Service Oriented Architecture and Web Services

Ing. Nicola Zaghini

nicola.zaghini@unibo.it
Outline

• SOA
  • Fundamental SOA
  • SOA Service
  • Service orientation principles

• Web Services SOA oriented
  • Proposal & framework
  • Logical components of automation logic
  • Service-orientation and the enterprise
    • Layers of Abstraction
  • Service role
  • Service description (WSDL)
    • WSDL: which style?
    • Messaging framework (SOAP)

• Message exchange patterns
• SOA Platform
SOA introduction

- SOA is Service Oriented Architecture

- Web Services and SOA are related but are independent ...

- SOA is a new paradigm regard object orientation...

- why we need new paradigm?

- --> follow the example
SOA introduction

**Gestione ODT**
- **LSU:**
  - Gestione siti, localizzazione.
  - Inserimento/importazione ODT.

- MDT assegnate al tipo di mezzo
- Singolo ODT
- Richiesta preventivo

**Gestione flotta**
- **TSP:**
  - Inserimento/importazione MDT.
  - Mezzi, tipi di mezzo, accessori.
  - Storia dei mezzi.
  - Calendario degli impegni dei mezzi.
  - Listino.
  - Calcolo automatico dei costi delle MDT.
  - Pianificazione (manuale e automatica) dei mezzi a disposizione e assegnamento della MDT al mezzo.
  - Gestione delle distanze note.

**Broker dei trasporti**
- **LSM:**
  - Listini concordati con i TSP.

- Richiesta valorizzazione
- Invio listino

- Avanzamento MDT
- Preventivo

**Calcolo MDT**
- **LSM:**
  - Tipi di materiale (STD), compatibilità.
  - Tipi di mezzo (STD).
  - Gestione dinamica del cluster.
  - Raggruppamento (manuale e automatico) di ODT in pacchetti -> generazione MDT.

**Calcolo Giri**
- **AUTOMATICO:**
  - Calcolo del percorso minimo in termini di chilometri, tempo, altri parametri.
  - Gestione delle distanze note.

**WS Geografico**
- Distanze tra siti ↘
- Richiesta distanze tra siti ↗

- Richiesta distanze tra siti ↘
- Distanze tra siti ↗

- MDT ↦
- Richiesta distanze tra siti ↗

- Invio listino
- Richiesta valorizzazione
- Avanzamento MDT
- Preventivo

- Singolo ODT
- MDT assegnate al tipo di mezzo
Fundamental SOA

“Because the term ‘service-oriented’ has existed for some time, it has been used in different contexts and for different purposes. One constant through its existence has been that it represents a distinct approach for separating concerns. What this means is that logic required to solve a large problem can be better constructed, carried out, and managed if it is decomposed into a collection of smaller, related pieces. Each of these pieces addresses a concern or a specific part of the problem. [...] What distinguishes the service-oriented approach to separating concerns is the manner in which it achieves separation.”

From: Service-Oriented Architecture Concept, Technology, and Design - Thomas Erl
Fundamental SOA

- Service-oriented architecture

  - a model in which business logic is decomposed into smaller, distinct units of logic

  - collectively this units comprise a large piece of business logic (individually can be distributed)

- BUT We want to

  - self-governing individual services -> independence between services

  - MUST ensure that they adhere to certain baseline conventions
An early incarnation of SOA
SOA Service

Service as a unit of logic within a context

• service has a **description**

• services relate using **messages**

  • we need messaging framework

  • message as “independent units of communication”

**SOA KEYs: Service, Description and Message**
How service encapsulate logic

- To retain their independence, services encapsulate logic within a distinct context. This context can be specific to a business task, a business entity, or some other logical grouping.
How services relate

- The relationship between services is based on an understanding that for services to interact, they must be aware of each other. This awareness is achieved through the use of service descriptions.
How services communicate

- After a service sends a message on its way, it loses control of what happens to the message thereafter. That is why we require messages to exist as "independent units of communication." This means that messages, like services, should be autonomous.
How services are designed

- When a solution is comprised of units of service-oriented processing logic, it becomes what we refer to as a service-oriented solution.
Service orientation principles

- **Common principle/1:**

  - **Services are reusable**
    Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.

