How To Develop Applications For Business Integration
Pulling J2EE, SOA and Web Services Together

André Tost
IBM Software Group
atost@us.ibm.com
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services versus SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services versus SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
  - BPEL
Service Oriented Architecture

- Application functionality contained in building blocks
- Create complete applications by connecting blocks together
- Model business relationships by providing blocks to others and using others' blocks
- Building blocks are services
A Service Oriented Architecture enables flexible connectivity of applications or resources by

- Representing every application or resource as a service with a standardized interface
- Enabling them to exchange structured information (messages, documents, ‘business objects’)
- Mediating the message exchange through a service integration bus
- Providing on-ramps to the bus for legacy application environments

This allows quick combination of new and existing applications to address changing business needs and improve operational effectiveness by managing the topology of the application ‘network’

The SOA infrastructure is also used to facilitate the management of business performance and quality of service
The Beginning: Roles & Functions

Set of industry-standard approaches to enable simplified connection of applications

Service Broker
- A searchable repository of service descriptions
- Service Providers publish their services
- Service requesters find services
- Service broker match providers with requesters

Service Provider
- Provide applications as WebService
- Publish their services
- Find service request to develop

Service Requester
- A client who needs a service
- Find service
- Publish their needs
How It Works...

- SOAP Processor
- XML Parser
- HTTP API (Java, C#, ...)
- "Wrapper" / "DD"
- Server- "Object"
- Client
- Proxy
- API (Java, C#, ...)
- "Wrapper" / "DD"
Characteristics of Service Oriented Architecture

- Reduced interdependency between software assets.
- Federated Control: Federated and policy-based security, management, and deployment.
- Standards-Based: Leverages open standards to represent software assets as services (XML, SOAP, WSDL, UDDI, …).
- Loosely Coupled: Reduced interdependency between software assets.
- Shared services: Allows individual software assets to become building blocks that can be reused in developing other applications (application assembly).
**A Couple of Best Practices**

A service encapsulates a well-defined invoke-able unit of business function, and exists either to provide information or to facilitate a change of business data from one valid and consistent state to another.

Services neither remember the last thing they were asked to do nor care what the next is, and are not dependent on the context or state of other services.

Services are defined using explicit interfaces that are independent of service implementations, and that both service requestors and service providers agree to.

Services should be invoke-able through defined communication protocols that stress interoperability and location transparency.

Any dependencies between services should be defined in terms of common business process, function and data models.
## The Art of Service Interface Design

<table>
<thead>
<tr>
<th>Service API</th>
<th>Code API</th>
<th>Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Few, large-grained interactions</td>
<td>Many, small-grained interactions</td>
<td>Less frequent interaction, but more data in each interaction</td>
</tr>
<tr>
<td>Every interaction the same</td>
<td>Lots of scope for variation between interactions</td>
<td>Less complex to change if one aspect of the interaction must change. All requestors must agree with the provider about the way the provider service works</td>
</tr>
<tr>
<td>Each interaction fulfills one step in a business process. There is no shared process at a lower level, and no shared state between calls (only business data)</td>
<td>Many interactions are needed to complete one step in a business process. Much shared process and state in these interactions</td>
<td>Less complexity to understand or coordinate if the requestor or provider need to change. Process, state and data models need only agree at the level of shared business process</td>
</tr>
</tbody>
</table>

### Granularity Balancing Act

- **Service APIs**: Flexibility
- **Code APIs**: Complexity
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services *versus* SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
  - BPEL
“Loose Coupling”??

- “Services are loosely coupled and invoked through communication protocols that stress location transparency and interoperability.”

![Diagram showing the concept of loose coupling with capacitors and communication protocols](image)
Aspects of Service Interactions ...

Requester  Provider

Location  Language  Data Format  Delivery Assurance  Interaction State  Semantic Interface

Platform  Protocol  Time  Security  Service Version  Service Provider Identity

Decoupled  Coupled

Service

Provider

Identity
Coupling Styles

- **Coupled**
  - Directly manipulated by service requester and provider application code.
  - *e.g.* business data model

- **Declared**
  - Clients and providers declare matching behaviour in interfaces.
  - *e.g.* WS-Security
  - **Transformed**
    - Specified in the service interface, but not manipulated by application code. Services declare different characteristics but the service infrastructure can mediate between requester and provider.
    - *e.g.* data format
  - **Negotiated**
    - Requester and provider interfaces declare a spectrum of behaviours, infrastructure negotiates an agreed behaviour for each interaction.
    - *e.g.* proposed WS-Policy, service provider identified dynamically through UDDI

- **Decoupled**
  - Entirely independent between client, provider and infrastructure.
  - *e.g.* platform independence through use of XML and HTTP
## Applying Coupling Styles to Aspects of Service Interactions

