Service-Oriented Architecture

Paola Turci

AOT Lab - DII - Università di Parma
Enterprise Application Integration

- Business Process Modelling
  - UML Activity Diagram
  - BPMN
  - WSBPEL
  - Process Integration Languages

- Integration issues and “traditional” approaches

- Service-Oriented Architecture
  - Service-Oriented paradigm
  - Web Services
    - Core Web Services Standards
  - Semantic Web Services: SAWSDL

- Enterprise Applications
  - Overview
  - Architectural solutions
    - Patterns of Enterprise Application Architecture

Source:
Thomas Erl
Service-Oriented Architecture: Concepts, Technology, and Design
“… Despite it’s apparent “newness” SOA, on a fundamental level, is based on a very old and established school of thought. Service-orientation, as a means of separating things into independent and logical units, is a very common concept.

…

Once applied to technology architecture, though, service-orientation is concerned with a specific part of our service-oriented world: **business automation**.

…

The manner in which an organization automates its business is a critical factor in determining the level of efficiency at which it operates and, ultimately, the extent of success it attains in its ventures.

This is what makes SOA so valuable. **By shaping automation logic through service-orientation, existing investments can be leveraged, business intelligence can be accurately expressed, and inherent automation agility can be achieved.** … “
Service-Oriented Architecture

- An environment where:
  - Services are ubiquitous and organically integrated
    - A service is a software building block that is **well-defined, self-contained**
      - Ideally **does not depend** on the context or state of other services
  - Systems are assembled from a **loosely coupled** collection of services, which
    - Have a **published interface**
    - Can **communicate with each other**
  - SOA and service-orientation are **implementation-agnostic paradigms** that can be realized through any suitable technology platform
    - Services compliant with Web Services standards (WSDL, SOAP, UDDI) are the most popular type of services available today
    - Focus on realizing SOA through and applying service-orientation principles to Web services technology
- Many companies are jumping in to Web services before standards emerge

"Most cultural change programs fail. Most strategic change programs fail. Most large IT programs fail or underperform. Aggressively adopting Web services at the enterprise level is all three combined. So the most critical decision is to see how not doing this can be competitively dangerous."

Samir Desai
CIO, Motorola
* \2003, CIO Magazine
“SOA and Web services adoption continues, but it is taking a long time for the industry to work out all of the specifications and standards. Core standards like SOAP and WSDL are widely adopted, and others like WS-Security are ready for broad adoption. But to build Web services that operate with high quality of service, the industry needs many other specifications like those for management, transactions, and advanced security. These are under development but as yet are mature enough only for aggressive technology adopters ...”

December 14, 2006
“Your Strategy For Web Services Specifications”
by Randy Heffner
Service orientation is a paradigm that frames what you do. Service-oriented architecture (SOA) is a type of architecture that results from applying service orientation.

We have been applying service orientation to help organizations consistently deliver sustainable business value, with increased agility and cost effectiveness, in line with changing business needs.

Through our work we have come to prioritize:

- **Business value** over technical strategy
- **Strategic goals** over project-specific benefits
- **Intrinsic interoperability** over custom integration
- **Shared services** over specific-purpose implementations
- **Flexibility** over optimization
- **Evolutionary refinement** over pursuit of initial perfection

That is, while we value the items on the right, we value the items on the left more.

---

**Guiding Principles**

*We follow these principles:*

- Respect the social and power structure of the organization.
- Recognize that SOA ultimately demands change on many levels.
- The scope of SOA adoption can vary. Keep efforts manageable and within meaningful boundaries.
- Products and standards alone will neither give you SOA nor apply the service-oriented paradigm for you.
- SOA can be realized through a variety of technologies and standards.
- Establish a uniform set of enterprise standards and policies based on industry, de facto, and community standards.
- Pursue uniformity on the outside while allowing diversity on the inside.
- Identify services through collaboration with business and technology stakeholders.
- Maximize service usage by considering the current and future scope of utilization.
- Verify that services satisfy business requirements and goals.
- Evolve services and their organization in response to real use.
- Separate the different aspects of a system that change at different rates.

Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.

At every level of abstraction, organize each service around a cohesive and manageable unit of functionality.
Elements of Service-Oriented Architectures

- **Loose coupling**: focus should be on high-level contractual relationships
- **Implementation neutrality**: the interface is what should matter
- **Flexible configurability**: late binding of components
- **Long lifetime**: components should exist long enough to be discovered, to be relied upon, and to engender trust in their behavior
- **Granularity**: interactions and dependencies should occur at as high level as possible
- **Reusability**: logic is divided into services with the intention of promoting reuse.
- **Composability**: Collections of services can be coordinated and assembled to form composite services.
Gartner's Hype Cycle

Don’t join in just because it’s “in”

Don’t miss out just because it’s “out”

Visibility

Maturity

Technology Trigger
Peak of Inflated Expectations
Trough of Disillusionment
Slope of Enlightenment
Plateau of Productivity

source: Gartner
Cycle provides a snapshot of the position of technologies relative to a market, region or industry

Hype Cycle's underlying message:

Don't invest in a technology just because it is being hyped or ignore a technology just because it is not living up to early over expectations. Be selectively aggressive — identify which technologies could be beneficial to your business, and evaluate them earlier in the Hype Cycle. For technologies that will have a lower impact on your business, let others learn the difficult lessons, and adopt the technologies when they are more mature.
First Hype Cycle for Emerging Technologies, 1995

Technology Trigger

Peak of Inflated Expectations
- Intelligent Agents
- Information Superhighway
- Virtual Reality
- Video Conferencing
- Wireless Communications

Trough of Disillusionment
- Handwriting Recognition
- Object-oriented Programming

Slope of Enlightenment
- Speech Recognition
- Knowledge-based Systems
- Plateau of Productivity