  - **Services share a formal contract** *
    For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.

  - **Services are loosely coupled** *
    Services must be designed to interact without the need for tight, cross-service dependencies.

  - **Services abstract underlying logic** *
    The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.
Service orientation principles

• Common principle/2:

  • **Services are composable**
    Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers (ESB principle...)

  • **Services are autonomous** *
    The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.

  • **Services are stateless**
    Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.

  • **Services are discoverable**
    Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.
Web services (the proposal of)

- “Web Services provide a standard means of interoperating between different software application on a variety of platforms and frameworks”

- ... Web Services Architecture W3C working group

- they focus on Interoperability!

- “WS is a software system designed to support interoperable machine-to-machine interaction over a network [...] using SOAP messages”

- “WS is an abstract notion that must be implemented by a concrete agent [...] the agent is the concrete piece of software that send and receive messages”
Web services framework

- Web services framework is flexible and adaptable -> large in scope

- Characterized by/1:
  - an abstract (vendor-neutral) existence defined by standards organizations and implemented by (proprietary) technology platforms
  - core building block that include Web services, service descriptions and messages
  - a communications agreement centered around service descriptions based on WSDL
Web services framework

• Characterized by/2:

• a messaging framework comprised of SOAP technology and concepts

• a service description registration and discovery architecture sometimes realized through UDDI

• architecture that support message pattern

• WS-* specifications
Web services framework
Web services framework

Chapter 3: Introducing SOA

3.1 Fundamental SOA

3.1.2 How services encapsulate logic

3.1.3 How services relate

3.1.4 How services communicate

Chapter 5: Web Services and Primitive SOA

5.2 Services (as Web services)

5.3 Service descriptions (with WSDL)

5.4 Messaging (with SOAP)
Web services framework

- based on open standard
- support vendor diversity
Logical components of the WS framework
Logical components of automation logic

Fundamental parts of the WS framework

- SOAP messages
- Web service operations
- Web services
- activities (typically used to represent the temporary interaction of a group of Web services)

Service oriented perspective

- messages
- operations
- services
- processes (static definition of interaction logic)
Logical components of automation logic

Automation logic is comprised of four parts

- **messages** = units of communication
- **operations** = units of work
- **services** = units of processing logic (collections of units of work)
- **processes** = units of automation logic (coordinated aggregation of units of work)

A primitive view of how SOA modularizes automation logic into units.
Logical components of automation logic

A primitive view of how units of communication enable interaction between units of logic.

The scope of an operation within a process
Logical components of automation logic

Operations belonging to different services representing various parts of process logic
Service-orientation and the enterprise

- Business logic is a documented implementation of the business requirements that originate from an enterprise’s business areas.

- Application logic is an automated implementation of business logic organized into various technology solutions.
Individual services, represented as service interfaces within the service interface layer, represent application logic originating from different platforms.
Layers of Abstraction

- Leveraging the concept of composition, we can build specialized layers of services (towards Enterprise Service Bus principles)
- Each layer can abstract a specific aspect of our solution, addressing one of the issues we identified

The application service layer exists to express technology-specific functionality
Logical components of automation logic

The business service layer is comprised of business services, a direct implementation of the business service model.

eg:
- Purchase Order Service
- Billing Service
Logical components of automation logic

The orchestration service layer introduces a level of abstraction that alleviates the need for other services to manage interaction details required to ensure that service operations are executed in a specific sequence.