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Coupling Style</th>
<th>Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Interface</td>
<td>Coupled</td>
<td>WSDL, Application code, common objects, data dictionary</td>
</tr>
<tr>
<td>Language</td>
<td>Decoupled</td>
<td>Open standards, Web Services, J2EE, HTTP, XML</td>
</tr>
<tr>
<td>Platform</td>
<td>Decoupled</td>
<td>As above</td>
</tr>
<tr>
<td>Data Format</td>
<td>Declared or Transformed</td>
<td>WSDL, XML, adaptors, stylesheets, AD tools, middleware</td>
</tr>
<tr>
<td>Protocol</td>
<td>Declared or Transformed</td>
<td>HTTP, SOAP, Middleware, adaptors</td>
</tr>
<tr>
<td>Location</td>
<td>Decoupled</td>
<td>Communication protocols, HTTP, SOAP, middleware</td>
</tr>
<tr>
<td>Service provider identity or implementation</td>
<td>Declared, transformed or negotiated</td>
<td>Service routing, middleware, &quot;broker&quot;s, UDDI etc.</td>
</tr>
<tr>
<td>etc. ... Time, Delivery Assurance, Error Handling, Security, Service Version, Interaction State</td>
<td>etc ...</td>
<td>etc ...</td>
</tr>
</tbody>
</table>
SOA in the Enterprise

As a counter-example, consider the use of simple SOAP/HTTP wrappers

- SOAP/HTTP and WSDL leverage URL addressing and the existing HTTP and DNS infrastructure for Service Oriented Architecture.
- But they do not address enterprise requirements for SOA:
  - Multiple qualities of service
  - Widespread integration and interoperability
  - Transformations and mediations
  - Substitution of service implementations

Critically, management of an SOA “infrastructure” provided in this way is distributed between service endpoints and the existing infrastructure.
“Enterprise Service Bus”

- Analysts describe a middleware capability ...
  
  - “...a new architecture that exploits Web Services, messaging middleware, intelligent routing and transformation...” Roy Schulte, Gartner, Predicts 2003: Enterprise Service Buses Emerge

- Vendors talk about their low cost Java and Web Services “bus” solutions and contrast them to “high cost, proprietary hub and spoke EAI middleware”.
  
  - “Message brokers tend to be rather heavy duty and often require significant additional skills ... In contrast, an ESB is a pre-packaged SOA implementation ...” Steve Craggs, Best of Breed ESBs ... distributed by Polar Lake, Sonic, Fiorano, etc.

- IT shops have built their own ... primarily using XML with EAI technology (e.g. message brokers or adaptors) or HTTP.

- My view:
  
  - ESB is a logical infrastructure component that can be described by patterns
  - The ESB is defined by capabilities required to support SOA in the Enterprise in combination with other integration styles (messages and events)
  - The ESB provides a means to concentrate control and distribute topology
## ESB Capabilities

<table>
<thead>
<tr>
<th>Communications, e.g.</th>
<th>Service Interaction, e.g.</th>
</tr>
</thead>
<tbody>
<tr>
<td>■ Routing, addressing, protocols, pub/sub, async.</td>
<td>■ Interface definition, service substitution, messaging model, SOAP, WSDL, directories</td>
</tr>
<tr>
<td><strong>Integration, e.g.</strong></td>
<td><strong>Quality of Service, e.g.</strong></td>
</tr>
<tr>
<td>■ Database, legacy, middleware connectivity, service aggregation, app server connectivity, protocol transformation.</td>
<td>■ Transactions, delivery assurance</td>
</tr>
<tr>
<td><strong>Security, e.g.</strong></td>
<td><strong>Service Level, e.g.</strong></td>
</tr>
<tr>
<td>■ Authentication, authorisation, non-repudiation, confidentiality, standards support (WS-Security, Kerberos etc.)</td>
<td>■ Performance, throughput, availability, scalability.</td>
</tr>
<tr>
<td><strong>Message Processing, e.g.</strong></td>
<td><strong>Management and Autonomic, e.g.</strong></td>
</tr>
<tr>
<td>■ Encoded logic, content-based logic, message and data transformations, intermediaries etc.</td>
<td>■ Service provisioning and registration, logging, metering, monitoring, systems management etc.</td>
</tr>
<tr>
<td><strong>Modelling, e.g.</strong></td>
<td><strong>Infrastructure Intelligence, e.g.</strong></td>
</tr>
<tr>
<td>■ Object modelling, common formats and libraries, public vs. private etc.</td>
<td>■ Business rules, policy driven behaviour, pattern recognition, etc.</td>
</tr>
</tbody>
</table>
**Minimum ESB capabilities for SOA**

- If we require ESB to support our definition of SOA and add:
  
  **The ESB provides the means to manage the service infrastructure and the capability to operate in today’s distributed, heterogeneous environment.**

- We can define a minimum capability ESB implementation:

<table>
<thead>
<tr>
<th>Communication</th>
<th>Integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>▪ Routing and addressing services providing location transparency.</td>
<td>▪ Support for multiple means of integration, e.g. Java 2 Connectors, Web Services, etc.</td>
</tr>
<tr>
<td>▪ An administration capability</td>
<td></td>
</tr>
<tr>
<td>▪ At least one messaging paradigm (e.g. request / response, pub/sub, etc.).</td>
<td></td>
</tr>
<tr>
<td>▪ At least one transport protocol that is or can be made widely available.</td>
<td></td>
</tr>
</tbody>
</table>

**Service Interaction**

- Service messaging and interfacing model.
ESB in Terms of Existing and Future Technology

- We can position current and future technologies:
  - "Whilst packaged ESB technologies will mature they are not the only way to implement an ESB."
  - "SOAP/HTTP provides opportunistic integration capabilities, but not an ESB."
  - "ESB capabilities are available in existing products such as WebSphere Application Server, Web Services Gateway or WebSphere Business Integration."

