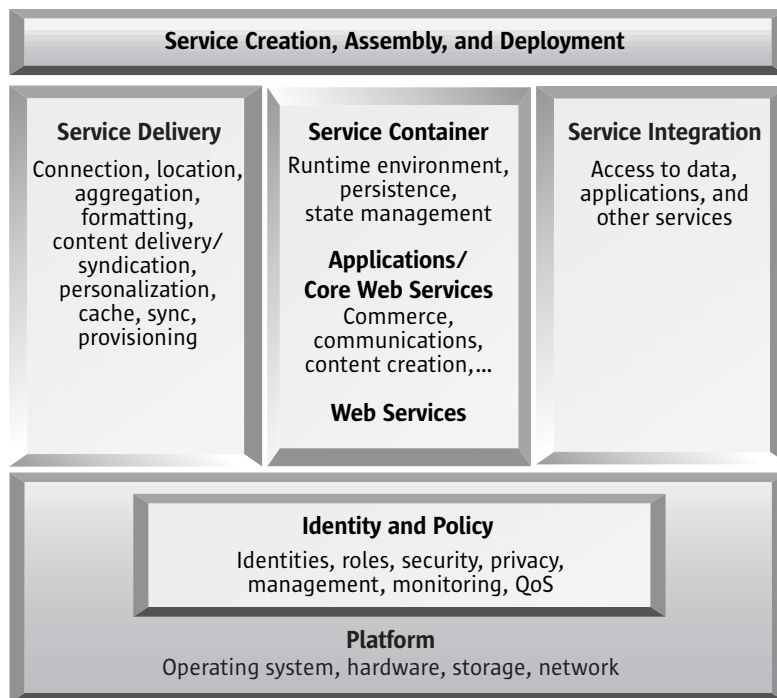
Sun uses the term *Services on Demand* to describe how enterprises use IT environments to transact and report business operations and communicate with others—anywhere, anytime, on any device. The Services on Demand concept is the foundation for modular, flexible, integratable, and automated access to digital assets, including computing resources.

The Services on Demand vision is of a comprehensive framework, encompassing traditional Net-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).

This vision represents evolution, not revolution—these new services will not supercede other network and development approaches. 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.

Sun provides a comprehensive product offering for delivering applications and services on the Sun ONE platform. For companies that need solutions that are sold, integrated, and supported worldwide, Sun delivers a complete set of products and service offerings.

An important aspect of the Sun ONE platform is that it is integratable. Because the Sun ONE architecture is based on open standards, companies can interoperate with their existing systems now, and with new add-ons in the future. Companies can also choose the components and systems they want to use, taking advantage of the open Sun ONE architecture. Developers can integrate the technologies they need to optimize deployment of their solution sets. Vendors are now offering tools and technologies that will reduce the cost, risk, and complexity of moving to this new model.



**Figure 1. Functional View of Sun ONE Platform**

The Sun ONE platform is the basis for scalable, reliable, and open standards-based Web applications today and Services on Demand tomorrow. The architecture encompasses (as shown in Figure 1, from the bottom up):

- OS, hardware, storage, and networking platform: Includes the directory technology necessary to define users, subscribers, organizations, and policy.

- Presentation, business logic, and data access:
  - Service Delivery is the presentation layer. In this architecture, the portal server delivers services to any device, aggregating content and providing security, personalization, and knowledge management.
  - The Service Container provides business logic—where the Web services run—utilizing an application server built on Java 2 Platform, Enterprise Edition (J2EE™) technology. The Service Container contains prebuilt Web services that an enterprise may build or buy, often hosted by an existing commerce or communications application.
  - Service Integration is the data access layer. It integrates enterprise assets such as business-to-business (B2B) and EAI applications, legacy applications, and database repositories.
- Tools for creating, assembling, deploying, and testing services.

The Sun ONE Architecture offers technology, products, and standards for Services on Demand, as well as superior integration across the platform, and defined interoperability with legacy systems and software provided from third-party vendors. The Sun ONE architecture offers these capabilities:

- Web applications middleware, including the standard messaging and Web server capabilities available today
- Web services middleware, which provides new features for basic XML Web services
- Identity and context (knowledge about user preferences, intentions, and goals) services
- Web client model, which defines how applications built with Java technology can be provisioned to portable devices and desktops, utilizing Java 2 Platform, Micro Edition (J2ME™) and desktop Java technologies
- Other middleware interfaces and core Web services currently provided as product interfaces: directory services, portal services, security and policy, e-commerce services, and communications capabilities
- Java and Web development tools, including the ability to wrap legacy languages for Web services
- Systems and applications management for Web applications and Web services
- Platform services relevant to Web applications and Web services containers

The Sun ONE architecture represents an integratable stack, based on open standards for APIs and protocols. It is strongly aligned to standard Web interfaces and bases its interoperability strategy on them. As demonstrated in this paper, best-of-breed products can be integrated into an overall Web service solution.

## Technologies

In addition to mature Web protocols such as HTTP, HTML, and SSL, a number of emerging technologies support the new Web services model. While not all are fully defined standards, they are maturing quickly with broad industry support. These key technologies are:

- The Extensible Markup Language (XML) describes a class of data objects called XML documents which are stored on computers, and partially describes the behavior of programs that process these objects. The goal of XML is to enable standardized data to be served, received, and processed on the Web in the way that is now possible with HTML.

- Simple Object Access Protocol (SOAP) provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. By itself, SOAP does not define any application semantics such as a programming model or implementation-specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems and software application environments.

SOAP consists of three parts:

- The SOAP envelope construct defines an overall framework for expressing what is in a message, who should deal with it, and whether it is optional or mandatory.
  - The SOAP encoding rules define a serialization mechanism that can be used to exchange instances of application-defined data types.
  - The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.
- Web Services Description Language (WSDL) is an XML format for describing network services as a set of end points operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, then bound to a concrete network protocol and message format to define an end point. Related concrete end points are combined into abstract end points (services). WSDL is extensible to allow the description of endpoints and their messages, regardless of what message formats or network protocols are used to communicate.
  - Universal Description, Discovery, and Integration (UDDI) is a specification for distributed Web-based information registries of Web services. UDDI is also a publicly accessible set of implementations of the specification that allows businesses to register information about the Web services they offer, so that other businesses can find them. The core component of the UDDI project is the UDDI business registration, an XML file used to describe a business entity and its Web services.

Conceptually, the information provided in a UDDI business registration consists of three components:

- White pages, including address, contact, and known identifiers
  - Yellow pages, including industrial categorizations based on standard taxonomies
  - Green pages, the technical information about services that are exposed by the business. Green pages include references to specifications for Web services, as well as support for pointers to various file- and URL-based discovery mechanisms, if required.
- Electronic Business XML (ebXML) is a worldwide project to standardize the exchange of electronic business data. The fundamental goal of ebXML is to create a single global electronic market. This is accomplished by developing one set of internationally agreed-upon specifications. ebXML is a complete, B2B framework that enables collaboration through the sharing of Web-based services. The framework supports the definition and execution of B2B processes expressed as choreographed sequences of service exchanges. Transports used to send XML messages between enterprises may utilize SOAP (or extensions to SOAP) provided by ebXML. The ebXML Messaging transport enables a message to be sent securely and reliably without the intervention of a programmer.

- Java technologies (based on the Java programming language) have been at the center of Web development for many years, from back-end servers to consumer devices. Many Services on Demand are enabled by Enterprise JavaBeans™ (EJB™), J2EE, Java Servlet, and JavaServer Pages™ (JSP™) technologies. Emerging Java specifications, known as Java APIs for XML, are enabling the creation of Services on Demand using familiar JSP and EJB components for the Java platform. These include technologies and protocols for working with XML, directories, repositories of services, and messaging, as well as transitioning existing services.