Orchestration languages (such as WS-BPEL) realize workflow management, bringing the business process into the service layer.
Service role

Service runtime classification depending on its processing responsibility in a given scenario (initiator - relayer - recipient of a message)
Service role

- **Service provider** role

  - is invoked via an external source

  - publish a service description (WSDL)
Service role

- **Service requester** role
  
  - invoke a service provider by sending msg
  
  - search the most suitable service provider studying available service descriptions
Service role

- **Service intermediator** role
  - forwarding to destination
  - passive: without altering content
  - active: process and alter message content, typically will lock for a particular SOAP header
    - e.g.: policy rule, load balancing, ...

- Service composition
  - Orchestration & choreography
Service Description

• Service Description as “contract” that can be used to build and validate messages
  • what kind of operation can I invoke on service X? - requester role
  • what kind of operation/request can I accept? - provider role

• WSDL Web Service Description Language
Service Description

Metadata that defines the only information made available to service requestors.
Service Description
Service Description

• WSDL - Web Server Description Language
  • Abstract description
    • interface characteristic without technology reference
  • Concrete description
    • connection to some real, implemented technology
WSDL

• Abstract Description - high level view of the service
  • definition - root element declaring namespace
  • types - where XML Schema is placed, to simple data to complex business document
  • example -> echo and ping operations
WSDL

• Abstract Description

• messages designed to receive or transmit

```xml
<wsdl:message name="echoRequestMessage">
  <wsdl:part name="part1" element="ns1:echoRequest"/>
</wsdl:message>

<wsdl:message name="echoResponseMessage">
  <wsdl:part name="part1" element="ns1:echoResponse"/>
</wsdl:message>

<wsdl:message name="pingRequestMessage">
  <wsdl:part name="part1" element="ns1:pingRequest"/>
</wsdl:message>
```
WSDL

<wsdl:types>
    <xs:schema targetNamespace="http://org.apache.axis2/xsd"
        elementFormDefault ="unqualified" attributeFormDefault="unqualified">
        <xs:element name="pingRequest">
            <xs:complexType>
                <xs:sequence>
                    <xs:element type="xs:anyType" name="element"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="echoRequest">
            <xs:complexType>
                <xs:sequence>
                    <xs:element type="xs:anyType" name="element"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>

        <xs:element name="echoResponse">
            <xs:complexType>
                <xs:sequence>
                    <xs:element type="xs:anyType" name="return"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
    </xs:schema>
</wsdl:types>
WSDL

• Abstract Description - high level view of the service

• **portType** (collection of) -> **operation**

```xml
<wsdl:portType name="MyServicePort">
  <wsdl:operation name="echo">
    <wsdl:input message="tns:echoRequestMessage"/>
    <wsdl:output message="tns:echoResponseMessage"/>
  </wsdl:operation>
  <wsdl:operation name="ping">
    <wsdl:input message="tns:pingRequestMessage"/>
  </wsdl:operation>
</wsdl:portType>
```

• **operation** is not (only) a method mapping
**WSDL**

- Concrete Description

  - **binding** -> concrete binding to SOAP

  ```xml
  <wsdl:binding name="MyServiceBinding" type="tns:MyServicePort">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <wsdl:operation name="echo">
      <soap:operation soapAction="echo"/>
      <wsdl:input>
        <soap:body use="literal" namespace="http://www.org.apache.axis2"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" namespace="http://www.org.apache.axis2"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  ```

explained later
WSDL

- Concrete Description
  
  - `service` -> physical address at which access service
  
  - `port` -> location information

```xml
<wSDL:service name="MyService">
  
  <wSDL:port name="MyServicePortType0"
    binding="tns:MyServiceBinding">
    <soap:address location="http://localhost:8080/MyService"/>
  
