Service Oriented Architecture and Web Services

Chris Haddad
Senior Consultant

chaddad@burtongroup.com
www.burtongroup.com

February 17, 2004
Introduction

Burton Group’s Mission

Burton Group is an enterprise IT research and advisory service company specializing in the in-depth analysis of emerging infrastructure technologies.

Our mission is to provide vendor-neutral strategic planning and technology evaluation services that empower IT professionals, enabling them to make strategic decisions regarding network technology in support of business objectives.
SOA and Web Services

Thesis

• Web services generate measurable ROI
  • Simpler, cheaper integration

• Web services can deliver even more valuable benefits
  • Requires adoption of service oriented architecture (SOA)

• SOA is a mind-set
  • A design philosophy, principles, and best practices

• Companies that adopt Web services and SOA early will gain a strategic advantage
  • IT systems that support adaptable, flexible business models
SOA and Web Services

Agenda

• Strategic context
• Defining Service Oriented Architecture (SOA)
• Introduce the Network Application Platform
• Detail Web Services infrastructure and framework
• Demonstrate Web Services clients and services
  • Apache Axis and Ant
  • IIS/WSE and VS.NET
Strategic Context: Goal

Business Goals

- Reduce costs and complexity
- Increase efficiency and consistency of business process execution
- Improve manageability
- Gain regulatory compliance
- Increase revenue
- Open new markets
Technology Goals

- Increase sharing, reuse, composition, orchestration, and interoperability of services and resources
- Create the potential for composition, re-composition in the face of dynamic business conditions
- Reduce work duplication among distributed development teams
- Streamline deployment and maintenance of service code
- Lessen platform and tool vendor lock-in
- Allow seamless evolution of applications
Strategic Context: Goal

Enabling business processes to span boundaries

Goals:

• Enable ad hoc integration of processes

• Support all constituencies
  • Employees, partners, suppliers, and customers

• Maintain quality of service
  • Security, reliability, availability, scalability, performance, etc.
Strategic Context: Challenges

Physical and logical boundaries

• Stovepipe application systems
  • Systems weren’t designed for integration

• Vendor, language, and OS lock-in strategies
  • Platforms don’t support federation of services
  • Security, transactions, reliability, data management, etc.

• We need a “platform-independent” application platform
  • A set of infrastructure services available to all applications regardless of vendor, language, or OS
What is an application platform?

- Used to build and deploy applications
- Defined by its APIs and protocols
- Examples:
  - .NET, COM
  - Java, J2EE, LAMP
  - CICS/COBOL
- Frameworks supply access to OS and infrastructure services
- Each platform supplies a unique set of infrastructure services
- Tightly integrated inside; but hard to integrate with other platforms
Strategic Context: Challenges

Proliferation created an integration nightmare

Applications

APIs & Protocols
Frameworks
OS & Infrastructure

Process
Network
Storage
Security

CICS/COBOL

Applications

APIs & Protocols
Frameworks
OS & Infrastructure

Process
Network
Storage
Security

Windows/.NET

Applications

APIs & Protocols
Frameworks
OS & Infrastructure

Process
Network
Storage
Security

Java/J2EE

Applications

APIs & Protocols
Frameworks
OS & Infrastructure

Process
Network
Storage
Security

Linux/Perl
Strategic Context: Challenges

How do you turn this physically disjointed thing...
Strategic Context: Challenges

... and platform specific access methods...
... in the face of complexity

- Mix ‘n match integration makes QoS more difficult
- One service may be used in multiple applications
- Operational semantics (security, reliability, transactions, synchronicity, etc.) may be different in each app
Strategic Context: Solution

... into this logically holistic Network Application Platform?
Strategic Context: Solution

Solution Components

• Service Oriented Architecture
  • Service design practices

• Network Application Platform
  • Web Services Framework
  • Infrastructure Services Model
  • Web Services Infrastructure

• Application Services
What is SOA?

- SOA is a design style, depending on consistent, wide-ranging implementation of appropriate:
  - Philosophy:
    - Maximize service reuse, sharing, and interoperability
  - Principles:
    - Service based interactions
    - Standard service contract: WSDL
    - Standard service-brokering environment: UDDI
    - Standard service-binding environment: SOAP
  - Best practices:
    - Loosely coupled services
    - Coarse-grained service interfaces
    - Standards-based service infrastructure
What is a service?