Emergent Computation

Type A
Type B
Type C

Recommended adoption time frame*

source: Gartner
Hype Cycle for Emerging Technologies, 2005

Plateau will be reached in:
- less than 2 years
- 2 to 5 years
- 5 to 10 years
- more than 10 years
- obsolete
- before plateau

Acronym Key
- 4G: fourth generation
- ASP: application service provider
- BPM: business process management
- P2P: peer to peer
- RFID: radio frequency identification
- SOA: service-oriented architecture
- VoIP: voice over Internet Protocol
- WiMAX: Worldwide Interoperability for Microwave Access
- XBRL: Extensible Business Reporting Language

source: Gartner (2005)
Hype Cycle for Emerging Technologies, 2008

Visibility

- Green IT
- Social Computing Platforms
- Solid-State Drives
- Video Telepresence
- Solid-State Drives
- Public Virtual Worlds
- Location-Aware Applications
- Service-Oriented Business Applications
- Tablet PC
- Web 2.0
- Virtual Assistants
- Electronic Paper
- RFID (Case/Pallet)
- Social Network Analysis
- Idea Management
- Corporate Blogging

Years to mainstream adoption:
- ○ less than 2 years
- ○ 2 to 5 years
- ○ 5 to 10 years
- ▲ more than 10 years
- ◊ obsolete before plateau

Source: Gartner (July 2008)
Hype Cycle for Emerging Technologies, 2009

- Cloud Computing
- E-Book Readers
- Social Software Suites
- Microblogging
- Green IT
- Video Telepresence
- Mesh Networks: Sensor
- Online Video
- Corporate Blogging
- Wikis
- Electronic Paper
- Tablet PC
- Idea Management
- Web 2.0
- Social Network Analysis
- Over-the-Air Mobile Phone Payment Systems, Developed Markets
- RFID (Case/Pallet)
- Speech Recognition
- Location-Aware Applications

- Video Search
- Human Augmentation
- Context Delivery Architecture
- Quantum Computing
- 3-D Flat-Panel Displays
- 3-D Printing
- Augmented Reality
- Surface Computers
- Mobile Robots
- Behavioral Economics
- Wireless Power
- Internet TV

- Technology Trigger
- Peak of Inflated Expectations
- Trough of Disillusionment
- Slope of Enlightenment
- Plateau of Productivity

Years to mainstream adoption:
- ○ less than 2 years
- ● 2 to 5 years
- ● 5 to 10 years
- ▲ more than 10 years
- × obsolete before plateau

As of July 2009
Entering the Plateau: B2B Web Services, Service-Oriented Architecture
SOA - Definition

- **Gartner**
  - A style of multi-tier computing that helps organizations share logic and data among multiple applications and usage modes.

- **IBM**
  - An Application Architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes.

- **OASIS**
  - A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.
SOA Early Model - Participant Roles & Interactions

- SOA is based upon the interactions between three roles:
  - Provider - the owner of the service
  - Registry or Broker - manages repositories of information on providers and their software assets
  - Requestor - discovers and invokes software assets provided by one or more providers

- There are three fundamentals interactions:
  - Publishing - providers publish information (or metadata) about services to a registry
  - Finding (service location) - requestors query a public or private registry for service description
  - Binding - requestors use the service description to create a message to be sent to the service provider
Service-Orientation Principles

- Service-orientation has become a **distinct design approach** which introduces commonly accepted principles
  - How its three core components (services, descriptions, and messages) are designed
The logical integration layer created by exposing legacy APIs via Web services offers a **standard** means of sharing data and programming logic.

- This has become a very attractive part of an integration architecture and, when properly designed, establishes a foundation for a service-oriented enterprise.
Completed work in 2004.

… A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. …
Why XML-Based?

- There is a growing need for **standard lightweight infrastructure** for **data exchange** in e-business applications
  - Everybody seems to agree that **XML** and **messaging based business transaction** will address these needs
  - Why XML?
    - It is a **universally accepted** standard way of structuring data (syntax).
    - It is a **W3C Recommendation**
    - The marketplace supports it with a lot of **free/inexpensive tools**.
    - The alternative to using XML is to define your own proprietary data syntax, and then build your own proprietary tools to support the proprietary syntax

XML ➔ **the lingua franca of the Web**
Component & Web Services Compared

- **Component-Based Model**
  - Mainly designed for processes **within the enterprise**
  - Different protocols and technologies (e.g. EJBs, DCOM, CORBA)
    - Typically, programming language dependent
    - Usually bound to a particular transport

- **Web Service Model**
  - Mainly designed for processes **across enterprises**
  - Uses common protocol and technologies (e.g. XML, SOAP, WSDL, ...)
    - Programming language independent (???)
    - Easily bound to different transport
Web Services - A New Paradigm?

- Web Services are something completely new: **Not True!**
- What is unique about Web service?
  - **XML-Based** - XML as the data representation layer for all Web services protocols and technologies
  - **Loosely-coupled** - a consumer of a web service is not tied to that web service directly
    - The only contract that have to be agreed upon between communicating parties is the **syntax and semantics of XML messages**.
      - No need to agree on object model
      - No need to agree on programming language,
      - No need to agree on programming APIs.
    - Ability to be **synchronous** or **asynchronous**
  - **Coarse-grained** - a piece of business logic
  - **Will allow**:
    - On the fly composition of new functionality
    - Decomposition and distribution of large scale processing tasks across many devices
Service Composition

- The size and scope of the logic represented by the service can vary.
- Service logic can encompass the logic provided by other services. In this case, one or more services are composed into a collective (service composition).
  - Applications can be developed out of web services assembled from all over the Internet.
Three Laws of Computing

- **Moore's Law**
  - Computing power doubles every 18 months

- **Gilder's Law**
  - Network bandwidth capacity doubles every 12 months

- **Metcalfe's Law (Net Effect)**
  - Value of network increases exponentially as number of participants increases

- **Another impact of web services is that they will trigger the Network Effect for integration technology.**
  Metcalfe’s Law describes the effect that is often illustrated with an example of FAX machines. The first FAX machine had zero value because it could communicate with no one. When a second came on line, the value increased. And as the network reached a critical mass, it compelled more and more users to get FAX machines. This is also called the Network Effect.
The Web services standards process began to fall apart this year. No fewer than four organizations — W3C, Oasis, WS-I and Liberty Alliance — are vying to preside over the process, each with different goals, each with differing degrees of power and influence.

