# *Lecture 7.1 – Service-oriented Architectures*

Luca Morandini
Data Architect – AURIN Project
University of Melbourne
luca.morandini@unimelh.edu.au

## Outline of the Lecture

### Part 1: Introduction to SOA

- What is an architecture
- Why SOA
- Design principles
- SOAP-WS vs ReST

### Part 2: SOAP-WS

- UDDI: service discovery
- WDSL: service description
- SOAP: service fruition

### Part 3: SOAP Example

## Part 1: Introduction to SOA

## What's in an Architecture?

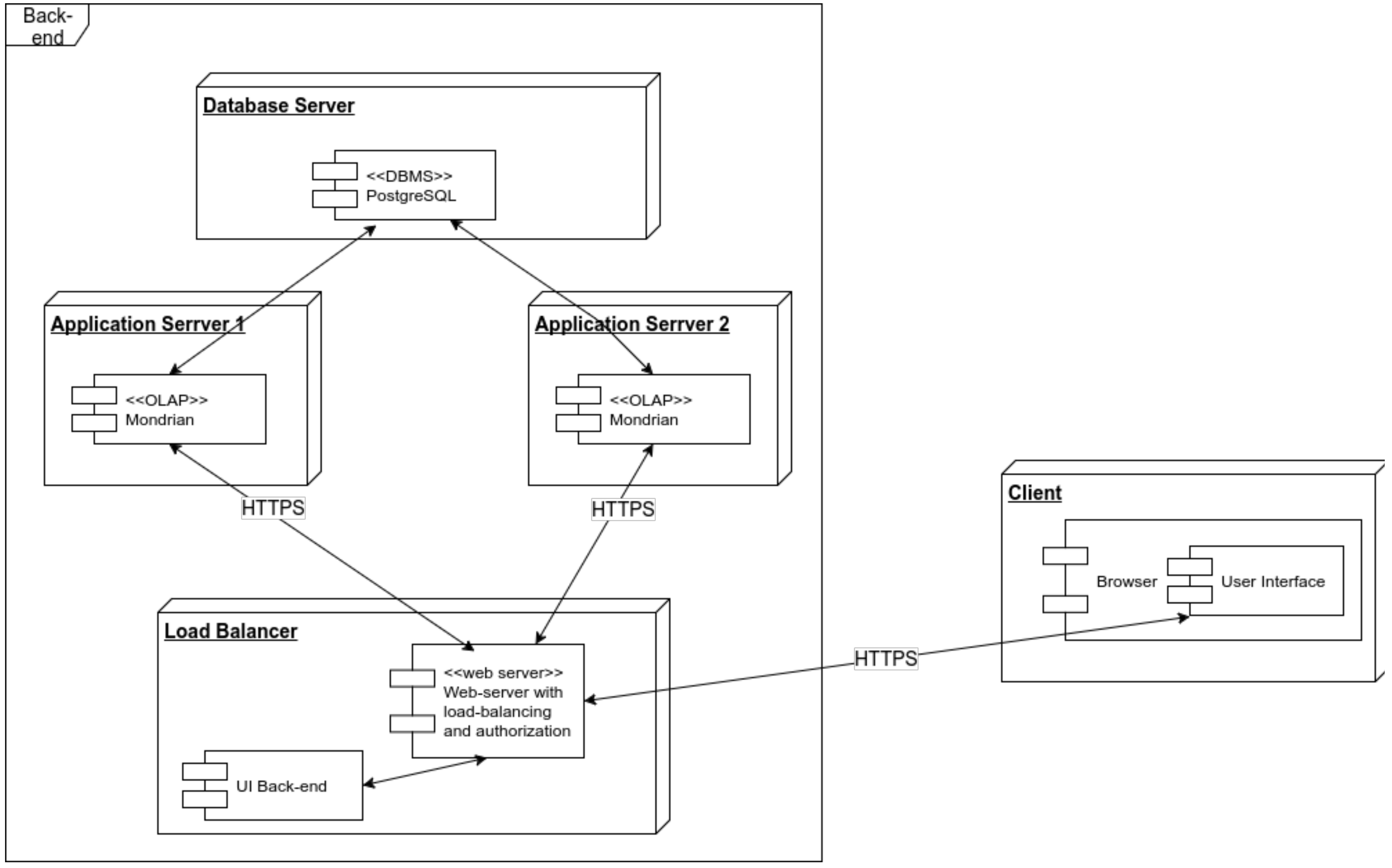
A (system) architecture, is just the way different software components are distributed on computers, and the way in which they interact with each other

Architectures are often difficult to describe in words, hence diagrams are often used

A standard graphic way to describe architectures is through the *Unified Modeling Language deployment diagram*

To draw a UML diagram tools such as the web-based *draw.io* tool, or UML Designer (open source) are often used. Other solutions are also widely available.