- Reusable functionality exposed and invoked through published programmatic interface
- Programmatic interface is service-descriptive “contract”
NAP Service Bus enables bindings between clients and shared, reusable services.
Service-brokering environment enables registration, finding, and binding within NAP Service Bus and ISM
SOA Principles

What are SOA design principles?

- Design functionality as accessible and reusable services.
- Expose service functionality through programmatic interfaces.
- Maintain an abstraction layer between service interfaces and implementations.
- Describe service interfaces using metadata in standard service contracts.
- Register and find service contracts using standard service registries.
- Generate communication code automatically from service contracts.
- Bind services using standard protocols.
SOA Best Practices

Best practice: Loosely coupled services

• Create new functionality with service sharing, reuse, and standards-based interoperability in mind.
• Abstract service external interface from internal implementation.
• Define service interfaces through standard service description languages, especially WSDL.
• Expose service functionality through standard document-oriented approaches, especially SOAP Document/Literal style interfaces.
• Wrap or encapsulate existing services, applications, operating platforms, and other resources with service-oriented interfaces.
• Use standard, agreed-upon data/document schemas.
• Create self-describing XML-based messages/documents.
Best practice: Coarse-grained service interfaces

• Define service contracts that abstract the native object model and interfaces from a broad swath of functionality.

• Define “chunky” interactions that package several function calls and responses into fewer, but larger, messages.

• Define service contracts that correspond to the granularity of activities and subprocesses within end-to-end business processes.

• Use model-driven development to decouple service specification from service contract definition
Interface Granularity

**Interactive client**
- Create Order
- Add Item
- Add Item
- Submit Order

**Web services client**
- Submit Order

**Business Interface**

**Interactive server**
SOA Best Practices

Best practice: Standards-based service infrastructures

- Register and categorize service contracts in UDDI registries.
- Enable endpoints to use late-binding approaches to service-contract retrieval and consumption.
- Rely on stateless, asynchronous, messaging-based interactions among service endpoints.
- Implement a data mapping/transformation middleware layer
- Implement Web services management (WSM) infrastructures to ensure performance, reliability, availability, operational management, lifecycle management, and security.
- Integrate WSM infrastructures with existing identity, access management, policy management, and security environments
How does SOA relate to traditional middleware frameworks?

- SOA design principles may be used to either loosely or tightly couple services to each other
  - ...whereas traditional distributed object-computing environments—CORBA, DCOM, DCE, etc.—tightly couple services to various platforms and/or each other

- SOA best practice involves developing abstract service contracts:
  - Abstract from service’s internal platform implementation
  - Expose selected metadata describing message/document schemas and flow patterns that characterize external interface
  - Schemas and flows don’t correspond directly to service platform’s internal configuration, logic, state, behavior, and interfaces

- Abstract service contracts enable loose coupling and coarse-grained service interfaces
  - ...which are key to SOA value proposition.
Network Application Platform

Infrastructure Services Model

Generic services
- Data
- Presentation
- Other
- Object
- Presence
- Versioning

Foundation services
- Context management
- Federation
- Event
- Description
- Discovery
- Policy
- Virtualization

Control services
- Communication
- Security
- Management
- Coordination
- Orchestration

NAP ISM exposes all infrastructure functionality through standard service contracts
A Web Service:

- An application that exposes its functionality through a “Web API”
- Built using the Web Services Framework

Web Services Framework:

- A new type of middleware technology
- For application-to-application communication
- Based on XML, SOAP, WSDL, UDDI
- Pervasive Web Infrastructure (TCP/IP, HTTP, DNS)
Web Services: WSF

Web Service Framework: Foundational Standards

Interface Discovery and Description (UDDI, WSDL)

Messaging (SOAP)

Data Definition (XML, XML Schema)

Transport Protocol (SMTP, HTTP, Other)

Start out with SOA basics . . .
Web Services: WSF

SOAP: XML messaging protocol

- Standard message format
  - SOAP envelope containing
    - Header (directives)
    - Body (payload)
  - Message format defined by XML Schema
  - Independent of transport protocol
    - HTTP, SMTP, FTP, Jabber, JMS, etc.