In addition, the Cortex project uses these key Java technologies:

- The J2EE Platform is an environment for developing and deploying enterprise applications. The J2EE platform consists of a set of services, applications, APIs, and protocols that provide the functionality for developing multitiered, Web-based applications. J2EE technology is designed for the mainframe-scale computing typical of large enterprise. As such, it simplifies application development and decreases the need for programming and programmer training by creating standardized, reusable modular components and by enabling the J2EE container to handle many aspects of programming automatically.
- The Java API for XML Messaging (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. 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 needs to contain an attachment part. There is no limitation on content in the attachment part, allowing images or any other kind of Multipart Internet Mail Extension (MIME) encoded content to be sent.
- The Java Naming and Directory Interface™ (JNDI) API provides access to a naming environment. It also offers methods for performing directory operations, such as associating attributes with objects and searching for objects using their attributes.
- The Java Message Service (JMS) is a standard API for messaging that supports reliable point-to-point messaging as well as the publish-subscribe model. The JMS API 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 intracompany operations. They allow separate business components to be combined into a reliable yet flexible system. The asynchronous reliable messaging component of the Sun ONE architecture is built upon the JMS API specification.

### Web Services

The term *Web services* now has a specific meaning that has been endorsed by industry vendors and analysts. Web services are self-describing components that can discover and engage other Web services to complete complex tasks over the Internet. Unlike hard-wired applications—for example, desktop or client-server applications—Web services are loosely coupled, and can dynamically locate and interact with other components on the Internet to provide a service.

A Web service typically passes an XML message. This can be accomplished synchronously in a remote procedure call style, or asynchronously, in a reliable messaging passing style. The main standard for RPC is SOAP. The reliable messaging style may utilize a message bus such as JMS or the emerging ebXML messaging service.

In Figure 2, a developer tool is accessing a UDDI registry to see what services are available. The registry could be a public or private enterprise registry. The tool browses the registry by sending XML messages in the SOAP protocol. Within the registry is the description of available services, as well as entries written in WSDL describing services and interfaces. This model is called *static lookup*, because service discovery is performed at development time.

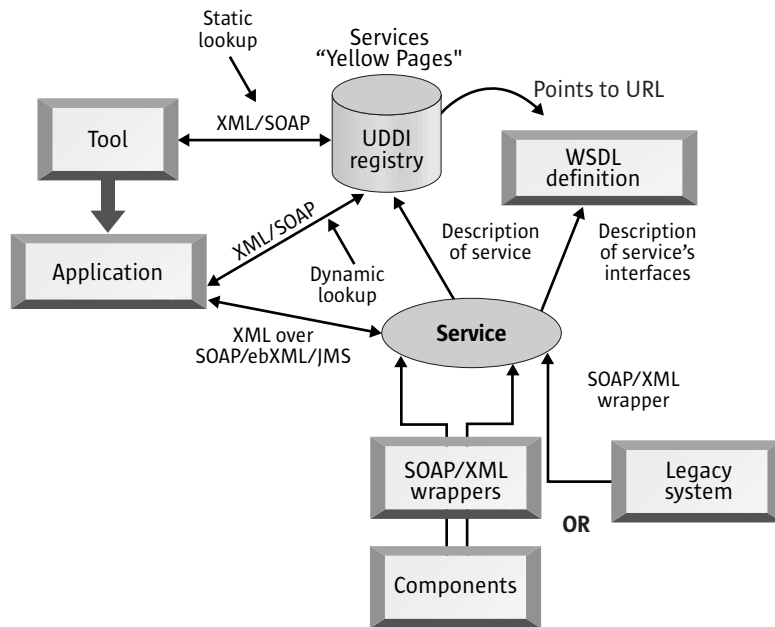


Figure 2. Functional View of Web Services

Sun recommends a phased approach when adopting Web services, as documented in the *Sun ONE Architecture Guide*. It has been observed that many companies incrementally adopt various Web services technologies. Often, companies start with XML, then use SOAP and WSDL technologies to create static Web services. Later, UDDI is implemented to enable more dynamic services. TransCanada adopted this approach in the evolution of their Web service technology-based systems.

## Messaging Overview

Messaging is a method of communication between software components or applications. A messaging system is a peer-to-peer facility. A messaging client can send messages to, and receive messages from, any other client. Each client connects to a messaging system that provides facilities for creating, sending, receiving, and reading messages.

Messaging enables loosely coupled distributed communication. A component sends a message to a destination, and the recipient can retrieve or receive the message from the destination. However, the sender and receiver do not have to be available at the same time in order to communicate. In fact, the sender does not need to know anything about the receiver; nor does the receiver need to know anything about the sender. The sender and the receiver need to know only what the message format is and what destination to use.

Note that messaging differs from tightly coupled technologies, such as remote method invocation (RMI), which require an application to know a remote application's methods. Messaging also differs from e-mail, which is a method of communication between people (or software applications and people), because messaging enables communication between software applications or components.

The Cortex model supports the traditional point-to-point mode of operation, as well as publish-and-subscribe messaging.

In the Cortex model, the client delegates message delivery to the message broker. The Web service client is only interested in understanding that message delivery is guaranteed by the broker. This is accomplished by receiving a positive acknowledgement when a message is issued.

If the client is interested in the ultimate completion status of the request received by the target Web service, an out-of-context message must provide a response to the client. This return message is initiated by the target service. It is the responsibility of the client to interpret the message and tie it back to its original context, if required.

## Chapter 3

# Scenarios

The purpose of the Cortex project is to deliver an interoperable and reliable messaging service, including a reusable enterprise messaging solution that provides for asynchronous exchange of data. By promoting the use of a simple, consistent approach to application interaction, the development team expects to simplify enterprise development and enable loosely coupled, reliable, asynchronous interactions among applications.

The Cortex project is developing a production-ready system that delivers an enterprise messaging platform to:

- Create the capability to replace multiple and proprietary messaging services
- Reduce license costs
- Lower costs associated with implementation and maintenance

The enterprise messaging system takes the form of a Web service that is complete and flexible enough to handle an organization's messaging needs. The service level must, at a minimum, be equal to that of the enterprise application using the system.

## Deliverables

The Cortex messaging service includes a number of components which are described in this document, including a JMS broker implementation, LDAP data store, JAXM providers, HTTP bridges, and an API, as well as administration tools to configure and support the messaging service were created.

A demonstration of source and target applications that show the capabilities of the Cortex messaging service is available in Sun's iForce Ready Centers.

## Operational Scenarios

The Cortex team worked with TransCanada's IT Business Solution delivery teams to identify different types of messaging scenarios. These scenarios address the anticipated messaging requirements that will be encountered in the TransCanada deployment.

In each of the scenarios, a synchronous relationship is at the heart of the client-to-message service level. When the client talks to the Cortex bridge, an immediate acknowledgement (ACK) is expected. This response is received from the message service and has no meaning at the application level because interaction there is asynchronous.

In the descriptions that follow, the annotated sequencing follows a convention. The numbers represent the chronological series of events, and the letters represent differing threads of execution.

**Point-to-Point Asynchronous Without Response— Scenario 1**

In Figure 3, the client application sends a message to Cortex, which delivers it to the target Web service. This scenario represents asynchronous “fire and forget,” where the client fires off a message and forgets about it. The only reply the Web service client receives is the synchronous ACK response from Cortex.

If the messaging service acknowledges the client’s request, the client is guaranteed that the message will be delivered to the target Web service. In other words, if the target Web service is down at the time of the client’s request, it is of no concern to the client. Note that the target Web service is free to be intermittently connected, if required.

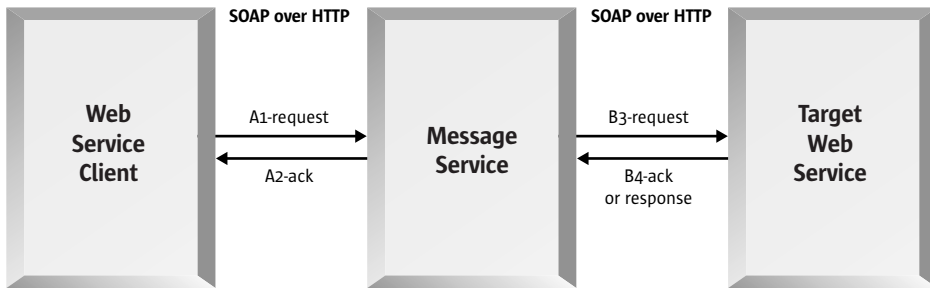


Figure 3. Messaging Scenario 1

**Point-to-Point Asynchronous with Response— Scenario 2**

In Figure 4, the scenario handles complete asynchronous communication to and from the client and the target Web service. A message is sent out through Cortex, and a response is received by the Web service client at a later time.