- We can address some specific customer pain points:
  - Many enterprises have implemented a bespoke “service bus”, often using proprietary XML data models and MOM like WebSphere MQ.
  - This definition of ESB does not mandate the use of Web Services.
  - We can suggest enterprises evolve towards a more standard based approach whilst retaining their investment in existing technology.
  - The flexible definition is consistent with an incremental approach.

- We can respond to a range of requirements in the marketplace:
  - We can include basic SOAP and JMS-based solutions, but also position more sophisticated offerings.
"Distributed Bus" versus "Centralized Broker"

**Distributed Infrastructure**

- Service Client
- Service Client
- Service Client

- **<Bus>ESB**
  - Service Provider
  - Service Provider
  - Service Provider

**Single Point of Control**

- Service Client
- Service Client
- Service Client

- **<Hub>ESB**
  - Service Provider
  - Service Provider
  - Service Provider

- **<Hub>Runtime Node**

**Configuration and Control Services**

- Configuration
- Control Client

- **<Bus>Enterprise Service Bus**
ESB Scenarios

- ESB Scenarios in Service Oriented Architecture:
  - Basic Integration of Two Systems
  - Enable Wider Connectivity to One or More Applications
  - Enable Wider Connectivity to Legacy Systems
  - Enable Wider Connectivity to an EAI Infrastructure
  - Implement Controlled Integration Between Organisations
  - Automate Processes by Choreographing Services
  - Implement a Robust SOA with Web Services Support

  ... the scenarios suggest particular ESB capabilities from the model.

- The scenarios also suggest particular architectural issues, e.g.:
  - Provision of interfaces of appropriate content and granularity
  - Importance of support for open standards
  - Control over external access
  - etc. ...

  ... the resolution of these issues in turn implies additional ESB capabilities.
Example Scenario: Enable Wider Connectivity to Legacy Systems

- **Description:**
  - Enormous investment in legacy technologies
  - Significant value in providing open standard, service-based access to those systems

- **Relevant issues:**
  1. Function and Data Interfaces
  2. Common Business Data Model
  3. Technologies for Interoperability
  4. Advanced Interaction Characteristics
  5. Legacy XML Support and Processing
  6. Availability of Services in EAI Infrastructure
  7. Service Provider Protection
  8. Consistent and Controlled Service Enablement
  9. External Access to Services
  10. Service Level Requirements
  11. Security Requirements

- **ESB capability requirements:**
  - Communications (all)
  - Service Interaction (Service interface definition, Service messaging models)
  - Integration (Database, Legacy and Application adapters)
  - Quality of Services (all)
  - Security (all)
  - Service Level (all)
  - Message Processing (Message / service aggregation and correlation, Message and data transformations)
  - Management and Autonomic (Service Provisioning and Registration, Logging, Metering and Monitoring)
Stepping from Scenarios toward Implementation

- We can use the Patterns for e-business Process Integration patterns to describe the ESB and its role in the ESB scenarios.
  - <ESB>
  - <Exposed ESB>
  - <Exposed Service Gateway>
  - By using the P4eb patterns, we can relate the ESB patterns to other patterns such as service directories, <Serial Process>, <Parallel Workflow>, etc.

- We can then match the requirements to technologies using the ESB capability model, e.g. for IBM products:
  - Web Services Gateway
  - WBI Message Broker
  - Bespoke implementations using HTTP and WebSphere MQ
  - Related technologies such as WBI Server Foundation, WBI Interchange Server, etc.
The ESB Modelled as a <Zone>

Each Port is identified with a specific protocol and set of addresses through which it provides access to the Process Services Zone.

Requester view of Address Service namespace controlled by the <ESB> over supported invocation protocols.

Service addresses of service implementations.

Service Requesters

<Zone>ESB

ESB capabilities and services are available in the <zone>.