  </wSDL:port>
</wSDL:service>
```
WSDL Semantic *(pills)*

- ...and what about semantic
  - how a service behaves under certain conditions
  - how service will respond to specific conditions
  - what specific tasks the service is most suited for
- OWL - OWLS (think about)
  - no standardized solution yet
Which style of WSDL should I use?

- In relation to WSDL binding to SOAP
  - RPC/encoded
  - RPC/literal
  - Document/encoded
  - Document/literal
- Following the example
  - myMethod operation with parameters (integer x, float y)
Which style of WSDL should I use?

- RPC/encoded - void myMethod(int x, float y)

**WSDL**

```xml
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
  </operation>
</portType>

binding .../>
```

**SOAP**

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:float">5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

*overhead*

*op. name*

*not WS-I compliant*
Which style of WSDL should I use?

- RPC/literal - void myMethod(int x, float y)

**WSDL**

```xml
<message name="myMethodRequest">
  <part name="x" type="xsd:int"/>
  <part name="y" type="xsd:float"/>
</message>

<portType name="PT">
  <operation name="myMethod">
    <input message="myMethodRequest"/>
  </operation>
</portType>

<binding .../>
```

**SOAP**

```xml
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

WS-I compliant

op. name
Which style of WSDL should I use?

• Document/literal

WSDL

```xml
<types>
  <schema>
    <element name="xElement" type="xsd:int"/>
    <element name="yElement" type="xsd:float"/>
  </schema>
</types>

<message name="myMethodRequest">
  <part name="x" element="xElement"/>
  <part name="y" element="yElement"/>
</message>
```

XML-Schema

```xml
<xElement>5</xElement>
<yElement>5.0</yElement>
```

SOAP

```xml
<soap:envelope>
  <soap:body>
    <xElement>5</xElement>
    <yElement>5.0</yElement>
  </soap:body>
</soap:envelope>
```

not WS-I compliant
Which style of WSDL should I use?

- Document/literal wrapped

WSDL

```xml
<types>
  <schema>
    <element name="myMethod">
      <complexType>
        <sequence>
          <element name="x" type="xsd:int"/>
          <element name="y" type="xsd:float"/>
        </sequence>
      </complexType>
    </element>
  </schema>
</types>

<message name="myMethodRequest">
  <part name="parameters" element="myMethod"/>
</message>
```

XML-Schema

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

SOAP

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x>5</x>
      <y>5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

WS-I compliant
WSDL binding SOAP

• Concrete Description

• binding -> concrete binding to SOAP

<wsdl:binding name="MyServiceBinding" type="tns:MyServicePort">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
  <wsdl:operation name="echo">
    <soap:operation soapAction="echo" />
    <wsdl:input>
      <soap:body use="literal" namespace="http://www.org.apache.axis2"/>
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" namespace="http://www.org.apache.axis2"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
SOAP

- Messaging Framework Specification
- Simple Object Access Protocol
  - originally designed to replace proprietary RPC protocols ->
    serialization of object
  - now the purpose is to define a standard message format !!!
  - extremely flexible and extensible
- The RPC-Style messages runs contrary to the SOA principle.
SOAP

• Each message packaged in ENVELOPE

• Header - area dedicated to hosting meta information --> WS-*

• Body - XML formatted data, is the message payload

• Message have high level of independence --> robustness and extensibility

• Fundamental in a loosely coupled env.
SOAP

• The SOAP Nodes

  • sender
  • receiver
  • intermediary
  • initial
  • ultimate

• Remember the model!!
SOAP & WSDL

• Processing of SOAP message using concrete definition
UDDI (pills)

- **Service description advertisement and discovery**
  UDDI V2.0 specifications approved as an OASIS Standard

- Not yet commonly implemented
MEPs - message exchange patterns

- *Interaction* between services
  - as result of engineering interaction
- A group of already mapped out sequence for the exchange of messages
- Simple MEPs as building block for Complex MEPs
MEPs - message exchange patterns

- Primitive MEPs
  - request-response
    - correlation concept
  - define synchronous communication
- fire and forget
  - single destination - multicast - broadcast
MEPs - message exchange patterns

- Primitive MEPs
MEPs - message exchange patterns

- Complex MEPs --> e.g.: publish-and-subscribe
MEPs - message exchange patterns

• Blocking or not blocking? Technical issue

  • only for request-response pattern

  • in a dual transport like Http is a client matter -> but Long Time Transaction?