This scenario is applicable when the target Web service takes a long time to respond to a request, potentially several hours. For example, human intervention or a large transaction may be required before a response can be sent. The messaging service is responsible for guaranteeing that the request is received by the target Web service, a response is sent by the target Web service, and the Web service client receives the response.

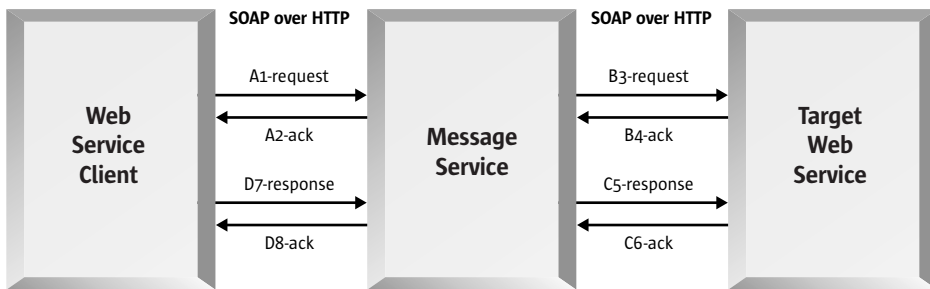


Figure 4. Messaging Scenario 2

### Publish-and-Subscribe Asynchronous without Response— Scenario 3

In the scenario illustrated in Figure 5, a SOAP message is published to multiple subscribing Web services by a client, which sends a single message that is delivered to all of the services at once. Note that the publish-and-subscribe scenario is usually applicable when a response is not expected.

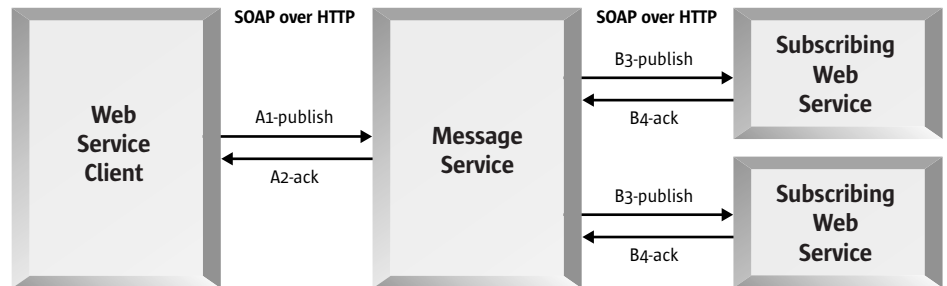


Figure 5. Message Scenario 3

## Security

### Application Level

Cortex does not attempt to address the issue of security in the application layer, as overall security is handled by a centralized policy server. In addition, several layers of infrastructure security are in place:

- **Operating system:** Protected by user accounts
- **Sun ONE Message Queue:** Native MQ authentication and authorization
- **LDAP:** Protected by user accounts

It is anticipated that a management tool delivered by subsequent Web services efforts in TransCanada will address security issues that are specific to Web services.

### Service Level Requirements

The Cortex service level is anticipated to be business critical, and therefore should be prepared for a 24x7 support period. The availability of the messaging service will be equal to at least that required by the current suite of enterprise applications and services.

## Key Design Decisions

The Cortex development team made a number of key architectural decisions over the course of the project, which are described below.

### Target versus URL

Cortex has introduced the concept of a target, which is declared within the header of a SOAP message, and used by the inbound bridge to resolve the JMS destination name for each message. By default, every JMS destination name that the inbound bridge is aware of resolves to an equivalent target name. In addition, a target can also act as an alias for a JMS destination name. In this manner, the target places a layer of indirection between the client and Cortex's actual JMS destination configuration, insulating the client from changes in target destination addresses.

A message placed onto a JMS destination may be picked up by one of the outbound bridge's consumers, and posted to a target Web service's URL. While the concept of a target may seem to introduce a layer of complexity, — why not declare the target Web service's URL directly in the SOAP message's header? — there are a number of reasons why it is undesirable to specify a target Web service's URL directly within a SOAP message. These include:

- To support any URL, Cortex would have to support the dynamic creation of JMS destinations to provide *in-order* reliable messaging. Since the JMS specification does not address dynamic creation or configuration of JMS destinations, Cortex would have to use vendor-specific API calls, which would have been against its vendor-independent design goal.
- Specifying the URL directly excludes the possibility of the outbound bridge performing a lookup of the desired target Web service's URL.
- Interoperability with pure JMS clients would have been compromised, as a URL has no meaning for JMS clients.

### SOAP Header Elements

In an earlier phase of the SPINE initiative, a service broker was developed to support asynchronous reliable messaging. The service broker expected SOAP messages intended for target Web services to be embedded within an outer SOAP message intended specifically for the service broker. Cortex, on the other hand, is more closely aligned with the industry's direction of using a single SOAP message while populating the header elements within the message. As a result, Cortex acts as a transparent proxy to target Web services.

Currently, Cortex recognizes two SOAP header elements.

- **Target:** The SOAP header element that is required by the inbound bridge is the target.
- **Message ID:** The next SOAP header element that is recognized is the unique message ID. This is a useful artifact, performing a critical role in correlating responses with requests, ensuring “exactly once” message delivery, and enabling the logging and tracing of messages as they are delivered. Note that the only current usage of a message ID within Cortex is to correlate response messages with request messages.

### In-Order Delivery

Not only does Cortex deliver messages to target Web services in a reliable manner, but it also delivers them in the order in which they were received. In the absence of business requirements that dictated otherwise, this design decision was the most logical one.

### JNDI

Cortex's inbound and outbound bridges fetch their configuration out of a JNDI compatible directory server. This was done so that the bridge's Web archives (WAR files) are configuration independent. In other words, the WAR files do not have to be rebuilt when the bridge's configuration changes.

In addition, Cortex's use of JNDI supports the recommended convention of retrieving JMS connection factories and JAXM provider connection factories out of JNDI.

### Message Acknowledgements

In the absence of a standard mechanism within the SOAP specification to acknowledge the receipt of SOAP messages, the Cortex project team adopted the convention that an *acknowledgement* is any SOAP message that does not contain a SOAP fault. This convention has the added benefit that legacy Web services will likely be able to acknowledge requests from Cortex with no code changes.

### Topics

Currently, Cortex relies solely on JMS topics in order to support the described messaging scenarios.

Scenario 2 and Scenario 3 required JMS topics, as more than one consumer would receive messages from the JMS destination. However, Scenario 1 could also be implemented with JMS queues or topics.

The project team envisions that Cortex will support both JMS queues and topics as business requirements dictate.

### Configuration Files

Since the inbound bridge may be physically deployed on a different machine than the outbound bridge, it was also decided to break apart Cortex's configuration into two separate configuration files: inbound and outbound. This method is especially useful when configuration tasks would impact only the inbound or outbound bridge, because it does not require a total system outage.

### Additional Design Considerations

A number of other important design considerations that surfaced during construction are described below.

#### *Exactly Once Message Delivery*

Due to the nature of HTTP, any messaging service based on HTTP is unable to ensure *exactly once* message delivery without the cooperation of both the client and the target Web service.

Exactly once message delivery means that a message is received by the target Web service once and only once. *At-least-once* message delivery means that a message is received one or more times by the target Web service.

Currently, Cortex supports at-least-once message delivery out of the box. As previously mentioned, exactly once message delivery is achievable with participation from the client and the target Web service. The key to achieving exactly once message delivery is the use of a unique message ID. In a nutshell, a client ensures that all outgoing messages have a unique ID. On the other hand, a target Web service performs a check each time that it receives a message, and behaves appropriately if it has received the same message already. The target Web service is responsible for caching the message's ID once it is successfully processed, and can also cache the response.

### *Location Independence*

Although target Web service location independence is not currently implemented, Cortex supports this functionality. By abstracting a target Web service's URL through the use of the target, the outbound bridge is free to perform any type of lookup (using a UDDI registry, for example) that may be required. Note that from the perspective of a Web service client, Cortex provides application-level location independence of the target Web service.

### *Translation and Transformation of Messages*

Translation and transformation of SOAP messages may occur at many points within a messaging environment. These include:

- Within the client, prior to sending the message
- Within the target Web service, prior to processing the message
- Within the target Web service's runtime container, prior to the target Web service receiving the message
- Within an intermediary middleware product such as a workflow management or integration server product (for example, the Sun ONE Integration Server)
- Within Cortex. Note that Cortex currently supports only transformation between JMS message and SOAP message formats. It does not perform translation of message content (through XSLT), as other options are available, as listed above.