Service Providers

Enterprise
The ESB Modelled as a <Zone> (Continued)

- Semantic Interface
- Language
- Platform
- Data Format
- Protocol
- Location
- Service Provider Identity or Implementation
- Time
- Delivery Assurance and Error Handling
- Security
- Service Version
- Interaction State

Service Requesters

Service Providers

Requester Perspective

Provider Perspective

<ESB>

T

The ESB Modelled as a <Zone> (Continued)
Linking Multiple Patterns

![Diagram showing service requesters, service providers, and exposed service gateways linked through an ESB and ESB Link.](image)
"The Web Services Gateway is a run-time component that provides configurable mapping based on WSDL documents. It maps any WSDL-defined service to another service on any available transport channel. It is usually deployed at the firewall and has access to internal services."

```
  UDDI
  WSDL A*
  WSDL B*
  Query
  SOAP/HTTP
  Service Provider
  WSDL C
  SOAP/HTTP

  WSG Filter
  WSG Filter
  WSG Filter
  WSG
  WAS
  Service Provider
  WSDL A
  Query
  Publish
  SOAP/HTTP
  EJB/RMI-IIOP
  SOAP/JMS
  WSIF provider
  Publish
  UDDI
  WSDL C*
  Client
  SOAP/HTTP
```
Enterprise Service Bus

Service Flow

Data

Existing Applications

New Service Logic

Portal Service

SOAP Service Request (e.g. J2EE, .NET)

B2B Interactions

Enterprise Service Bus

B2B Interactions
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services *versus* SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
  - BPEL
The Business Integration Programming Model Challenge

WBI Traditional Concepts

Adaptors, Business Objects, Mediations, Collaborations, Organization Model, Relationships, Maps, Business Processes, ...

Service Oriented Architecture, Components, Object Technology, Transformation, Wiring, Assembly, BPEL, Human Interaction, Common Event Infrastructure

J2EE and Web Services

EJBs, Web Services, WSDL, Containers, Qualities-of-Service, Portlets, SDO, JCA, JMS …

A Business View

Bring it all together in a consumable programming model

(Service components, SDO, Patterns, Tooling, Templates)

Technology View

Patterns, Best Practices, etc.
I can assemble, but how can I customize? What I want is not quite what I found. Why do things look similar but are different? I cannot look for things by name? I do not know who might be my partner.

Service Oriented Binding
- Broker
- Consumer
- Provider

Web Services, WSDL, Partners

Mediation
- Target Selection (Selector FW), WBI
- Mapping/Relationships, Transformation/routing at message level

Flow Composition
- BPEL, Compensation, microflow

Policy Enabled Business Rules

Business Rules
- Connector/Adapter

Container Based
- Declarative, Configurable

JCA 1.5, Inbound, Outbound

Component Model
- J2EE, Service, SDO

Programming Model Elements
Enterprise Service Bus (from the outside in)

- Enables a service-bus of loosely-coupled components
- Standardized services specification
- Multi-protocol support — separates developer and deployment concerns
Enterprise Services Bus (from the inside out)

- Web browser, Web svc client
- EJB client
- JMS msg producer
- EIS
- HTTP listener
- IIOP listener
- JMS listener
- EIS listener
- Container
- Application components
- JDBC provider
- JMS provider
- IIOP provider
- EIS adapter
- Database
- Message Publisher
- RMI/IIOP target e.g. EJB
- EIS
Virtualized Messaging – Core Model

- End Applications are represented as Message Producers and/or Consumers
- *Destinations* virtualize the resources attached to the bus
  - Have a distribution pattern (one or all), and an option to queue messages
  - Can be connected together by the administrator, or dynamically
- Mediations modify routing, format and/or content of a message
  - Logic hosted in the network rather than at an endpoint
Virtualized Messaging for Web Services

- Messaging can be deployed in a single process or distributed across a bus.
- The core model and the bus topology are hidden from applications.
- Applications unaffected by changes to the topology.

Diagram:

- Administration
- Producer
- Destination
- Consumer
- Mediation
- WSPolicy doc

Types of Messaging:
- JAX-RPC
- Proxy
- SOAP/HTTP
- SOAP/JMS
- RMI/IIOP
- JMS
- RMI/IIOP
- HTTP
- DB2 Stored Procedures
- Java Beans
- Legacy/Procedural
- Other Relational Persistence

Connector Support:
- J2CA In-bound
- Servlet
- AXIS Runtime
- XML/HTTP
- JMS Message Listener
- ORB

Consuming on J2CA Inbound Extender Support
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services *versus* SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
  - BPEL
WSEE – JSR 109 1.1

- Defines web services support within a J2EE environment
- Leverages JSR 101 work
- Defines client programming model

WSDL document

- `<portType>`
- `<SOAP binding>`
- `<service>`
- `<port>`

JAXRPC mapping

- `<portType>`
- `<SOAP binding>`
- `<service>`
- `<port>`

Generated Svc Intfc

- Use JNDI to access svc intf
  - Use Svc Intfc to access endpoint interface
    - `getPort(port,SEI)`
    - `getPort()`
    - `createCall()`

J2EE DDs

- `<service-ref>`
  - `<service-interface>`
  - `<wsdl-file>`?
  - `<jaxrpc-mapping-file>`?
  - `<service-endpoint-interface>`?