  • two separate transport connection for request and response is a client and service matter --> WS-*

  • WS-Addressing (later)
MEPs And WSDL

- In WSDL 1.1 terms
  - Request-Response -> WS-I ok
  - Solicit-Response -> WS-I ok
  - One-way operation -> WS-I ko
  - Notification Operation -> WS-I ko
MEPs And WSDL

- In WSDL 2.0 terms

  - In-out pattern = Request-Response
  - Out-in pattern = Solicit-Response
  - In-only pattern = One-way operation
  - Out-only pattern = Notification Operation
  - Robust in-only -> fault message from receiver are allowed
  - In-optional-out pattern -> the response is optional
SOA Platform

- Basic platform building block
SOA Platform

- Common SOA platform layer
SOA Platform
Service Processing task

- Service provider are expected to
  - supply a public interface (WSDL)
  - receive a SOAP message from requester
  - processing the header block within SOAP m.
  - validate and parse payload of SOAP m.
  - transform payload in a different format
  - encapsulate business processing logic
Service Processing task

- Service provider are expected to

  - assemble SOAP message containing the response to the original request SOAP

    - WS-Addressing and correlation

- transform the contents of the message back into the form expected by the requestor

- transmit the response SOAP
Service Processing task

- Service requester are expected to
  - contain business processing logic that calls a service provider
  - interpret a service provider’s WSDL definition
  - assemble a SOAP request in compliance with service provider WSDL definition
  - trasmitt SOAP request message to service provider
Service Processing task

- Service requester are expected to
  - receive a SOAP response message
  - validate and parse the SOAP response
  - transform payload in a different format
  - process SOAP header block
Service Processing task

Service provider
Service Processing task

Service requester

![Diagram of service processing task]

- soap request
- soap response
- business logic
- message processing logic
- service requestor
- proxy
- transformation
- validation
- other
- header processing
SOA support in J2EE
SOA support in J2EE

• **Java API for XML Processing (JAXP)** This API is used to process XML document content using a number of available parsers. Both Document Object Model (DOM) and Simple API for XML (SAX) compliant models are supported, as well as the ability to transform and validate XML documents using XSLT stylesheets and XSD schemas.

• **Java API for XML-based RPC (JAX-RPC)** The most established and popular SOAP processing API, supporting both RPC-literal and document-literal request-response exchanges and one-way transmissions. Example packages that support this API include:

• **Java API for XML Registries (JAXR)** An API that offers a standard interface for accessing business and service registries. Originally developed for ebXML directories, JAXR now includes support for UDDI.

• **Java API for XML Messaging (JAXM)** An asynchronous, document-style SOAP messaging API that can be used for one-way and broadcast message transmissions (but can still facilitate synchronous exchanges as well).

• **SOAP with Attachments API for Java (SAAJ)** Provides an API specifically for managing SOAP messages requiring attachments. The SAAJ API is an implementation of the SOAP with Attachments (SwA) specification.

• **Java Architecture for XML Binding API (JAXB)** This API provides a means of generating Java classes from XSD schemas and further abstracting XML-level development.

• **Java Message Service API (JMS)** A Java-centric messaging protocol used for traditional messaging middleware solutions and providing reliable delivery features not found in typical HTTP communication.
SOA support in J2EE
SOA support in .NET
Bibliography

- Web Service Architecture W3C working group: http://www.w3.org/TR/ws-arch

- Service-Oriented Architecture Concept, Technology, and Design - Thomas Erl - Prentice Hall PTR

- Enterprise Service Bus - David A. Chappell - O’REILLY

- Some article from: http://www-128.ibm.com/developerworks/webservices