## Chapter 4

# Architecture

Cortex is designed to provide a reliable SOAP messaging service using the Sun ONE Message Queue and Sun ONE Directory Server products. The Sun ONE Integration Server, while not required, may be deployed for additional functionality. Cortex is built with servlets, and uses the JMS and JNDI APIs. SOAP functionality is enabled by the JAXM protocol.

## Operational Overview and Model

Logically, Cortex may be grouped into two components, the inbound and outbound bridges.

The responsibility of the inbound bridge is to accept SOAP messages via HTTP, convert them into JMS messages, and transfer them onto a JMS destination. The inbound bridge's configuration is stored within a JNDI compatible directory server.

The responsibility of the outbound bridge is to receive JMS messages from a JMS destination, convert them into SOAP messages, and send them to target Web services via HTTP. The outbound bridge's configuration is stored within a JNDI compatible directory server.

The only integration between the inbound bridge and the outbound bridge is via JMS. A JMS compatible message broker is used because it provides reliable messaging out of the box. Cortex components were created to provide SOAP over HTTP gateways into and out of JMS.

Figure 6 provides a functional view of Cortex's architecture, starting at the upper left side:

1. Messages come in from clients in SOAP format over HTTP to the inbound bridge, which is a JAXM servlet. This servlet is configured with the inbound configuration file, which is in XML format and retrieved from the JNDI compatible directory server.
2. The JAXM servlet accepts the message, transforms it into a JMS bytes message, and forwards it to the JMS Message Broker. The JMS bytes message is placed onto a JMS topic.
3. The inbound bridge acknowledges receipt of the message back to the original client.
4. The JMS bytes message is received by a consumer within the outbound bridge, which has been configured to listen for messages on any number of JMS topics.
5. The outbound bridge takes the message out of the JMS wrapper and transforms it into a SOAP message.
6. The outbound bridge posts the message to the appropriate target, which is set by the outbound configuration file (also in XML format and retrieved from the JNDI compatible directory server).
7. The target acknowledges receipt of the message.

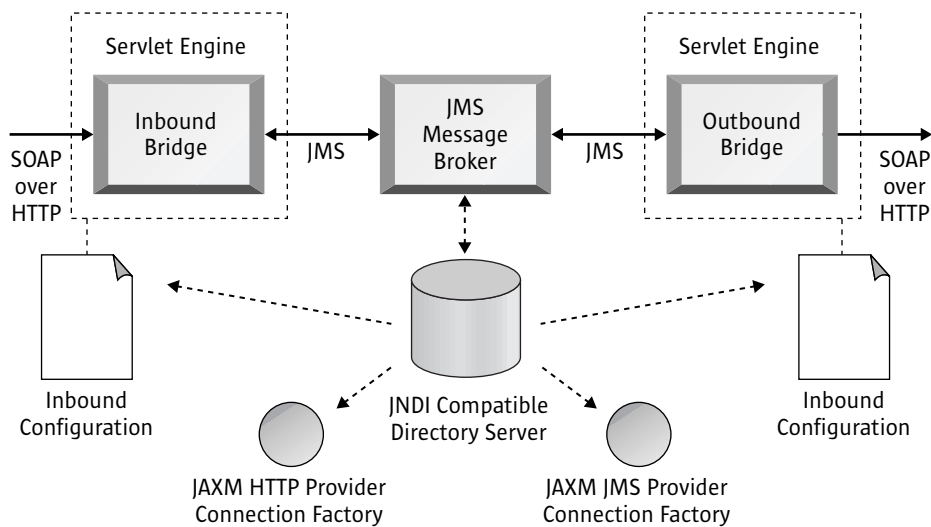


Figure 6. Functional View of Cortex Messaging

The connection factories at the bottom of Figure 6 abstract some of the detail of using the JMS API for developers, enabling the use of JNDI when sending SOAP messages. By using the JAXM Provider specification, Cortex is ready to implement ebXML when required.

This architecture offers two distinct technical advantages. The first is that clients (or other producer applications) do not need to know the physical address of any subscribers. Secondly, target addresses for all clients are message producers centrally administered at the outbound bridge.

Figure 7 provides more detail on the inbound and outbound bridges.

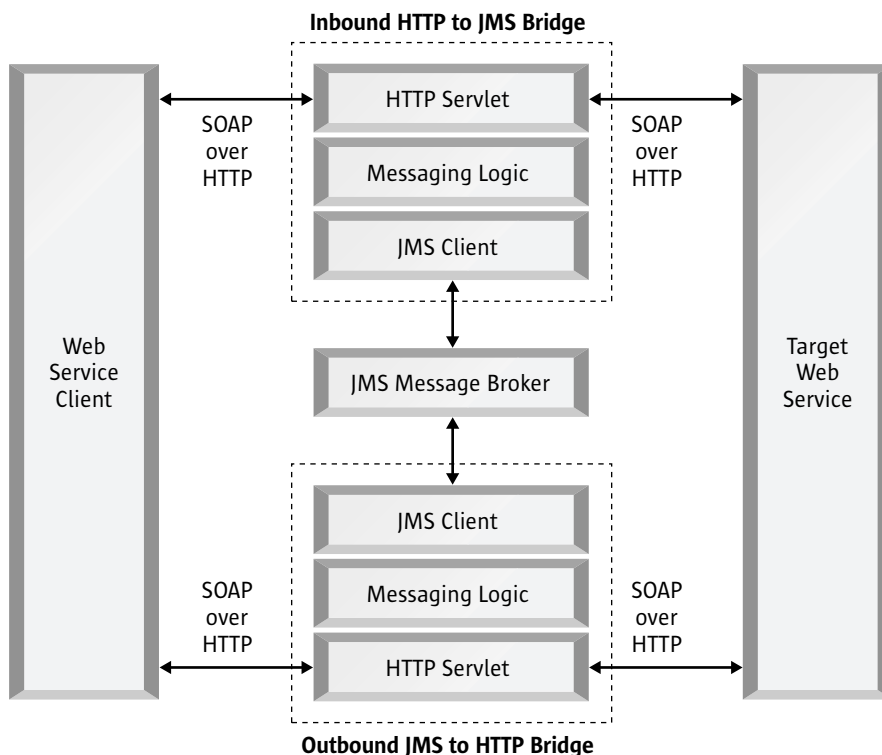


Figure 7. A Logical View of the Cortex

### Inbound Bridge

- The inbound bridge is composed of an HTTP servlet, messaging logic, and a JMS producer. A JMS producer is a JMS client that produces messages onto a JMS destination. Only one multithreaded JMS producer is used by the inbound bridge.
- Message logic parses the SOAP target using the configuration file.
- A JMS producer publishes the message to a JMS topic.

### Outbound Bridge

- The outbound bridge is composed of one or more JMS consumers, messaging logic, and HTTP clients. A JMS consumer is a JMS client that consumes messages off a JMS destination.
- The JMS consumer (client) picks up a message, and transforms it from a JMS message into a SOAP message.
- The message is posted to the target Web service.
- If the message expects a meaningful response, the outbound servlet can stamp the message with a correlation ID. This is done in case of a failure—to make it clear if the message has not been consumed.

It is noteworthy to recognize that the only integration between the inbound and outbound bridges is via JMS. This allows the inbound and outbound bridges to be stateless, and recover following a catastrophic failure.

### HTTP Messages

The inbound bridge is expecting an HTTP POST, and will accept messages that conform to the SOAP 1.1 message specification or the SOAP 1.1 message with attachments specification.

The only prerequisite of the SOAP message is that it contains a target within the SOAP header of the message. The following code sample highlights the target code:

```
<soap-env:Envelope xmlns:soap-env="http://schemas.xmlsoap.org/
  soap/envelope/"
  xmlns:spine="http://www.tcpl.ca/spine">
  <soap-env:Header>
    <spine:Target>sampleTarget</spine:Target>
  </soap-env:Header>
  <soap-env:Body>
    ...
  </soap-env:Body>
</soap-env:Envelope>
```

### JMS Messages

Once the inbound bridge has received a SOAP 1.1 message or a SOAP 1.1 message with attachments, it will proceed to create an equivalent JMS bytes message. A SOAP 1.1 message is easily transformed into a JMS bytes message by streaming the contents of the SOAP message directly into the bytes of a JMS bytes message.