WSDL to Java emitter

- Use JNDI to access svc intf
  - Use Svc Intfc to access endpoint interface
    - `getPort(port,SEI)`
    - `getPort()`
    - `createCall()`
JSR 109 1.1 (Continued)

Defines deployment model

- JAXRPC mapping
- WSDL document
- Endpoint Interface
- Client Stub
- Find
- Publish
- Tooling
- Runtime
- Bind
- Deployment
- SOAP Engine
- Java Business Object (JavaBean or EJB)
JSE 1.4: Changes to DDs - ejb-jar.xml and webservicesclient.xml

```
<ejb-jar id="ejb-jar_ID">
  <display-name>Stock Quote Sample EJB</display-name>
  <enterprise-beans>
    <session id="Session_1">
      <ejb-name>com_ibm_websphere_samples_webservices_stock_StockQuote</ejb-name>
      <home>com.ibm.websphere.samples.webservices.stock.StockQuoteHome</home>
      <remote>com.ibm.websphere.samples.webservices.stock.StockQuote</remote>
      <ejb-class>com.ibm.websphere.samples.webservices.stock.StockQuoteBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
  </enterprise-beans>
  <assembly-descriptor id="AssemblyDescriptor_1">
  </assembly-descriptor>
</ejb-jar>

<webservicesclient>
  <component-scoped-ref>
    <component-name>com_ibm_websphere_samples_webservices_stock_StockQuote</component-name>
    <service-ref>
      <description>Stock Quote Service</description>
      <service-ref-name>service/StockQuote</service-ref-name>
      <service-interface>com.ibm.websphere.samples.webservices.stock.StockQuote</service-interface>
      <port-component-ref>
        <service-endpoint-interface>com.ibm.websphere.samples.webservices.stock.StockQuote</service-endpoint-interface>
      </port-component-ref>
    </service-ref>
  </component-scoped-ref>
</webservicesclient>
```
New DDs – ejb-jar.xml

```xml
<ejb-jar id="ejb-jar_ID"
  xmlns="http://java.sun.com/xml/ns/j2ee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  version="2.1">
  <display-name>Stock Quote Sample EJB</display-name>
  <enterprise-beans>
    <session id="Session_1">
      <ejb-name>com_ibm_websphere_samples_webservices_stock_StockQuote</ejb-name>
      <home>com.ibm.websphere.samples.webservices.stock.StockQuoteHome</home>
      <remote>com.ibm.websphere.samples.webservices.stock.StockQuote</remote>
      <service-endpoint>com.ibm.websphere.samples.webservices.stock.StockQuote</service-endpoint>
      <ejb-class>com.ibm.websphere.samples.webservices.stock.StockQuoteBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
      <service-ref>
        <description>Stock Quote Service</description>
        <service-ref-name>service/StockQuote</service-ref-name>
        <service-interface>com.ibm.websphere.samples.webservices.stock.StockQuote</service-interface>
        <port-component-ref>
          <service-endpoint-interface>com.ibm.websphere.samples.webservices.stock.StockQuote</service-endpoint-interface>
        </port-component-ref>
      </service-ref>
    </session>
  </enterprise-beans>
  </ejb-jar>
```

JSR 109 1.1 (Continued)
```xml
<?xml version="1.0" encoding="UTF-8"?>
<webservices
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    version="1.1">

<!-- DOCTYPE webservices PUBLIC "+//IBM Corporation, Inc./DTD J2EE Web services 1.0//EN" "http://www.ibm.com/webservices/xsd/j2ee_web_services.dtd"-->
<webservices>
    <webservice-description>
        <webservice-description-name>StockQuoteFetcher</webservice-description-name>
        <wsdl-file>META-INF/StockQuoteFetcher.wsdl</wsdl-file>
        <jaxrpc-mapping-file>META-INF/StockQuoteFetcher_mapping.xml</jaxrpc-mapping-file>

    <port-component>
        <port-component-name>xmltoday-delayed-quotes</port-component-name>
        <port-component-name>urn:xmltoday-delayed-quotes</port-component-name>

        <webservice-address>http://stock.webservices.samples.websphere.ibm.com</webservice-address>
        <localpart>StockQuote</localpart>

    </wsdl-port>

    <service-endpoint-interface>com.ibm.websphere.samples.webservices.stock StockQuote</service-endpoint-interface>
    <service-impl-bean>
        <ejb-link>com.ibm.websphere.samples.webservices_stock StockQuote</ejb-link>
    </service-impl-bean>

</service-endpoint-interface>
</webservice-description>
</webservices>
```
Current Data Access Challenges

- Many different models/APIs for data and metadata retrieval and representation
  - Relational Databases (JDBC), XML files, JMS, Web Services (JAX-RPC), Enterprise Information Systems (EIS)
- Lack of support for standard application patterns
  - Transfer Object
  - Optimistic concurrency
  - Pagination of large data-sets
Consequences of Current Data Access Challenges

- Programmers focus on learning technologies, rather than solving business problems
- Programmers do a lot of low-level coding
- Tooling does not offer an easy development experience for J2EE application developers
Solution: Service Data Objects (SDO)

Unified data representation & retrieval across heterogeneous data sources.