"The Business of Standards Is Business"

* Oct. 1, 2003 Issue of CIO Magazine
Standard Organizations

- **W3C**
  Founded in 1994 by the inventor of the Web.
  Traditionally focused on the Web infrastructure level, it has moved into Web services as an extension of its core standards. All submissions it ratifies into standards must be free of royalty fees.

- **Oasis (Organization for the Advancement of Structured Information Standards)**
  Founded in 1993, it worked on the standard generalized markup language (SGML) until XML came along in 1998. Then it shifted its focus to XML and later Web services, SOA, …
  It lets individual technical committees decide whether they want to consider specifications that have royalties attached to them.

- **WS-I (Web Services Interoperability Organization)**
  Founded in February 2002 by Microsoft, IBM and seven other vendors. Its goal is to foster standardized interoperability. It provides profile documents that establish a proven and tested collection of standards. Creates guidelines and tests for interoperability.
  **WS-I is now part of OASIS**
Four standards define the critical elements of Web services:

- **Extensible Markup Language (XML+XMLSchema)**
  - Describes format of the request and response; data types
- **Simple Object Access Protocol (SOAP)**
  - Describes handshaking with server
- **Web Service Definition Language (WSDL)**
  - Allows servers to describe services being offered
- **Universal Description, Discovery, and Integration (UDDI)**
  - Protocol for listing services in a directory

This first generation Web services architecture allows for the creation of independent WSs capable of encapsulating isolated units of business functionality
The baseline for interoperable Web services
- Different Web services pieces in an installation-ready package

WS-I has specified basic profiles providing implementation guidelines for how related Web services specifications should be used together for best interoperability. The first one was the Basic Profile version 1.0:
- HTTP 1.1
- XML 1.0
- XML Schema 1.0
- SOAP 1.1
- WSDL 1.1
- UDDI 2.0

Basic Profile 2.0, Final Material, 2010-11-09
Relationship Between First-Generation Specification

- **UDDI** enables discovery of

- **WSDL** describes

- **SOAP** binds to

- **Web Services** enables communication between
Web Services
Simplified Architecture

1. Publish
   (Service description using WSDL)

2. Discover
   (query using WSDL)

3. Bind or Invoke
   (request and response based on WSDL)

Registry

(Service Provider)

(Service Requestor)

(UDDI Repository of WSDL Interfaces)
Web Services RPC - Example

1. Communications protocol
2. Message format
3. Description language
4. Discovery mechanism

Source: http://www.msdnaa.net/browse/01_Architecture.ppt
Document-based, mostly asynchronous, conversational interactions

- Service interactions typically happen through asynchronous document exchanges

- Communications protocols
- Message format
- Description language
- Discovery mechanism

Source: [http://www.msdnaa.net/browse/01_Architecture.ppt](http://www.msdnaa.net/browse/01_Architecture.ppt)
Second-Generation Web Services

- Technology and Standards are still evolving
  - SOAP, WSDL, UDDI are not enough
- Business web services is the next step, but more works are needed in
  - Quality of Service, management
  - Security, transaction, state and user context
  - Workflow management,
  - Provisioning, Accounting
WS-* extensions

- Message-level security, cross-service transactions, reliable messaging, orchestration, choreography and several other extensions, represent the second generation Web services platform
  - Generally labeled as “WS-*”, consist of numerous specifications that build upon the fundamental first-generation messaging framework (e.g. WS-Security).
  - These extensions address specific areas of functionality with the overall goal of elevating the Web services technology platform to an enterprise level.
High-level relationships between first- and second-generation standards
SOAP

See:

http://www.w3.org/TR/2007/REC-soap12-part0-20070427/
Originally conceived to bridge the gap between disparate RPC-based communication platforms

- SOAP acronym: Simple Object Access Protocol

Evolved into the most widely supported protocol for XML web services

- Establishes a standard message format; an XML document capable of hosting **RPC** but also **document-centric data**
  - Now SOAP acronym is frequently referred to as the Service-Oriented Architecture (or Application) Protocol

**SOAP is not an answer for all problems**

- Inefficient due to character (not binary) data and large headers
- Will not replace other distributed computing technologies (e.g. RMI)
Current status

- SOAP 1.1 specifications
  - An industrial standard
- SOAP 1.2 specifications
  - A W3C Recommendation (2007)
- Developed by several vendors
Codify several things:

- **Message envelope**
  - Format for message framing and extensibility

- **Encoding rules**
  - Rules for encoding common data types and application–defined data types in XML form
  - Messages are constructed using the data types defined in W3C XML schema

- **RPC convention**
  - Defines constructs to support RPC interaction between senders and receivers.