A SOAP 1.1 message with attachments demands greater sophistication, as the SOAP message is no longer simply XML content, but is now a multipart MIME message. In order to parse this message, certain MIME headers are required. These are present within the HTTP headers when the SOAP message is posted to the inbound bridge. Thus, it is necessary to store these MIME headers into the JMS message as JMS properties.

Cortex provides the `JMSUtils` utility class that assists in the transformation of SOAP messages to JMS bytes messages and vice versa.

## Interoperability

Despite the existence of the inbound and outbound bridges, applications may still interface directly with JMS or in combination with the bridges. This ability provides tremendous flexibility when providing a Web services messaging infrastructure.

## Operation

### Messaging Scenario 1

Figure 8 illustrates point-to-point asynchronous without response:

- Client synchronous to messaging service
- Messaging service synchronous to target

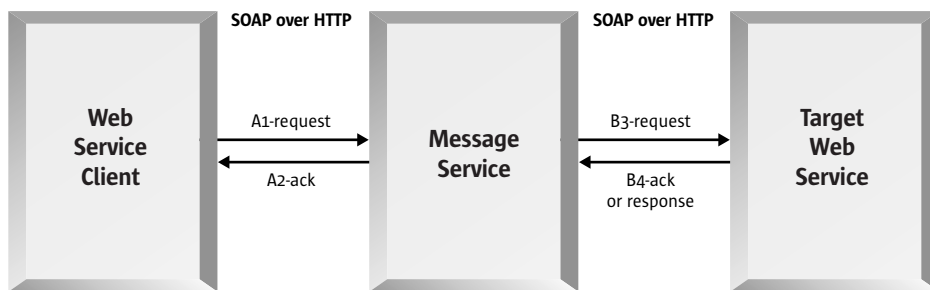


Figure 8. Messaging Scenario 1

1. A SOAP message is posted to the inbound bridge via HTTP.
2. The inbound bridge retrieves the target from the SOAP message.
3. The inbound bridge queries its configuration and maps the target into a configured JMS topic.
4. The inbound bridge transforms the SOAP message into a JMS bytes message.
5. The JMS bytes message is placed onto the JMS topic.
6. The inbound bridge acknowledges the original caller by responding with a SOAP message that does not contain a SOAP fault.
7. The JMS bytes message is received by a consumer (durable subscriber) within the outbound bridge that has been configured to listen for messages on the JMS topic.
8. The outbound bridge's consumer proceeds to transform the JMS bytes message into a SOAP message.
9. The outbound bridge's consumer posts the SOAP message to its configured target Web service URL.
10. A target Web service receives the request and executes the business logic necessary to fulfill it.
11. Upon completion, the target Web service responds with a SOAP message that does not contain a SOAP fault.
12. Upon receiving the acknowledgement, the outbound bridge's consumer commits the JMS bytes message that was originally received.

## Messaging Scenario 2

Figure 9 illustrates point-to-point asynchronous with response:

- Client asynchronous to messaging service
- Messaging service asynchronous to target

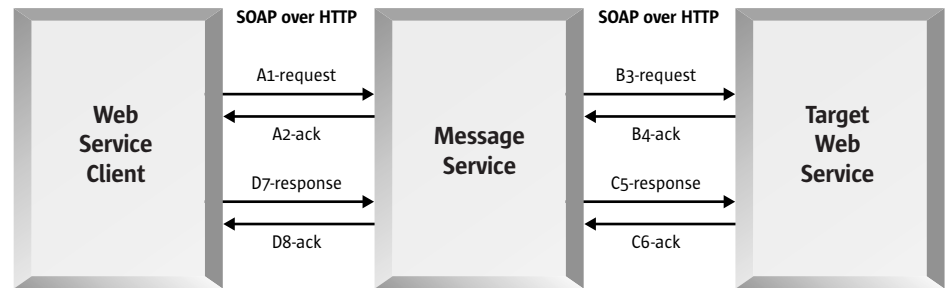


Figure 9. Messaging Scenario 2

1. A SOAP message is posted to the inbound bridge via HTTP.
2. The inbound bridge retrieves the target from the SOAP message.
3. The inbound bridge queries its configuration and maps the target into a configured JMS topic.
4. The inbound bridge transforms the SOAP message into a JMS bytes message.
5. The JMS bytes message is placed onto the JMS topic.
6. The inbound bridge acknowledges the original caller (client) by responding with a SOAP message that does not contain a SOAP fault.
7. The JMS bytes message is received by a consumer (durable subscriber) within the outbound bridge that has been configured to listen for messages on the JMS topic. The consumer has also been configured to listen onto a response topic for the existence of a response, and is also known as the *blocking* consumer.
8. The outbound bridge's consumer transforms the JMS bytes message into a SOAP message.
9. The outbound bridge's consumer sets the target on the SOAP message to correspond to the response target, which is the first of two pieces of information that the target Web service needs in order to respond.
10. In addition, the outbound bridge's consumer ensures that the SOAP message has a unique message ID, so that it can correlate the response with the request. The message ID is the second piece of information that the target Web service needs in order to respond. Note that if the SOAP message does not include a message ID upon receipt, then the JMS message ID is used as the SOAP message's unique ID. This is done so that in the case of a rollback, the same message ID will be used.
11. The outbound bridge's consumer posts the SOAP message to its configured target Web service URL.
12. A target Web service (developed to behave asynchronously) receives the request and immediately responds with a SOAP message that does not contain a SOAP fault.

13. The outbound bridge's consumer blocks for a configured amount of time, while listening for a response to arrive on the response topic. Not only must a message arrive, but the response message must also have the same message ID as the request SOAP message. In this manner, the blocking consumer can correlate the response with the request. Any response message that does not correlate is thrown away.
14. The target Web service executes any business logic necessary to fulfill the request.
15. Upon completion, the target Web service forms a response SOAP message destined for the inbound bridge.
16. Prior to sending, the target Web service extracts the target and message ID from the original request, and sets them on the response message.
17. The target Web service posts the response SOAP message to the inbound bridge via HTTP.
18. The inbound bridge retrieves the target from the response SOAP message.
19. The inbound bridge queries its configuration and maps the target into a configured JMS topic.
20. The inbound bridge transforms the response SOAP message into a JMS bytes message.
21. The JMS bytes message is placed onto the JMS topic.
22. The inbound bridge acknowledges the target Web service by responding with a SOAP message that does not contain a SOAP fault.
23. Upon receiving the acknowledgement, the target Web service's responsibilities are complete.
24. Since the response message has now been placed onto a topic and it correlates with the request, the blocking consumer no longer blocks.
25. As a result, the blocking consumer may commit the JMS bytes message that was originally received.
26. Meanwhile, an additional callback consumer has been configured to deliver the response to the client. The callback consumer delivers and commits the JMS bytes message received (as described in Scenario 1).

### Messaging Scenario 3

Figure 10 illustrates the publish-subscribe asynchronous without response:

- Client synchronous to messaging service
- Messaging service synchronous to target

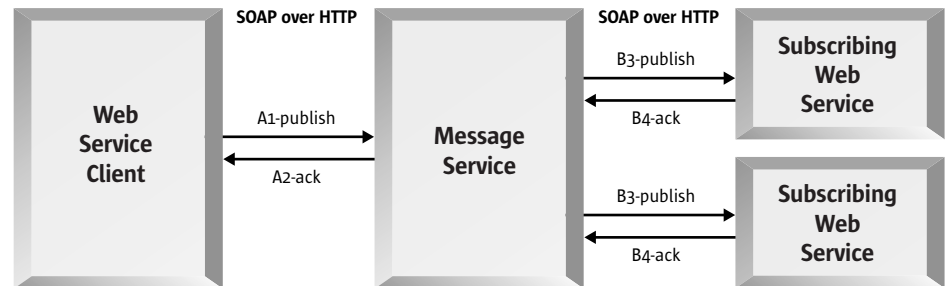


Figure 10. Messaging Scenario 3

1. A SOAP message is posted to the inbound bridge via HTTP.
2. The inbound bridge retrieves the target from the SOAP message.
3. The inbound bridge queries its configuration and maps the target into a configured JMS topic.
4. The inbound bridge transforms the SOAP message into a JMS bytes message.
5. The JMS bytes message is placed onto the JMS topic.
6. The inbound bridge acknowledges the original caller by responding with a SOAP message that does not contain a SOAP fault.
7. The JMS bytes message is received by one or more consumers (durable subscribers) within the outbound bridge which have been configured to listen for messages on the JMS topic.
8. The outbound bridge's consumers transform the JMS bytes message into a SOAP message.
9. The outbound bridge's consumers post the SOAP message to their configured target Web service URLs.
10. Each target Web service receives the request.
11. Each target Web service executes any business logic necessary to fulfill the request.
12. Upon completion, each target Web service responds with a SOAP message that does not contain a SOAP fault.
13. Upon receiving the acknowledgements, the outbound bridge's consumers commit the JMS bytes message that was originally received.