- Standard processing rules

- A standard way to exchange XML messages
Web Services: WSF

SOAP Message Processing

Transport Packet

SOAP Envelope
- SOAP Header directives

SOAP Body
- payload

- Strip off transport
- Process SOAP envelope
- Process SOAP headers
- Process SOAP body
- Map XML to language
  - Invoke application
WSDL: Describes a Web service

- What it does
- How to communicate with it
- Where to find it

- Defines the Web service contract
  - Client uses WSDL to construct an interface to the SOAP service
What:
  Operation: Submit Order
  Input message: Order
  Output message: Confirmation

How:
  Binding: SOAP over HTTP
  Use: Document/Literal

Where:
  Service: OrderService
  Location: http://my.org/orders
Java API for WSDL

JWSDL

• API to manipulate WSDL
  • Used by SOAP runtime and tools
  • Used with DII

• Most SOAP tools provide WSDL compilers and generators
  • wsdl2java
  • Java2wsdl

• Open source WSDL API
  • WSDL4J
Web Services: WSF

UDDI: A registry service for Web services

- Categorized listing
- Service types
- Service implementations

- Helps you find a Web service and its description (WSDL)
Web Services: WSF

UDDI Data Model

- businessEntity
- businessService
- bindingTemplate
- tModel

keyed Reference

Service

WSDL
Web Services: WSF

Client and service interactions

1. Web Service
2. WSDL
3. UDDI Registry
4. Service
5. SOAP

UDDI Registry points to description, WSDL describes service, client finds service, SOAP communicates with XML messages.
Web Service Specification Composition

- **Service Composition**
  - (WSRP, WS-Notification, WS-Eventing)

- **Quality Of Service**
  - (WS-ReliableMessaging, WS-Transaction, WS-Authorization)

- **Description**
  - (WSDL, WS-Policy, WS-PolicyAttachments)

- **Messaging**
  - (SOAP, WS-Addressing, WS-Reference, WS-Security)

- **Data Definition**
  - (XML, XML Schema)

Layer advanced semantics . . .
Web Services: WSF

Web Services Framework Operational Semantic Categories

- Provisioning
- Presentation
- Orchestration
- Management
- Security
- Reliable messaging
- Transactions
- Data management
- Description
- Discovery
- Communication

By mapping business requirements
### Java WSP support for JAX* and WS-*

<table>
<thead>
<tr>
<th></th>
<th>Axis</th>
<th>BEA</th>
<th>Cape</th>
<th>IBM</th>
<th>IONA</th>
<th>Oracle</th>
<th>Novell</th>
<th>Sun</th>
<th>Systinet</th>
<th>WM</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP 1.2</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>H2</td>
<td>Q1</td>
<td>Q1</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>preview</td>
</tr>
<tr>
<td>UDDI 3.0</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H2</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
<td>N</td>
<td>Q204</td>
<td>N</td>
</tr>
<tr>
<td>SAAJ</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.1</td>
<td>1.2</td>
<td>1.2</td>
<td>1.1</td>
<td>N</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.1</td>
<td>1.1</td>
<td>1.0</td>
<td>partial</td>
</tr>
<tr>
<td>JAXM 1.1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>partial</td>
</tr>
<tr>
<td>JAXR 1.0</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H2</td>
<td>N</td>
<td>Q1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>partial</td>
</tr>
<tr>
<td>JWS/ML 1.0</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>JWS/ML 1.0</td>
<td>N?</td>
<td>'04</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Q1</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>SwA (MIME)</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Q1</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>WS-Attachments (DIME)</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td>WS-Security</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Q1</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>partial</td>
</tr>
<tr>
<td>WS-Routing/Referral</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>partial</td>
</tr>
<tr>
<td>WS-Addressing</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H2</td>
<td>Q1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
</tr>
<tr>
<td>WS-RM</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>H2</td>
<td>Q2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
</tr>
<tr>
<td>WS-Reliability</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H1</td>
<td>N</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
</tr>
<tr>
<td>WS-Coord/Txn</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H2</td>
<td>Q2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>WS-CAF</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Q2</td>
<td>H1</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>WS-RP</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Q1</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>WS-Policy</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>H2</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td></td>
</tr>
</tbody>
</table>
How does SOA relate to the Web services framework (WSF)?