- **Asynchronous messages**

- **Binding with underlying protocols**
  - Binding for sending messages over (e.g.) HTTP
SOAP Nodes

- Represent the processing logic responsible for transmitting, receiving and performing a variety of processing tasks on SOAP messages.
When processing a message, a SOAP node assumes one or more roles

- Roles determine how headers are processed
- An optional attribute `env:role` (SOAP 1.2) is used to identify headers blocks intended for specific types of receivers. The two most common values are:
  - Next
  - UltimateReceiver

- The roles are associated only to types of SOAP nodes that perform a receiving function:
  - Intermediaries, which will process the header blocks identified with the next role
  - Ultimate receivers, which will process both

- A node first processes mandatory headers (`mustUnderstand="1"`), then others
SOAP Message Structure

SOAP Message

- Primary MIME part
- Attachment
- Attachment
- Attachment

SOAP Envelope

- SOAP Header
  - SOAP Block
  - SOAP Block

- SOAP Body
  - SOAP Block
  - SOAP Block
SOAP Message - Envelope

- The root element, which represents the container of a SOAP message

```xml
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    ...
  </env:Header>
  <env:Body>
    ...
    <!-- payload -->
  </env:Body>
</env:Envelope>
```

- Namespaces serve two functions
  - They help distinguish between different versions of SOAP
    - Es: SOAP 1.1: http://schemas.xmlsoap.org/soap/envelope/
  - The associated schema defines the structure of the SOAP elements: Envelope, Header, Body, and Fault
    - This can then be checked by a parser/validator
<!-- Envelope complex type and global element decl. -->

<xs:element name="Envelope" type="tns:Envelope" />

<xs:complexType name="Envelope">
  <xs:sequence>
    <xs:element ref="tns:Header" minOccurs="0" />
    <xs:element ref="tns:Body" minOccurs="1" />
    <xs:any namespace="##other" minOccurs="0" maxOccurs="unbounded" .../>
  </xs:sequence>
  <xs:anyAttribute namespace="##other" .../>
</xs:complexType>
Header (optional)

Common uses of header blocks include:

- Implementation of (predefined or application-specific) SOAP extensions, such as those introduced by second-generation specifications
  - ebXML messaging service provides security and reliability by defining elements that can be embedded in the header structure
- Identification of target SOAP intermediaries
  - While SOAP message progresses along a message path, intermediaries can add, remove or process information in SOAP header blocks
- Providing supplementary meta information about the SOAP message
SOAP Message - Body

- **Body** (*mandatory*)
  - Acts as a container for the data being delivered by the message
    - Data within the body is often referred to as “*payload*” or “*payload data*”
      - Application data
      - RPC methods and parameters
  - Can also be used to host exception information

```
<env:Envelope
  xmlns:env=http://www.w3.org/2003/05/soap-envelope>
  <env:Body>
    <env:Fault>
      ...
    </env:Fault>
  </env:Body>
</env:Envelope>
```

- The `<env:Body>` and its content are implicitly targeted and are expected to be understood by the ultimate receiver
The following sample code is taken from http://www.w3.org/TR/soap12-part0/

```xml
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation
      xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>
        uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d
      </m:reference>
      <m:dateAndTime> 2001-11-29T13:20:00.000-05:00 </m:dateAndTime>
    </m:reservation>
    <n:passenger
      xmlns:n="http://mycompany.example.com/employees"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Åke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
</env:Envelope>
```

- `env:mustUnderstand="true"` - the node(s) must absolutely process this header block in a consistent manner with the specification, or throw a fault
<env:Body>
  <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
    <p:departure>
      <p:departing>New York</p:departing>
      <p:arriving>Los Angeles</p:arriving>
      <p:departureDate>2001-12-14</p:departureDate>
      <p:departureTime>late afternoon</p:departureTime>
      <p:seatPreference>aisle</p:seatPreference>
    </p:departure>
    <p:return>
      <p:departing>Los Angeles</p:departing>
      <p:arriving>New York</p:arriving>
      <p:departureDate>2001-12-20</p:departureDate>
      <p:departureTime>mid-morning</p:departureTime>
      <p:seatPreference/>
    </p:return>
  </p:itinerary>
  <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
    <q:preference>none</q:preference>
  </q:lodging>
</env:Body>
</env:Envelope>
Marshalling is the packing of procedure parameters into a message packet.

- The RPC stubs call type-specific procedures to marshal (or unmarshal) all of the parameters to the call
  - On the client side, the client stub marshals the parameters into the call packet
  - On the server side the server stub unmarshals the parameters in order to call the server’s procedure.
- Vice versa for the response
SOAP Messaging

- Sends XML data from one application to another - **loose coupling**
- More flexible than RPC
  - Separates data from code
  - Any data can be passed
    - But ...
      - Application must encode and decode data
- Allows disconnected operation
  - Queued vs. Direct

Sender ➔ Receiver

Sender ➔ ▶️ ▶️ ▶️ Receiver

*direct*

*queued*
SOAP messages may be exchanged using a variety of "underlying" protocols

The specification of how SOAP messages may be passed from one SOAP node to another using an underlying protocol is called a **SOAP binding**

- SOAP over Java Message Service 1.0 (W3C Recommendation February 2012)
- Any SOAP `env:Envelope` infoset representation is made concrete through a protocol binding, providing
  - A serialized representation of the infoset that can be conveyed to the next SOAP node
  - Mechanisms to support *features* that are needed by a SOAP application
    - An encrypted channel
    - A reliable delivery channel
    - ...
SOAP HTTP Binding

- HTTP has a well-known connection model and a message exchange pattern
  - SOAP messages are wrapped in either an HTTP request or response packet

- Two message exchange patterns
  - **HTTP POST** method for conveying SOAP messages in the bodies of HTTP request and response messages
    - HTTP-specific instantiation of a binding feature called the SOAP request-response message exchange pattern,
  - **HTTP GET** method in a HTTP request to return a SOAP message in the body of a HTTP response
    - Uses a feature called the SOAP response message exchange pattern
POST /Reservations HTTP/1.1
Host: travelcompany.example.org
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: nnnn

<?xml version='1.0' ?>
<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope" >
  <env:Header>
    <t:transaction
      xmlns:t="http://thirdparty.example.org/transaction"
      env:encodingStyle="http://example.com/encoding"
      env:mustUnderstand="true" >5</t:transaction>
  </env:Header>
  <env:Body>
    <m:chargeReservation
      xmlns:m="http://travelcompany.example.org/">
      ...
    </m:chargeReservation>
  </env:Body>
</env:Envelope>
The data transmitted between companies is not always textual in nature (e.g. graphics files and PDF documents)

- There is an extension to the basic SOAP message structure that can accommodate non-SOAP (and non-XML) attachments to SOAP messages.
  - Uses the same encoding mechanism used in Internet email systems
  - Combines the SOAP protocol with the MIME format to allow any arbitrary data to be included as part of a SOAP message
    - The MIME protocol allows multiple arbitrary blocks of data to be put together in a message
The Core Architecture is built on three specifications: WSDL, SOAP and WS-Addressing.