## Interoperability

As shown in Figure 11, Cortex provides reliable messaging capabilities for virtually any client application or Web services interface of application environments such as Web applications, Java technology applications, and .Net applications. A primary design goal of the inbound bridge is its ability to accept any standard SOAP 1.1 message or SOAP 1.1 with attachments message. In theory, this means that clients may use any SOAP 1.1-compliant message generation API when communicating with the inbound bridge. Applications and services can communicate using SOAP over HTTP, or directly with the JMS brokers using a JMS API.

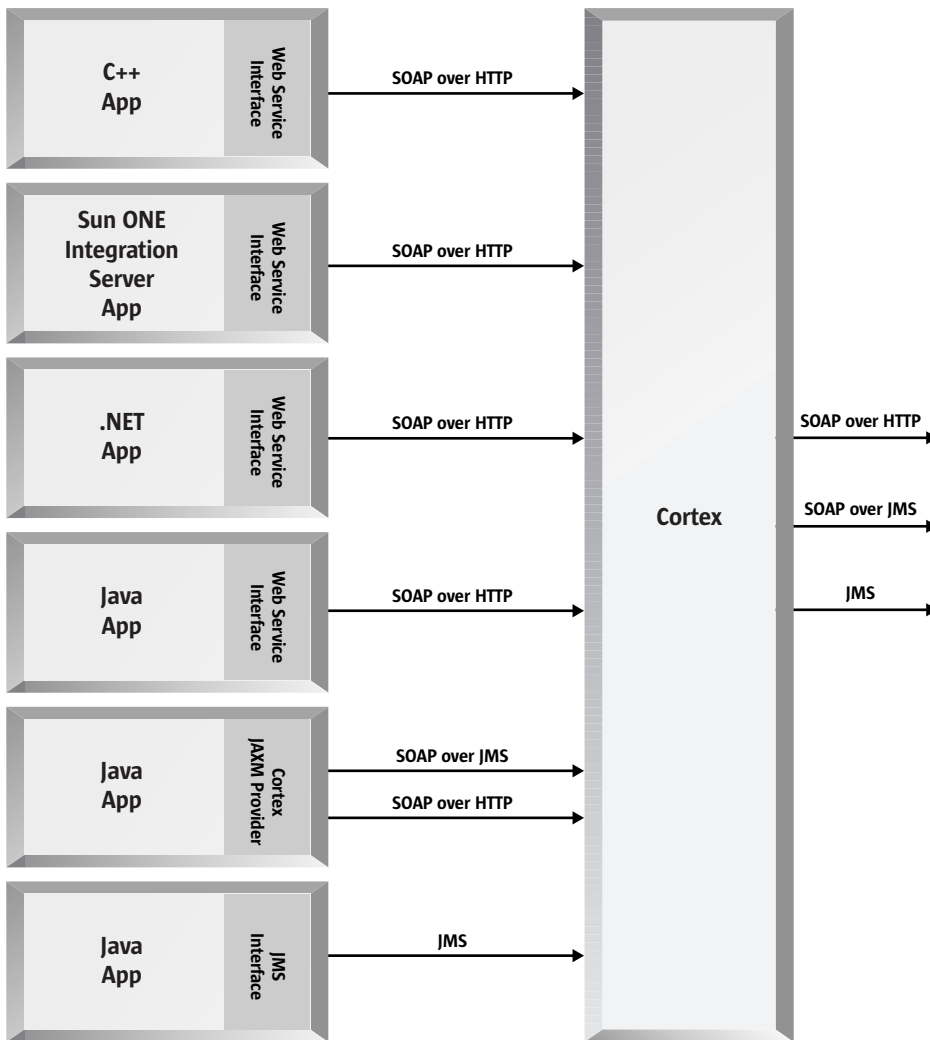


Figure 11. Cortex Provides Messaging for Enterprise Applications and Services

### JAXM Messaging Package

For Java technology services, the `javax.xml.soap` package (part of JAXM) contains the API for creating and populating a SOAP message. In addition, this package includes the API necessary for sending synchronous request-response messages.

The JAXM `javax.xml.messaging` package contains the API needed for using a messaging provider, providing the capability to send one-way messages. Cortex has developed two implementations of a JAXM provider connection:

- **JAXM HTTP provider connection:** Facilitates the creation and sending of SOAP messages over HTTP to the inbound bridge. Clients are not required to specify the URL of the inbound bridge, as it has been configured at runtime within the provider connection implementation.
- **JAXM JMS provider connection:** Allows clients to bypass the inbound bridge and send SOAP messages directly onto a JMS destination. Like the HTTP provider connection implementation, the only concern of clients is to specify the target.

## Failures and Recovery

A hallmark of a reliable messaging system is the ability to guarantee delivery of a message. If a message is not delivered, a reliable messaging system must detect the failure, try again to deliver the message, and provide notification as required.

This section briefly introduces common failure points. As is the nature of reliable messaging, if an acknowledgement is not received, Cortex assumes the message was not consumed, and tries to send it again. Five common failure points affect the end-to-end delivery of messages:

1. Inbound bridge down
2. Outbound bridge down
3. JMS message broker down
4. Target Web service down
5. JNDI compatible directory server down

### Inbound Bridge Down

If the inbound bridge is not running, clients will not be able to connect to Cortex via HTTP. Clients will likely receive an exception indicating their inability to connect to the servlet, or a SOAP fault indicating that the inbound bridge is not running.

### Outbound Bridge Down

If the outbound bridge has failed, clients sending messages to the inbound bridge or directly to JMS will be unaffected. Messages will collect within the JMS message broker until the outbound bridge is brought back online or is restarted.

### **JMS Message Broker Down**

A JMS message broker failure will result in the inability of both the inbound and outbound bridges from establishing connections to the broker. From a client perspective, the inbound bridge will return a SOAP fault. Naturally, no messages will be delivered via the outbound bridge until the JMS message broker is brought back online. When this occurs, the outbound bridge should automatically reconnect and proceed to deliver any outstanding messages.

### **Target Web Service Down**

If a target Web service is down, the outbound bridge will be unable to deliver messages to the target Web services. As a result, the outbound bridge will suspend delivery of messages to this service for a configured amount of time. The outbound bridge itself never sleeps, but the JMS consumer that is partnered with the target Web service does sleep. Multiple JMS Consumers are encompassed by the outbound bridge, so other JMS consumers will still be active.

After a specified time interval has passed, the outbound bridge will attempt to redeliver the message. This cycle will continue for as long as the delivery of the message is unsuccessful and the JMS bytes message itself has not expired. To preserve in-order delivery, all additional messages destined for the target Web service will be queued in the JMS message broker until the Web service is back on line.

### **JNDI Compatible Directory Server Down**

Depending on the sequence of events, exceptions due to the JNDI compatible directory server being down may manifest themselves in several ways. Both the inbound and outbound bridges may be unable to start because they are unable to fetch their configurations, and clients will likely receive JNDI naming exceptions, as their lookups for connection factories will fail.

## **Administration**

### **Overview**

This section is a brief introduction to the administration of Cortex, which is accomplished by administering these logical components:

- Inbound Bridge
- Outbound Bridge
- JNDI
- JMS

Administrators and developers can use commands to bind and unbind objects to the JNDI repository.

Note that all configurations for the inbound and outbound bridges are stored in the Cortex JNDI repository in LDAP. These configurations may be updated by modifying the `inboundconfig.xml` or `outboundconfig.xml` files and rebinding them to the LDAP repository.