Client

SDO Core APIs

Custom Mediator

Access APIs
Data APIs
Metadata Access APIs
Metadata APIs

JDBC Mediator

Access APIs
Data APIs
Metadata Access APIs
Metadata APIs

EJB Mediator

Access APIs
Data APIs
Metadata Access APIs
Metadata APIs

File

Data Base

Data Base

File

Data Base

Data Base

Data Base

File
SDO: Design Points

- Unified data access and representation across heterogeneous data stores
- SDO supports
  - Dynamic and static (strongly typed) data APIs
  - Disconnected programming model
  - Introspection of data
  - Change history for data modifications
  - Relationship integrity
- Designed to integrate well in a tooling environment
  - Data objects defined/configured using wizards and views
  - Tooling can integrate utilities for generating static data access APIs
- Not intended to replace other data access technologies
- SDO proposal was published jointly by BEA and IBM as JSR 235
## SDO Comparison with Other Technologies

<table>
<thead>
<tr>
<th>Technology</th>
<th>Model</th>
<th>API</th>
<th>Data Source</th>
<th>Metadata API</th>
<th>Query Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDBC RowSet</td>
<td>Connected</td>
<td>Dynamic</td>
<td>Relational</td>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td>JDBC CachedRowSet</td>
<td>Disconnected</td>
<td>Dynamic</td>
<td>Relational</td>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td>Entity EJB</td>
<td>Connected</td>
<td>Static</td>
<td>Relational</td>
<td>Java introspection</td>
<td>EJBQL</td>
</tr>
<tr>
<td>JDO</td>
<td>Connected</td>
<td>Static</td>
<td>Relational, Object</td>
<td>Java introspection</td>
<td>JDOQL</td>
</tr>
<tr>
<td>JCA</td>
<td>Disconnected</td>
<td>Dynamic</td>
<td>Record-based</td>
<td>Undefined</td>
<td>Undefined</td>
</tr>
<tr>
<td>DOM and SAX</td>
<td>N/A</td>
<td>Dynamic</td>
<td>XML</td>
<td>XML infoset</td>
<td>XPath, XQuery</td>
</tr>
<tr>
<td>JAXB</td>
<td>N/A</td>
<td>Static</td>
<td>XML</td>
<td>Java introspection</td>
<td>N/A</td>
</tr>
<tr>
<td>JAX-RPC</td>
<td>N/A</td>
<td>Static</td>
<td>XML</td>
<td>Java introspection</td>
<td>N/A</td>
</tr>
<tr>
<td>SDO</td>
<td>Disconnected</td>
<td>Both</td>
<td>Any</td>
<td>SDO Metadata API, Java Introspection</td>
<td>Any</td>
</tr>
</tbody>
</table>
SDO Features

- Dynamic Data API
- Support for Static Data API
- Complex Data Objects
- Change History
- Navigation through graphs of data
- Rich Metadata
- Validation and Constraints
- Relationship integrity
- XML support and XML schema support
**SDO: Architecture**

- **Client**
- **DataGraph**
- **DataObject**
- **Metadata**
- **Pluggable Data Mediator**
- **Data Source specific**
- **SDO Core API**
- **CRUD**
- **Access APIs**
- **Data APIs**
- **Metadata Access APIs**
- **Metadata APIs**
- **Data Source**
SDO: DataObject

- Data is held in a disconnected, source-independent format defined by the DataObject interface
- Data access APIs are
  - Dynamic (generic)
  - Static and strongly typed (generated)
- Includes a reference to metadata
- Holds primitive data or multi-valued fields
- Supports relationship integrity
- Supports getting/setting values in DataObject graph using an XPATH expression
SDO: DataGraph

- Represents a basic unit of data transfer between client and data store
- Encapsulates set of DataObjects
  - Contains a root DataObject
  - References metadata from DataObjects
  - All other DataObjects are reachable by traversing the references from the root DataObject
- Contains change information
  - Indicates which DataObjects have been added, updated, or deleted
**SDO: Data Mediator Service**

- Created for specific data store
  - JDBC Mediator, EJB Mediator, *etc.*
- Responsible for populating DataGraph
- Queries and updates the data store
  - Implements an optimistic concurrency strategy for updates
- Stateless with respect to the DataGraph
SDO DataObject Dynamic API

- String getString(String path) throws IllegalArgumentException;
- String getString(int fieldIndex) throws IndexOutOfBoundsException;
- void setString(String path, String value) throws IllegalArgumentException;
- void setString(int fieldIndex, String value) throws IndexOutOfBoundsException;
- int getInt(String path);
- int getInt(int fieldIndex);
- void setInt(String path, int value) throws ...;
- void setInt(int fieldIndex, int value) throws ...;
- An so on for all the “standard” types
- Object getObject(String path) throws ...
- Object getObject(int fieldIndex) throws ...
- void setObject(String path, Object value) throws ...
- void setObject(int fieldIndex, Object value) throws ...;
- DataObject getDataObject(String path) throws ...
- DataObject getDataObject(int fieldIndex) throws ...
- void setDataObject(String path, DataObject row) throws ...
- void setDataObject(int fieldIndex, DataObject row) throws ...;
- List getList(String path) throws ...
- List getList(int fieldIndex) throws ...;
//Use Data Mediator to get data graph
//Then get root DataObject
DataObject root = dataGraph.getRootObject();
DataObject dept = root.getDataObject("department");