- WSF defines standards for a NAP that is platform- and language-independent:
  - XML: standard data and metadata syntax
  - SOAP: standard middleware protocol
  - WSDL: standard service-contract schema
  - UDDI: standard service registry

- WSF-based NAP supports ubiquitous, any-to-any interoperability among services, applications, and other distributed resources:
  - Java, .NET, LAMP, etc.

- WSF is preferred NAP environment, enabling (but not guaranteeing) loosely coupled interoperability
WSF-based NAPs realize SOA promise, but not magic bullet

- WSF stack still immature, incomplete, evolving:
  - Lack universally implemented XML/SOAP-based standards for identity, reliability, transactions, events, pub/sub, and other important services
- Legacy applications and platforms must be migrated or wrapped to support WSF
  - Integration brokers and middleware platforms often necessary to mediate between WSF environments and legacy environments
- Tight coupling “worst practices” may inadvertently be used in WSF-based development:
  - …such as generating WSDL service definitions automatically from platform-specific object models, thereby “burning” platform artifacts (e.g., methods and APIs) into WSDL portTypes
Web Services Infrastructure

Physical and logical components
Web Services Platform

What is a Web Services Platform?

- **Tools**
  - Generate WSDL and proxies
  - Administer services
  - Monitor traffic
- **Run-time**
  - SOAP Message processing
  - Service agent invocation
- **Management Extensions**
  - Monitoring
  - Service Provisioning
  - Security
WSDL-based code generators

- SOAP client proxy generated from WSDL
- WSDL generated from class or template generated from WSDL
- Client
  - Java, C++, VB, C#, etc.
- SOAP Proxy
- Service Template
- Web Service
  - Java, C++, VB, C#, etc.
Web Services Platform: Runtime

Client

WSL

Deployment Descriptor (WSDD)

SOAP and WSDL processing

The application that implements the service

Corporate application
Desktop application
Wireless device
Web portal
etc.

Web Services Runtime Server

Java Application Server

WS Container

Web Service

SOAP Message Processor

DB

Legacy

Web services runtime server
Code generated by WS tools
Application code
Web Services Platform

Logical component view
Axis Architecture

Axis Components – Detail View

Transport
- HTTP
- SMTP
- JMS
- BEEP

Message processing
- Handlers
  - Transform
  - Management
  - Security
- Serialization Framework
  - Java Object
  - Serializer
  - Deserializer

Service Dispatch
- RPC
- MSG
- EJB
- COM

Web Service

Configuration Repository
- Flows
  - Request
  - Response
  - Fault
- Service Configuration
- Type Mapping

Administration

Security

Tools
- AdminClient
- SOAPMonitor
- WSDL2Java
- Java2WSDL
Axis Architecture

Transport Responsibilities

- Interface to network protocol
- Accept SOAP requests
- Create Message Context
- Send SOAP Response
Axis Architecture

Message Processing Responsibilities

- Identification of SOAP node’s role and initialize blocks
- Map SOAP message to Java objects
- Invoke interceptors (chains and handlers)
- Process SOAP header directives
- Hand-off to service dispatcher
Message Flows

- Assignment of processing blocks for SOAP request separated from SOAP response
- Exception processing channel defined by Fault Flow
**Axis Architecture**

**Processing Scope**
- Localize execution of handlers and chains
- Header fault check (MU)

**Transport Chains**

**Global Configuration Chains**

**Local Service Chains**

- Identification of SOAP node

**Request flow**

**Response flow**

**Fault Flow**

- SOAP Request
- Transport Listener (http)
- Pivot Handler (provider)
- SOAP Response
Serialization Framework Capabilities

- Mapping Java types to/from XML determined by
  - literal schema or section 5 encoding style. Document and rpc conventions.
  - JAX-RPC compliant

- Typical mappings
  - primitive data types (int, char, float, etc.)
  - Generic arrays and Java beans
  - custom serialization

Best Practice Tips:
- Use doc/literal
- Avoid collections
- Use MSG-style to bypass body serialization
Axis Architecture

Service Dispatching Responsibilities
• Load web service agent
• Format endpoint call
• Execute business logic
• Serialize return value as SOAP message
Axis Architecture

Axis Registry Deployment Descriptor (WSDD) Key Elements

- **Service Descriptor**
  - SOAP URI
  - Dispatcher type
  - Service Java class
  - WSDL File
  - Type mappings and handlers