## Inbound Bridge

Configuration of the inbound bridge is accomplished through the use of an XML file. The inbound bridge also provides a simple administrative GUI that can start and stop the inbound bridge, as well as display the current inbound configuration (Figure 14). The Inbound Bridge Administration screen (Figure 12) is available in the administrative servlet and located at `/inbound/bridge` at the deployed URL.



Figure 12. Inbound Bridge Administration Screen

## Outbound Bridge

Like the inbound bridge, the configuration of the outbound bridge is accomplished through the use of an XML file. As well, the outbound bridge provides a simple administrative GUI (Figure 13) that can be used to start and stop the outbound bridge, as well as display the current outbound configuration. The Outbound Bridge Administration screen is available in the administrative servlet located at `/outbound/bridge` at the deployed URL.



Figure 13. Outbound Bridge Administration Screen

## JNDI

JNDI is used to store all of Cortex's configuration and deployment properties. In addition, Cortex makes the JAXM provider connection factories available via JNDI.

Standalone inbound and outbound administrative applications have been provided to assist in the binding, rebinding, and unbinding of the required properties with a JNDI compatible directory server.

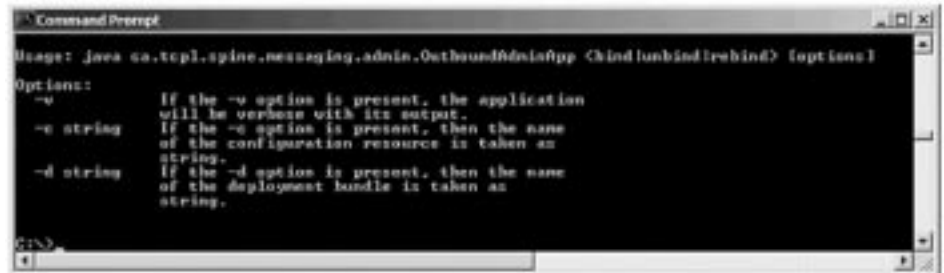


```

C:\> Command Prompt
Usage: java ca.tcpl.spine.messaging.admin.InboundAdminApp <bind|unbind|rebind> [options]
Options:
  -v                If the -v option is present, the application
                    will be verbose with its output.
  -c string         If the -c option is present, then the name
                    of the configuration resource is taken as
                    string.
  -d string         If the -d option is present, then the name
                    of the deployment bundle is taken as
                    string.
C:\>

```

Figure 14. Inbound Bridge Administration Configuration



```

C:\> Command Prompt
Usage: java ca.tcpl.spine.messaging.admin.OutboundAdminApp <bind|unbind|rebind> [options]
Options:
  -v                If the -v option is present, the application
                    will be verbose with its output.
  -c string         If the -c option is present, then the name
                    of the configuration resource is taken as
                    string.
  -d string         If the -d option is present, then the name
                    of the deployment bundle is taken as
                    string.
C:\>

```

Figure 15. Outbound Bridge Administration Configuration

## JMS

Lastly, Cortex requires that the appropriate JMS destinations (topics and queues) be configured as referenced in the bridges' configuration.

Since the JMS specification does not address this, vendor-specific administration of the JMS message broker is necessary.

## Tuning Tools

An indication of the messaging service's performance is available by using the ant target `test.load`. This test runs a set of benchmarks, sending 2,000 messages to the service and timing the outcome. The number of messages sent per second is printed out to the screen. This test enables the effect of any tuning measures to be assessed.

## Chapter 5

# Conclusion

Cortex is a key building block in the SPINE Web services initiative. Reliable messages based on open standards are essential to the creation of widely available Web services. The ultimate goal is to offer Services on Demand—the ability to deliver Web services to virtually any device.

Sun's architecture for delivering Services on Demand is the Sun ONE platform. With market-leading tools for creating, assembling, deploying, integrating, and testing, Sun delivers a comprehensive environment. Its appeal to TransCanada is strong, because Web services are built with open technologies on an integratable framework that can embrace and extend existing IT assets. Sun's commitment to open APIs is demonstrated with this project: integrators, third-party products, and industry-standard technologies all contributed to its success.

Today, TransCanada is using the Sun ONE architecture to move ahead with their vision of “better, faster, cheaper” to achieve its goals of reduced costs and a decreased time-to-market response, improving interoperability across technologies in an IT environment that can be almost seamlessly improved, maintained, and updated faster and more easily than before.

Sun has the vision, architecture, products, and expertise to deliver Services on Demand. With market-leading tools for creating, assembling, deploying, integrating, and testing, Sun offers a comprehensive environment for the Sun ONE architecture. Sun's commitment to open APIs is demonstrated with this project: integrators, third-party products, and industry standard technologies all contributed to its success. TransCanada is moving to achieve its goals of reduced costs and a decreased time to market response, by improving interoperability across different technologies in an IT environment that can be almost seamlessly improved, maintained, and updated faster and more easily than before.

## For More Information

### **Sun Open Net Environment (Sun ONE)**

[www.sun.com/sunone](http://www.sun.com/sunone)

### **Sun ONE Architecture Guide**

[www.sun.com/software/sunone/docs/arch/index.html](http://www.sun.com/software/sunone/docs/arch/index.html)

### **Sun ONE Messaging Queue**

[www.sun.com/software/products/message\\_queue/home\\_message\\_queue.html](http://www.sun.com/software/products/message_queue/home_message_queue.html)

### **Sun ONE Integration Server**

[www.sun.com/software/products/integration\\_srvr\\_eai/home\\_int\\_eai.html](http://www.sun.com/software/products/integration_srvr_eai/home_int_eai.html)

### **Sun ONE Directory Server**

[www.sun.com/software/products/directory\\_srvr/home\\_directory.html](http://www.sun.com/software/products/directory_srvr/home_directory.html)

### **Sun ONE Web Server**

[www.sun.com/software/products/web\\_srvr/home\\_web\\_srvr.html](http://www.sun.com/software/products/web_srvr/home_web_srvr.html)

### **Java API for XML Messaging (JAXM)**

[java.sun.com/xml/downloads/jaxm.html](http://java.sun.com/xml/downloads/jaxm.html)

### **JAXM Tutorial**

[java.sun.com/webservices/docs/1.0/tutorial/doc/JAXM.html](http://java.sun.com/webservices/docs/1.0/tutorial/doc/JAXM.html)

## Appendix A

# Company Information

## About Sun

Sun provides the end-to-end architecture that enables customers to support existing application needs while building a solid foundation for Web services of the future. The Sun Open Net Environment (Sun ONE) platform is an open, integratable stack designed to create and deploy Services on Demand and emerging Web services.

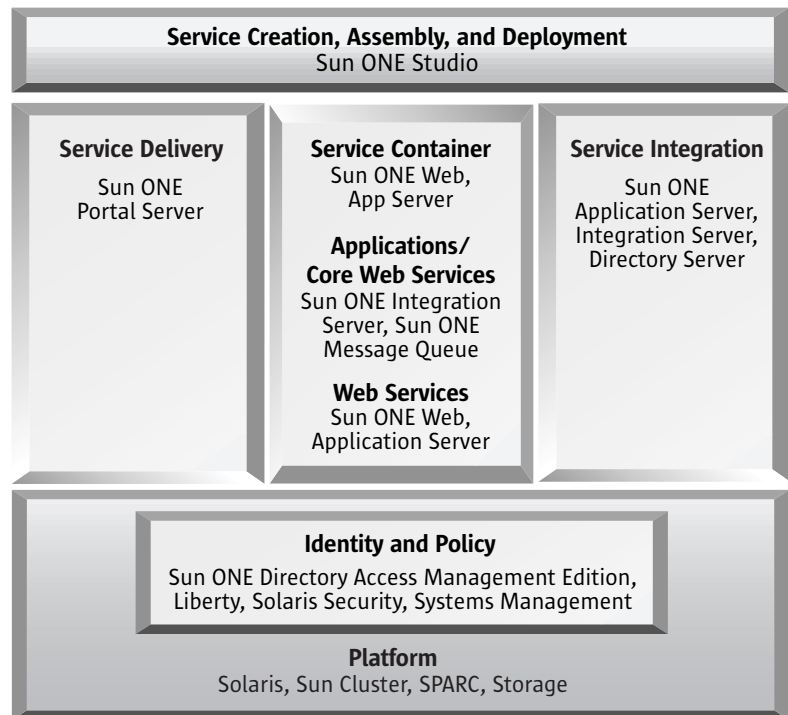


Figure 16. The Sun ONE Platform

Sun offers many resources that can help organizations to embrace Services on Demand.