Provides a framework
- To process SOAP messages
- To deploy a Web service (with or without WSDL)
- To send and receive SOAP messages with different transports

Provides a Client API that can be used to invoke Web services.
- This API supports both the Synchronous and Asynchronous programming models.
Sender creates the SOAP message.

Axis "handlers" perform any necessary actions on that message such as encryption of WS-Security related messages.

Transport sender sends the message.

On the receiving end, the transport listener detects the message.

Transport listener passes the message on to any handlers on the receiving side.

Once the message has been processed in the "pre-dispatch" phase, it is handed off to the dispatchers, which pass it on to the appropriate application.

from: http://axis.apache.org/axis2/java/core/docs/userguide.html
XML Data Binding

- Applications need to convert the XML to or from their own internal data structures
  - *Data binding* is the term used for techniques that handle the conversion between XML and application data structures.
- Axis2 provides several options for mapping WSDL to objects. Two of these are:
  - ADB (Axis2 DataBinding) is designed specifically for Axis2; it is probably the simplest method, but it does have limitations. It is not meant to be a full schema binding application, and has difficulty with structures such as **XML Schema element extensions and restrictions**.
  - XMLBeans is a fully functional schema compiler, however, it is a bit more complicated to use than ADB. It generates a huge number of files, and the programming model, while being certainly usable, is not as straightforward as ADB.
WSDL

See: http://www.w3c.org/TR/wsdl20
Web Service Standards

WS-BPEL

Composition

WS-Transaction

Quality of Service

WS- Reliable Messaging

Interaction

WS-Security

WSDL, WS-Policy, UDDI

Description

Other protocols Other services

SOAP (Logical Messaging)

XML, Encoding

Source:
Why WSDL?

- In order to call a SOAP endpoint, you need to know
  - target URL
  - required input
  - ...

- … web services need to be described in a consistent manner in order to be discovered by and interfaced with other services or applications

**WSDL makes it possible to describe such details**

- Machines can or could :-) figure out from WSDL documents what services are available and how to invoke them without previous manual pre-arrangement or pre-configuration
The Web Service Description Language is a W3C specification

- XML language

- Web service is described as
  - A set/collection of communication endpoints (ports)
  - Endpoint is made of two parts
    - Abstract definitions of operations and messages
    - Concrete binding to networking protocol (and corresponding endpoint address) and message format
  - This separation enhances reusability
Elements of WSDL 2.0

- **Abstract definition**
  - **Message**
    - Used to communicate with the WS
    - Typed definitions of data being exchanged
  - **Interface**
    - A group of operations offered by one endpoint of the WS
      - An operation is an abstract description of an action and refers to input and/or output messages
    - There can be more than one interface for a single WS

- **Concrete**
  - **Binding**
    - Maps an interface to a concrete protocol and data format (e.g. SOAP1.1 over HTTP)
  - **Service**
    - Aggregate set of related endpoints
    - Maps each binding to an endpoint (network address: URL for HTTP)
A simple service providing *book prices*

- A single operation called *GetBookPrice*
- Deployed using SOAP 1.1 over HTTP
- Request takes an *author name* and a *title* of type *string*
- Response returns a *price* as a *float*
WSDL Example (I) - Definitions and Namespaces

<!-- Defines name of the web service and multiple namespaces -->
<definitions name="BookStore"
  <!-- Namespace of this document -->
  targetNamespace="http://example.com/BookStore.wsdl"
  xmlns:tns ="http://example.com/BookStore.wsdl"
  xmlns:xsd1="http://example.com/BookStore.xsd"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl"
  <!-- Default namespace (for elements without a namespace prefix) -->
  xmlns="http://schemas.xmlsoap.org/wsdl"/>
WSDL Elements - Types

- Data type definitions
  - e.g. string, integer or custom data type
  - Used to describe exchanged messages
    - Used between the client and server

- WSDL is not tied to any specific typing system, but it uses W3C XML Schema as canonical type system
<!-- Defines the types used in messages -->
<wsdl:types>

<!-- wsdl:types encapsulate schema definitions of communication types, here using xsd -->
<xsd:schema
targetNamespace="http://example.com/BookStore.xsd"
xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">

<xsd:element name="BookRequest">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="author" type="string"/>
      <xsd:element name="title" type="string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
<xsd:element name="BookPrice">
  <xsd:complexType>
    <xsd:all>
      <xsd:element name="price" type="float"/>
    </xsd:all>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
</wsdl:types>
A portType is a named set of operations offered by the Web service

- An operation itself is a name given to a correlated exchange of messages
  - Each operation consists of a pattern of messages
    - Message names must be namespace-qualified (QName)
    - The sequence of messages defines the behavior of the operation
      - Four basic patterns

<table>
<thead>
<tr>
<th>Operation Behavior</th>
<th>Sequence of Messages (from client to service)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request-response</td>
<td>Input then Output</td>
</tr>
<tr>
<td>Solicit-response</td>
<td>Output then Input</td>
</tr>
<tr>
<td>Notification</td>
<td>Output only</td>
</tr>
<tr>
<td>One-way</td>
<td>Input only</td>
</tr>
</tbody>
</table>
<wsdl:message name="GetBookPriceInput">
    <wsdl:part name="body" element="xsd1:BookRequest"/>
</wsdl:message>

<wsdl:message name="GetBookPriceOutput">
    <!-- Zero or more part elements -->
    <wsdl:part name="body" element="xsd1:BookPrice"/>
</wsdl:message>

<wsdl:portType name="BookStorePortType">
    <!-- Combines multiple messages to form operations -->
    <wsdl:operation name="GetBookPrice">
        <wsdl:input message="tns:GetBookPriceInput"/>
        <wsdl:output message="tns:GetBookPriceOutput"/>
    </wsdl:operation>
    <!-- More operations -->
</wsdl:portType>
A binding defines **message format** and **protocol details** for operations and messages defined by a particular portType.