- **Handler Descriptor**
  - Logical name
  - Handler Java class
  - Configuration parameters

- **Type Mapping**
  - XML QName
  - Java serialization class
  - Factories

```xml
<deployment>
  <handler>
    <parameters/>
  </handler>
  <service>
    <typemapping/>
  </service>
</deployment>
```
Java Web Services Standards

Standard APIs and Component Model

- Java APIs for SOAP
- J2EE Container/Component Model
- Java API for WSDL
- Java API for UDDI
- J2EE 1.4
Java APIs for SOAP

**JAX-RPC: RMI-style API**

- Application interacts via native Java objects
- Standard Java to/from XML mappings
- Proxy automatically generates SOAP messages
- WSDL-aware

**SAAJ: Low-level API**

- Application works with SOAP envelope
- Attachments
Enterprise Web Services

WSEE

• New container/component model for JAX-RPC in J2EE
• Port component
  • Can run in Web or EJB container
  • Implemented as a Java Class in Web container
  • Implemented as Stateless Session Bean in EJB container
  • Standard deployment descriptor
Java API for XML Registries

JAXR

• Supports both ebXML reg/rep and UDDI
• Uses ebXML data model
• Higher-level business abstraction
J2EE 1.4

Integrated Web Services Support

- J2EE 1.4 specification requires:
  - JAX-RPC
  - SAAJ
  - WSEE
  - JAXR
Building a Web Service

Application Service Components
Building a Web Service

• Build a new service
  • Start with code
    • Follow procedure for an existing application
  • Start or follow-up with WSDL
    • WSDL compiler generates application skeleton
    • Fill in appropriate application code

• Best Practices
  • How to encapsulate an application
    • Expose it directly
    • Build a façade, adapter, or wrapper
  • Decision criteria:
    • Does it have an existing API?
    • What’s the interface granularity?
      • Methods
      • Data Objects

Interoperability Tips

Don’t expose Java-centric objects to SOAP clients (i.e. turn maps into arrays)

Polymorphism and inheritance can be dangerous
Building a Web Service

Design Checklist

• Infrastructure Handlers
  • Security
  • Session Management
  • Transformation
  • Reliability

• Mapping Types
  • XML QName to Java classes
  • Serializers

• Exceptions

• Service Agent Code

• WSDL Generation
Building a Web Service: Exceptions

Server exception maps to SOAP fault

- faultcode = identifies cause of the fault
- faultstring = exception message
- faultactor = who generated the fault
- detail = fully qualified exception name
- Client runtime might be able to rethrow the exception
  - if it can’t, it should throw a generic exception e.g.,
    java.lang.RuntimeException
Mapping Java language types to/from XML

- Determined by
  - literal schema
  - encoding style

Typical mappings

- primitive data types (int, char, float, etc.)
- JavaBeans patterns mapping
  - public fields or fields with getters/setters
- generic Java types (List, HashMap, etc.)
- custom serialization
Building a Web Service: Collections

Often require custom serializers

Java Collection | XML Container | .NET Collection

Serializer | Deserializer
Building Web Service Clients

Practical design

• Building a Client
  • Proxy code generation from WSDL or DII
  • Supporting complex types
Building a Web Service Client

Development Checklist

• Find WSDL file
• Generate client interface
• Write the client code
  • Use the interface to invoke operations on the remote service
Client API Choices

• **Compiled Stub**
  • Compile time binding
  • Service interface and proxy specifics defined at compile time

• **Dynamic Proxy**
  • Runtime binding
  • Service interface and protocol specifics defined at compile time

• **Dynamic Invocation Interface (DII)**
  • Runtime discovery
  • Service interface and protocol specifics determined at runtime
Building a Web Service Client

Compiling a Client Proxy

1. Order Service
   - Submit Order
   - Get Order Status
   - List Backorders
   - Get Customer History
   - Get Order History

2. Generate WSDL

3. WSDL
   - Operation: Submit Order
   - Input message: Order
   - Output message: Confirmation
   - Use: SOAP over HTTP
   - Location: http://my.org/orders

4. Generate Code

5. Client
   - Submit Order

6. SOAP message
   - <Order>

   SOAP message
   - <Confirmation>