These include:

- **Sun Professional Services:** Thousands of service professionals have been assisting, architecting, developing, deploying, and managing Services on Demand for businesses in more than 40 countries. Using field-proven methodologies and technologies, Sun consultants can design and implement the policies and best practices to meet unique business goals.
- **SunTone<sup>SM</sup> Program:** To provide customers with a means for identifying high-quality sources for Web-based services, Sun established the SunTone Certification and Branding program. These documented standards for excellence provide guidelines for architecting and operating required levels of service. SunTone certification can be applied to all aspects of a Web service, including infrastructure, service provider, applications, security, and management practices. The SunTone Architecture Methodology is based on many years of extensive experience designing and delivering Internet-based business solutions for thousands of customers across all industries. The SunTone program promotes progress toward reaching the WebTone—available, reliable Services on Demand.
- **iForce Initiative:** Sun's iForce initiative brings together Sun and its global industry partners to deliver proven solutions designed to help customers—ranging from startups to large enterprises—harness the power of the Internet and drive business advantage. iForce solutions differ from competitive, single-company offerings because the iForce community provides a rich array of products, programs, and services.
- **iForce Ready Centers:** Located around the world, iForce Ready Centers assist Sun customers with everything from brainstorming technological options for creating and IT infrastructure to proof-of-concept demonstrations to actual pilot programs. An iForce solution is a pretested aggregation of best-of-breed applications that is scalable, customizable, and adheres to open standards.

Sun has been a leader in Internet products and technologies for years. The Sun ONE platform leverages and extends existing Web service capabilities to Services on Demand.

These software products were instrumental in creating the Cortex proof of concept:

### **Sun ONE Integration Server, EAI Edition**

The Sun ONE Integration Server, EAI Edition, is designed for enterprises that need to integrate packaged, custom, legacy, and new Java technology applications. The product makes it possible for companies to integrate core business processes with multiple applications running on multiple operating systems across multiple communication protocols within the enterprise. By automating business processes across distributed heterogeneous information systems, companies can increase productivity and efficiency.

The Sun ONE Integration Server, EAI Edition software provides the following functionality:

- Hub-and-spoke architecture enables a structured EAI solution
- Workflow management for enterprise service usage
- Logging for process history

### **Sun ONE Message Queue**

Sun ONE Message Queue product is an implementation of the Java Message Service specification, a standardized API for handling asynchronous messaging between Java applications. It combines open standards, high reliability, and high performance, improving developer productivity for business integration needs. The Sun ONE Message Queue software is designed to enable disparate applications across the enterprise to communicate and work together efficiently. A message-oriented-middleware (MOM) product, Sun ONE Message Queue software enables the integration of legacy, enterprise resource planning (ERP), and new applications that are internal or external to the organization. This frees developers from having to focus on networking details, allowing them to concentrate on the business logic of Java applications. As a result, organizations reduce the cost of development and speed time to market. The Sun ONE Message Queue software is available in two editions, Platform Edition and Enterprise Edition. It features JMS Provider functionality and provides interapplication messaging through a message broker.

### **Sun ONE Directory Server 5.1**

As an integral component of the Sun ONE platform, the Sun ONE Directory Server 5.1 provides the foundation for a new generation of e-business applications and services. Based on a highly advanced, carrier-grade architecture, the Sun ONE Directory Server product delivers a high-performance, highly scalable user management infrastructure that helps organizations manage identity, relationships, and risk. As business relationships become increasingly complex, an e-business infrastructure must include a central directory for the consolidation of employee, customer, supplier, and partner information. By centralizing users, groups, and access controls across multiple applications, the Sun ONE Directory Server dramatically simplifies administration and helps lower the total cost of ownership. The Sun ONE Directory Server provides an e-business foundation that can change the way organizations deliver services to employees, customers, suppliers, and partners across multiple boundaries.

In the Cortex project, The Sun ONE Directory Server software is used as an LDAP repository of user and configuration information.

### **Sun ONE Web Server 6**

The Sun ONE Web Server is a software product for developers engaged in building dynamic Web applications for e-commerce sites. Multiplatform support makes it possible for developers to work in the operating system environment of their choice. The product works with Java Servlet and JavaServer Pages technologies to generate personalized content while speeding development. Centralized server management, content management, and rapid application development features combine to deliver a powerful means for enterprises to move their businesses to the Internet.

### **Solaris Operating Environment**

The Solaris OE is the foundation of the Sun ONE architecture. It redefines the operating system to a services platform by combining traditional OS functionality, application services, and identity management. Its capabilities are essential to the security, manageability, and quality of Services on Demand that are created using the Sun ONE platform. The Solaris OE enables IT organizations to deliver on the promise of massive scale, continuous real-time computing, and secure systems—while increasing service levels, reducing risk, and decreasing costs. And because the Solaris OE is tightly integrated with the Java 2 platform, it will remain at the forefront of operating environment offerings.

### **About ThoughtWorks**

ThoughtWorks, Inc. is a leading provider of custom e-business application development and advanced system integration services to Global 1000 companies. ThoughtWorks acts as the software integrator on the SPINE project, delivering solutions that require sophisticated software capable of integrating advanced business functionality with a company's existing core I.T. assets.

ThoughtWorks is headquartered in Chicago with offices in New York, San Francisco, Nashville, Calgary, London, Melbourne, and Bangalore, India. Founded in 1993, the company is privately held.

## Appendix B

# Example Code: Cortex's JAXM Provider Connection API

```
import ca.tcpl.spine.messaging.SpineMessage;
import javax.naming.InitialContext;
import javax.xml.messaging.ProviderConnection;
import javax.xml.messaging.ProviderConnectionFactory;
import javax.xml.soap.MessageFactory;

public class JAXMProviderConnectionSample {
    public static final String PROVIDER_CONNECTION_NAME = "spine-provider";
    public static void main(String[] args) throws Exception {
        String providerConnectionName = PROVIDER_CONNECTION_NAME;

        // Fetch the JNDI initial context.
        InitialContext context = new InitialContext();

        // Get a Provider Connection Factory to use.
        ProviderConnectionFactory factory =
            (ProviderConnectionFactory) context.lookup(providerConnectionName);

        // Ask the factory to create a Provider connection.
        ProviderConnection provider = factory.createConnection();

        // Create a message, appropriate for use with this provider.
        MessageFactory msgFactory =
            provider.createMessageFactory("spine-message");
        SpineMessage message = (SpineMessage) msgFactory.createMessage();

        // Set the target within the SPINE message.
        message.setTarget(...);

        // Populate the message's body.
        ...

        // Send message.
        provider.send(message);

        // Close the connection.
        provider.close();
    }
}
```

**SUN™** © 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, California 95054 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, Sun Open Net Environment, Sun ONE, Java, iForce, Solaris, J2EE, J2ME, JavaBeans, EJB, JavaServer Pages, JSP, Java Naming and Directory Interface, and SunTone are trademarks or registered trademarks 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 United States 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.

© 2002 Sun Microsystems, Inc., 4150 Network Circle, Santa Clara, Californie 95054 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.

THE NETWORK IS THE COMPUTER, Sun, Sun Microsystems, le logo Sun, Sun Open Net Environment, Sun ONE, Java, iForce, Solaris, J2EE, J2ME, JavaBeans, EJB, JavaServer Pages, JSP, Java Naming and Directory Interface, et SunTone 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.

Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, CA 95054 USA Phone 650-960-1300 or 1-800-555-9SUN Web sun.com



Sun Worldwide Sales Offices: Africa (North, West and Central) +33-13-067-4680, Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323, Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India-Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand-Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China-Beijing +86-10-6803-5588; Chengdu +86-28-619-9333; Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Singapore +65-6438-1888, Slovak Republic +421-2-4342-94-85, South Africa +27 11 256-6300, Spain +34-91-596-9900, Sweden +46-8-631-10-00, Switzerland-German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44-1-276-20444, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800