**Built-in extensions to allow expression of SOAP-specific details**

- `<soap:binding>`
  - Indicates binding will be made available via SOAP
  - style attribute indicates message format
    - document: simple XML documents
    - rpc: additional wrapper element indicating the function name

- `<soap:operation>`
  - Indicates binding of a specific operation to a specific SOAP implementation (SOAPAction, header)

- `<soap:body>`
  - For each operation, specifies details of the input/output messages
    - Encoding, fault, …

- `<soap:address>`
  - Location, it provides info about where service is accessible
<wsdl:binding name="BookStoreSoapBinding"
    type="tns:BookStorePortType">

  <soap:binding style="document" (or RPC)
    transport="http://schemas.xmlsoap.org/soap/http"/>

  <wsdl:operation name="GetBookPrice">
    <soap:operation soapAction="/BookStore/GetBookPrice"/>
    <wsdl:input> <soap:body use="literal" />
  </wsdl:input>

    <wsdl:output> <soap:body use="literal" />
  </wsdl:output>

  </wsdl:operation>

</wsdl:binding>
<!-- specifies location of one or more ports -->
<wSDL:service name="BookStoreService">
  <wSDL:documentation>Simple service</wSDL:documentation>
  <wSDL:port name="BookStorePort"
    binding="tns:BookStorePortType">
    <soap:address
      location="http://example.com/BookStore"/>
  </wSDL:port>
</wSDL:service>
</wSDL:definitions>

- A service groups a set of related ports together
  - A port defines an individual endpoint by specifying a single address for a binding
SOAP Encoding of the *GetBookPrice* Operation (Input)

POST /BookStore HTTP/1.1
Host: http://example.com

... 
Content-Type: text/xml; charset=utf-8
SOAPAction: “/BookStore/GetBookPrice“ //SOAP 1.1
Authorization: …
Content-Length: …

...

<?xml version="1.0" encoding="utf-8"?><soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
><soap:Body><GetBookPrice
xmlns="http://example.com/BookStore.wsdl"><author>Dante
Alighieri</author><title>Divina Commedia</title>
</GetBookPrice></soap:Body></soap:Envelope>
WSDL Design Issues

- Decomposition of a service into endpoints
- Granularity of operations
- Structure of the service, messages, data elements …
WSDL – Concluding Remarks

- Provides:
  - IDL description
  - Access protocol and deployment details
  - Functional information needed to programmatically access a service

- Does not include
  - QoS
  - Semantic information
- W3C Recommendation, August, 2007
- A simple, incremental approach
  - Largely based on WSDL-S
  - Built on WSDL
- 3 extensibility elements
  - `modelReference`
  - `liftingSchemaMapping`
  - `loweringSchemaMapping`
- Values are lists of URIs
- Can be used in both WSDL and XML Schema documents
- No Preconditions and Effects
No SAWSDL annotations defined for these WSDL components.

Annotated using modelReference and schemaMapping.

Source:
Semantic Annotations for WSDL and XML Schema
WWW2007, W3C Track, Banff, May 2007
• **modelReference**: used to specify the association between a WSDL or XML Schema component and a concept in some semantic model (ontologies, standards like Rosetta Net, ebXML).
  - It can be used to annotate the following:
    - WSDL components
      - Interfaces
      - Operations
      - Faults
    - WSDL Type Definitions
      - Type definitions
      - Element declarations
  - **liftingSchemaMapping**: specifies a mapping between a WSDL Type Definition in XML and semantic data.
  - **loweringSchemaMapping**: specifies a mapping between semantic data and a WSDL Type Definition in XML
    - Recommended languages: XSLT, Xquery
The annotation of the *operation* element carries a reference to a concept in a semantic model that provides a high level description of the operation, specifies its behavioral aspects or includes other semantic definitions.

**Source:**

*Semantic Annotations for WSDL and XML Schema*  
WWW2007, W3C Track, Banff, May 2007
Annotating Types

Semantic Annotations for WSDL and XML Schema
WWW2007, W3C Track, Banff, May 2007

Source:

(modelReference to establish a semantic association)

WSDL complex type element

Data level heterogeneity

OWL ontology

Address

hasStreetAddress

StreetAddress

hasCity

xsd:string

hasZip

xsd:string

semantic match

<scomplexType name="POAddress"
  sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#Address"
  sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/Ont2POAddress.xslt">
  <all>
    <element name="streetAddress" type="string" />
    <element name="poBox" type="string" />
    <element name="city" type="string" />
    <element name="zipCode" type="string" />
    <element name="state" type="string" />
    <element name="country" type="string" />
    <element name="recipientInstName" type="string" />
  </all>
</complexType>
User specified mappings from Web service message element to semantic model concept (say OWL Ontology)

- **liftingSchemaMapping**: from WS message element to OWL concept
- **loweringSchemaMapping**: from OWL concept to WS message element

Source:
Semantic Annotations for WSDL and XML Schema
WWW2007, W3C Track, Banff, May 2007
UDDI
See: http://www.oasis-open.org/standards
An infrastructure that enables the publishing and discovery of web services

- A neutral third party that facilitates dynamic and loosely coupled business-to-business (B2B) interactions.
  - A registry is available to organizations as a shared resource, often in the form of a web-based service

- Currently there are a variety of specifications for XML registries. These include:
  - **ebXML** Registry (Managed by OASIS standards body, approved under ISO 15000)
  - **UDDI**, Universal Description, Discovery and Integration (Managed by OASIS standards body)
What Is UDDI?