//Get the first course in the data graph
List courses = (DataObject) dept.getList("courses");
DataObject course = dept.get(0);

//Access a student in the first course
List students = course.getList("students");
DataObject student = (DataObject) students.get(1);

//Same as above code snippet, but using XPATH
String xpath = "courses.0/students.1";
DataObject studentWithXPath = dept.getDataObject(xpath);

//Another XPATH example
xpath = "courses[num=567]/students[id=555]";
DataObject studentWithXPath = dept.getDataObject(xpath);
//Make data modifications
dept.setString("name", "Computer Science");

//Creating a new DataObject
DataObject newStudent = course.createDataObject("students");
newStudent.set("lastName", "Smith");
newStudent.set("id", "556");

//Deleting a DataObject from the DataGraph
student.delete();
SDO meta-model
SDO meta-data

- Meta-data import from
  - XML Schema
  - UML
  - Relational schemas
  - COBOL
  - Java
  - ...

Service Component Model

Requirements

- We need an analogous abstraction for a service-oriented component model
  - An abstraction that covers stateless session EJBs, web services, POJOs, BPEL4WS processes, database access, JCA access, etc.
  - Separates “business logic” from “infrastructure logic”
    - application programmers can focus on business problem
    - application logic can be much more portable
    - Not everyone has to be an IT infrastructure expert
  - Covers both usage of services/components and development of services/components
  - A uniform model for application programmers and for tools
    - Enables advanced, domain-specific, task-oriented tools
  - Uniform programming model for “programming in the small” (application development) and “programming in the large” (application integration)
    - Integrates with and provides programming model for the Enterprise Service Bus
      - ESB enables mediated application integration without modifying existing applications
Service Component Model Requirements (Continued)

- Not a new set of middleware abstractions!!
  - Why would anyone think they could do a better job than J2EE?
  - Instead, implement on top of existing middleware including J2EE and EJBs

- Rather it’s a component model that allows you to separate your business logic from your infrastructure logic by building:
  - “business logic components” – service components whose implementation contains only business logic and calls to other service components
  - and
  - “infrastructure components” – service components that encapsulate the code that calls the J2EE APIs and expose business interfaces to “business logic components”

- Infrastructure components are part of the business application and have APIs that are specific to the business application that they are part of.

- It also raises the bar of container support for components by providing a declarative approach to asynchronous programming and other features
Declarative Service Model

Core elements: Interface, Implementation, Reference and Wire

Service Component
Service1

interface = "Service1"
Service1.java
Java Interface

implementation = "Service1Impl"
Service1Impl.java
Simple Java Class

Service1
wire:
source = "Service1"
target = "Service2"

Service Component
Service2

interface = "Service2"
Service2.java
Java Interface

implementation = "Service2Impl"
Service2Impl.java
Simple Java Class

wire:
source = "Service2"
target = "Service2"
Protocol Transparency – Binding/Wiring Alternatives

Service Implementation Types
What Is an ESB Mediation?

- The ability to manipulate a message as it traverses the bus
  - Transform the message
  - Reroute the message to a different destination (or sequence of destinations)
  - Copy and route the message to additional destinations
  - Allow interaction with non-messaging resource managers (*e.g.* Databases)
The Basics of the Model

- Mediations are “message handlers”
  - They operate on a message or service request/reply
  - They are weakly or “generically” typed
  - They are independent of the end-point applications
  - Applications do not need to be modified when a mediation is added

- We will want to compose or aggregate individual pieces of function
  - By providing a collection of configurable (template) mediation components
  - Administratively or programmatically
  - Simple chains are a primary model but more complex graphs are also required and supported
Applications use Producers and Consumers to communicate via Destinations.

Destinations are Jetstream managed points of communication rendezvous (e.g. JMS, topics, JMS Queues, Web Service ports and services).

- Predefined, customizable beans
- Composition/Assembly
- Association of "mediation" to destination represents bean assembly configuration
Physical Realisation of a Mediated Destination

*optional
Administration

- A HandlerList is attached to a Destination by specifying its name
  - Can be referred to from any number of Destinations
- A set of properties administered on the destination can be used defined
  - These are available for use by the MediationHandlers
Mediation Detail

- MediationHandler

- schoolLog: Log

- schoolMapper: XSLTTransform

- modifyBean: ModifyRoutingBean

- newSchoolOutput: SendBean

- ESB supplied
- User written mediations
Agenda

- Service Oriented Architectures
  - Vision and Reality
  - Web Services *versus* SOA
- Introduction to Enterprise Service Bus
  - loose coupling
  - patterns
- Programming Model Elements
  - ESB runtime architecture
- SOA Component Model
  - J2EE anyone?? —> JSR109
  - SDO
  - Service Components
  - ESB Mediation
- Business Process Abstraction
  - BPEL
Integrating the Enterprise with Services Oriented Architecture (SOA)
Business Integration Programming Model: “Component Kinds”

- Programming model defines contract between tools and runtimes
  - One integrated runtime that implements the programming model
- Reduced complexity and progressive disclosure
  - Runs in a J2EE environment, but not all business integration users need the full power of the J2EE programming model
- Support application assembly and template-based development
  - Enable solution assembly via pre-canned solution building blocks like process templates, business objects
- Straightforward representation of business-level concepts and patterns
  - Process choreography, business activity monitoring and support for integration patterns like mediation, adapter, composite, state, strategy, ...