- Standard-based specifications for service description and discovery
  - Information that is shared
  - The APIs that are used to access it
- UDDI registry itself implemented as a web service
- An industry initiative
  - Microsoft, IBM, HP, Oracle, SAP, Accenture, Ariba, Commerce One
- Essential for dynamic usage of Web services
  - A “phone directory” for Web services that lists available Web services from different companies, their descriptions and instructions for using them
    - It can be thought of as a DNS for business applications
How UDDI Works?

1. SW companies, standards bodies, and programmers populate the registry with descriptions of different types of services.

2. Businesses populate the registry with descriptions of the services they support.

3. UBR assigns a programmatically unique identifier to each service and business registration.

4. Marketplaces, search engines, and business apps query the registry to discover services at other companies.

5. Business uses this data to facilitate easier integration with each other over the Web.

<table>
<thead>
<tr>
<th>UDDI Version</th>
<th>Year Released</th>
<th>Key Objective</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>2000</td>
<td>Public registry foundation of Internet-based business services</td>
</tr>
<tr>
<td>2.0</td>
<td>2003</td>
<td>Align specification with emerging Web Services and provide support for flexible, external taxonomies</td>
</tr>
<tr>
<td>3.0</td>
<td>2004</td>
<td>Support secure interaction of private and public implementations as key element of service-oriented infrastructure</td>
</tr>
</tbody>
</table>
Businesses register public information about themselves

- **“White pages”**
  - Business name, text description, contact info, and known identifiers

- **“Yellow pages”**
  - Industrial categorizations
    - **Industry**: NAICS (Industry codes - US Govt.)
    - **Product/Services category**: UNSPSC
    - **Location/geographic region**: Geographical taxonomy (ISO 3166)
  - Implemented as name-value pairs to allow any valid taxonomy identifier to be attached to the business white page

- **“Green pages”**
  - Technical information about services

- **UDDI Business Registry**
  - [http://www.xmethods.com](http://www.xmethods.com)
  - ...
UDDI allows users to define multiple taxonomies

- Users are not tied to a single system
  - They can employ an unlimited number of appropriate classification systems simultaneously

**Taxonomy Systems (or category systems):**
- UNSPSC (*United Nations Standard Products and Services Classification*)
- NAICS (*North American Industry Classification System*)

**Identifier Systems:**
- D-U-N-S Number (*Data Universal Numbering System*), uniquely identifies companies globally) universal standard for identifying and keeping track of over 80 million businesses worldwide
The core information model used by a UDDI registry is defined in several XML schemas.

UDDI Business Registry records the following information

- **businessEntity**
  - Basic information on business including name, contact
  - **businessKey** – unique business identifier, assigned during registration: UUID
  - **identifierBag** other business identifiers (e.g. D&B D-U-N-S® Number)
  - **categoryBag** business categories (e.g. NAICS code, UNSPSC)

- **businessService**
  - Services provided by a business entity

- **bindingTemplate**
  - Defines how to connect/communicate with a business service
    - Describes an instance of a Web service offered at a particular network address, typically given in the form of a URL
    - Describes the type of Web service being offered using references to tModels, application-specific parameters, and settings.

- **tModel**
  - The extensibility mechanism
  - **overviewDoc** – optional element; redirects users to additional references
Example of a Registration

**businessEntity**
- TB993...
- Harbour Metals
- www.harbourmetals.co.au
- "Serving Inner Sydney Harbour for ..."
- contacts
- businessServices
- identifierBag
- categoryBag

**keyedReference**
- EE123...
- NAI CS 02417

**keyedReference**
- DFE-2B...
- DUNS 45231

**tModelKeys**

**Peter Smythe**
- 872-6891
- 4281 King’s Blvd, Sydney, NSW
- Peter@harbourmetals.co.au

**businessService**
- 23T701e54683nf...
- Online catalog
  - “Website where you can ..." BindingTemplates

**BindingTemplate**
- 5E2D412E5-44EE-...
- http://www.sydneynet/harbour...
- tModelInstanceDetails

**tModelInstanceInfo**
- 4453D6FC-223C-3ED0...
- http://www.rosetta.net/catalogPIP

**keyedReference**
- DFE-2B...
- DUNS 45231

**Source**: [http://www.uddi.org/pubs/UDDI_Overview_Presentation.ppt](http://www.uddi.org/pubs/UDDI_Overview_Presentation.ppt)
- Provides inquiry and publishing APIs, allowing applications to interface programmatically with a registry

- Finding Business and Service
  - Includes set of methods to discover records
  - Includes set of methods to retrieve detailed records

- Builds on SOAP
  - Identifies all records by UUIDs
    - Universally Unique Identifier - 16-byte; its canonical hexadecimal form may look like this:
      - \texttt{uuid}: 550e8400-e29b-41d4-a716-446655440000
Comment: This diagram illustrates several models of registry interaction enabled by Version 3 of the UDDI specification. Through mechanisms like publish/subscribe and replication among peer nodes of a registry, the information in UDDI servers can be fully public (like the UBR), semi-private (such as the affiliated registries shown here), or even fully private and isolated from the public network (as depicted in the “Private Domain” above).

SAWSDL Publication and Discovery Using UDDI

1. SAWSDL file creating using annotations (**modelReferences**) pointing to semantic model

2. Service published in UDDI along with annotations

3. Service request created using terms from semantic model

4. Discovery based on annotations

**Source:**

Example of hybrid service matching with OWLS-MX
RESTful Web Services
What is a RESTful Web Service?

- **Representation State Transfer**
  - Was first introduced in 2000 by Roy Fielding at the University of California, in his academic dissertation
  - Defines a set of **architectural principles** (not a standard) …. by which you can design Web services that focus on system's resources
  - Defines a pattern of usage with HTTP to **create, read, update, and delete (CRUD)** data
  - “Resources” are identified by uniform resource identifiers (URIs)

- **Very popular protocol model**
  - Has gained widespread acceptance across the Web as a simpler alternative to SOAP- and WSDL- based Web services
  - Adoption by mainstream Web 2.0 service providers — including Yahoo, Google, Facebook, Amazon, Twitter, YouTube, Flickr, …
A concrete implementation of a REST Web service follows four basic design principles:

- Use HTTP methods explicitly and in a way that is consistent with the protocol definition (RFC 2616)
  - CRUD operations:
    - To create a resource on the server, use POST.
    - To retrieve a resource, use GET (idempotence = free of side effects)
    - To change the state of a resource or to update it, use PUT.
    - To remove or delete a resource, use DELETE.

- Be stateless: stateless server-side components are less complicated to design and write

- Expose directory structure-like URIs
  - URIs determine how intuitive the REST Web service is

- Transfer XML, JavaScript Object Notation (JSON), …

- Two specifications for its description:
  - Web Application Description Language (WADL)
  - WSDL 2.0 HTTP binding extension
- POST /students HTTP/1.1 Host: myserver
  Content-Type: application/xml
  <?xml version="1.0"?>
  <student>
    <name>John</name>
  </student>

- GET /students/John HTTP/1.1
  Host: myserver
  Accept: application/xml

- PUT /students/John HTTP/1.1
  Host: myserver
  Content-Type: application/xml
  <?xml version="1.0"?>
  <student>
    <name>Alice</name>
  </student>
To give client applications the ability to request a specific content type, REST service should make use of the built-in *HTTP Accept header*, where the value of the header is a MIME type. This allows the service to be used by a variety of clients.

This mechanism is known as *content negotiation*, which lets clients choose which data format is right for them and minimizes data coupling between the service and the applications that use it.

<table>
<thead>
<tr>
<th>MIME type</th>
<th>Content-type</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON</td>
<td>application/json</td>
</tr>
<tr>
<td>XML</td>
<td>application/xml</td>
</tr>
<tr>
<td>XHTML</td>
<td>application/xhtml+xml</td>
</tr>
</tbody>
</table>
package com.sun.jersey.samples.helloworld.resources;
import javax.ws.rs.GET;
import javax.ws.rs.Produces;
import javax.ws.rs.Path;
// class will be addressable at the URI "base URL +/helloworld"
@Path("/helloworld")
public class HelloWorldResource {
    // The java method will process HTTP GET requests
    @GET
    /* The Java method will produce content identified by the MIME Media type
    "text/plain" */
    @Produces("text/plain")
    public String getMessage() {
        return "Hello World";
    }
}
HTTP Request and Response

GET / helloworld HTTP/1.1

HTTP/1.1 200 OK
server: ...
Content Type: text/plain

Hello World
SOA: Analysis, Design and Implementation

Source: Thomas Erl
Service-Oriented Architecture: Concepts, Technology, and Design
Enterprise Application Integration

- **Business Process Modelling**
  - UML Activity Diagram
  - BPMN
  - WSBPEL
  - Process Integration Languages

- **Integration issues and “traditional” approaches**

- **Service-Oriented Architecture**
  - Service-Oriented paradigm
  - Web Services
    - Core Web Services Standards
  - Semantic Web Services: SAWSDL

- **Enterprise Applications**
  - Overview
  - Architectural solutions
    - Patterns of Enterprise Application Architecture

Source: Thomas Erl
Service-Oriented Architecture: Concepts, Technology, and Design
Modeling services is fundamentally a process of gathering and organizing business model information.

- Business process logic can be decomposed into a series of granular steps that represent candidate service operations.
- Candidate service operations are grouped into logical contexts that represent candidate services.
In an enterprise model, the service interface layer is located between the business process and application layers.

Three layers of abstraction:

- **Application service layer**
  - Provides reusable functions/operations related to new or legacy applications.

- **Business service layer**
  - Services are a direct implementation of the business service model.

- **Orchestration layer**
  - Modeling tools exist, allowing technical analysts and architects to graphically create business process diagrams.
Business Process Modeling

an abstract business process definition expressed through the WS-BPEL language

Source: Thomas Erl
Service-Oriented Architecture: Concepts, Technology, and Design
Modeling Service Layer

Source: Thomas Erl

Service-Oriented Architecture: Concepts, Technology, and Design
Service-Oriented Design

- WSDL is the focal point of service design, as it is used to design the abstract and concrete definitions of service interfaces
  - Auto-generating service interfaces by deriving them from existing component classes is not a desirable service design approach for building SOA.
  - Hand coding WSDL service definitions and associated XSD schema content provides the highest degree of independence and can be supplemented with an editor that provides validation and testing features
Advantages:

- Services can be designed to accurately represent the context and function of their corresponding service candidates.
- Conventions can be applied to service operation names, which leads to standardized endpoint definitions.
- The granularity of operations can be modeled in abstract to provide consistent and predictable interface designs that also establish a message size and volume ratio suitable for the target communications infrastructure.
- Underlying applications are required to conform to the expression of the service design, not vice versa. (This often results in the need for a business façade layer to compose older components)
- The design of business services can be assisted by business analysts to ensure an accurate representation of business logic.
Enterprise Service Bus is an infrastructure that can be used as a backbone upon which to build service-oriented applications

- Facilitates the requirements of a highly-scalable, fault tolerant, message-driven, service-oriented enterprise.
- Provides API which can be used
  - To develop services and makes services interact with each other reliably.
Technically ESB is a messaging backbone which provides a standards-based bus and adapters for multiple protocols along with common services.