![Diagram of Business Integration Model]

- Adapters
- AsBO
- GBO
- Mediations
Sample Business Process

- Processing of a claim in an insurance company

1. receive claim data
2. automatic check
3. manual check necessary
4. clerk does manual check
5. compensate claim
6. compensation of claim declined
Business Processes as Web Services

WSDL
Port Type & Operation

receive claim data

automatic check

clerk does manual check

compensate claim

compensation of claim declined

I am using this web service ...
Business Processes between Web Services

- receive claim data
- automatic check
- clerk does manual check
  - compensate claim
  - compensation of claim declined
- eMail Service

Claim Checking Service
- manual check
- automatic check
- compensate claim
- compensation of claim declined
- eMail Service

Claim Compensation Service
Elements of a BPEL Process

- **Activities**: subtasks of the process
- **Control Links**: define the process' control flow
- **PartnerLinks**: placeholders for process callers and service providers
- **Variables**: hold data used in the business process
- **FaultHandlers** (optional): enclose activities that are performed in cases of error
- **CorrelationSets** (optional): support process instance identification
Elements of a BPEL Process: Activities

A BPEL Business Process is composed of

- Basic activities
  - Which are the things that we need to do as part of a business process
  - Receive input, reply to business partners or other business processes, manage exceptions, make decisions

- Structuring activities
  - Help us organize and manage the complexity of the flows
  - Typical programming constructs
BPEL Basic Activities

**Receive**
Wait for a message to arrive. Optionally start a new process instance when the message arrives.

**Reply**
Send a message in reply to a message that was received through a Receive.

**Invoke**
Invoke a one-way or a request-response operation offered by a partner.

**Pick**
Wait for one of multiple messages to arrive or for a time-out alarm to go off.

**Empty**
A "no-op" instruction in the business process.

**Terminate**
Immediately terminate the process instance.
BPEL Basic Activities

- **Assign**
  Update the values of variables with new data.

- **Wait**
  Wait for a given time period or until a certain time has passed.

- **Throw**
  Generate a fault from inside the business process.

- **Staff**
  Invoke an interaction with a human user.

- **JavaSnippet**
  Invoke an inline snippet of Java code.

- **Subprocess**
  Invoke another process – life cycle events will be propagated.
BPEL Structuring Activities / Elements

- **Flow**
  - Holds multiple activities that are performed concurrently.

- **Sequence**
  - Holds multiple activities that are performed sequentially in lexical order.

- **While**
  - Holds a (basic or structuring) activity that is repeated until a success criterion has been met.

- **Link**
  - Synchronizes two activities that are enclosed in a Flow (i.e., enforces a certain execution order).

- **Switch**
  - Selects one branch of activity from a set of choices.

- **Scope**
  - Allows for the definition of a nested activity with its own fault handler.
Elements of a BPEL Process: Variables

- Hold data that constitutes the state of a process
  - May be received from or sent to partners
  - Can be specified as input or output variables for invoke, receive, and reply activities
  - May hold state data related to the process and never exchanged with partners

- Associated with WSDL message types
Elements of a BPEL Process: Partner Links

- Partner: BPEL term for any entity that a process is interacting with
  - Business Partner, *i.e.* a web service
  - Internal Service, *i.e.* an EJB
  - Process Starter, *e.g.* a web application
  - ...

- Partner Link: "Placeholder" for a partner
  - Part of the process definition
  - No need to specify concrete service endpoints within the process model
  - Allows for late binding of partners (at assembly time ↔ build time)

- Allows for long-running, stateful interactions with a partner
Create Artifacts – BPEL File

Receive claim data

Automatic check

Clerk does manual check

Compensate claim

Compensation of claim declined

BPEL

Receive claim data

Write claim data to invoke input

Automatic check

Write claim data to staff activity input

Clerk does manual check

Write claim data to invoke input

Compensate claim

Compensation of claim declined

Write decline reason to invoke input
Web Resources

- IBM developerWorks web services domain
- RedBook “Implementing an SOA Using an Enterprise Service Bus”
- IBM on-demand homepage
- IBM web services homepage
- OASIS
- World Wide Web Consortium (W3C)
  - http://www.w3.org/
- Java Community Process (JCP)
  - http://www.jcp.org
- Apache
  - http://xml.apache.org
- WS-I
  - http://ws-i.org