# Example of a Deployment Diagram



## Why SOA?

When an architecture is completely contained within the same machine, components communicate through *function calls* or *object instantiations*. For example:

- `fd = open(name, flags, perms)`
- `c=DriverManager.getConnection(db_url,user,pass)`

However, when components are distributed, function calls and object instantiations cannot always be used directly.

Therefore, components (more properly, *systems*) have to interact in more loosely-coupled ways. *Services* are often used for this.

Every system in a SOA should be considered as autonomous, but network-reachable and inter-operable through services.

## SOA Core Ideas

A Service Oriented Architecture (SOA) has several core ideas (IBM):

- A set of services that a business wants to provide to their customers, partners, or other areas of an organization
- An architectural pattern that requires a service provider, mediation, and service requestor with a service description
- A set of architectural principles, patterns and criteria that address characteristics such as modularity, encapsulation, loose coupling, separation of concerns, reuse and composability
- A programming model complete with standards, tools and technologies that supports web services, ReST services or other kinds of services
- A middleware solution optimized for service assembly, orchestration, monitoring, and management

## SOA Principles #1

- **Standardized service contract:** Services adhere to a communications agreement, as defined collectively by one or more service-description documents.
- **Service loose coupling:** Services maintain a relationship that minimizes dependencies and only requires that they maintain an awareness of each other.
- **Service abstraction:** Beyond descriptions in the service contract, services hide logic from the outside world.
- **Service reusability:** Logic is divided into services with the intention of promoting reuse.
- **Service autonomy:** Services have control over the logic they encapsulate.
- **Service statelessness:** Services minimize resource consumption by deferring the management of state information when necessary
- **Service discoverability:** Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
- **Service composability:** Services are effective composition participants, regardless of the size and complexity of the composition.

## SOA Principles #2

- **Service granularity**: A design consideration to provide optimal scope at the right granular level of the business functionality in a service operation.
- **Service normalization**: Services are decomposed and/or consolidated to a level of normal form to minimize redundancy. In some cases, services are denormalized for specific purposes, such as performance optimization, access, and aggregation.
- **Service optimization**: All else equal, high-quality services are generally preferable to low-quality ones.
- **Service relevance**: Functionality is presented at a granularity recognized by the user as a meaningful service.
- **Service encapsulation**: Many services are consolidated for use under a SOA. Often such services are not planned to be under a SOA.
- **Service location transparency**: The ability of a service consumer to invoke a service regardless of its actual location in the network.

## SOA for the Web: Web Services

- Web services can implement a service-oriented architecture
- Two main flavors
  - ReSTful Web Services
  - SOAP/WS

- Both uses HTTP, hence can run over the web (although SOAP/WS can run over other protocols as well)
- They are by far the most used (especially ReST) but not the only ones:
  - Geospatial services (WFS, WMS, …) are a kind of SOAP service
  - Health services (HL7)
  - SDMX (Statistical Data Markup eXchange (ABS))

# SOAP/WS vs ReST

- These are two different architectural design to call services over HTTP
- SOAP/WS is built upon the paradigm of the *Remote Procedure Call*; practically, a language independent function call that spans another system
- ReST is centered around *resources*, and the way they can be manipulated (added, deleted, etc.) remotely
- ReST is more of a style to use HTTP than a separate protocol,
- ...while SOAP/WS is a stack of protocols that covers every aspect of using a remote service, from service discovery, to service description, to the actual request/response

## Part 1: SOAP/WS

## UDDI & WS-*

The *Uniform Description Discovery and Integration* is a protocol to access a registry of services.

WS-* Refers to additional attendant standards for SOAP web services, e.g.

- WS-Security
- WS-ReliableMessaging
- WS-Notification
- WS-Addressing
- ..

These extensions go in the SOAP headers for additional functionality

# Example of UDDI

```
<tModel authorizedName="..." operator="..." tModelKey="...">
  <name>HertzReserveService</name>
  <description xml:lang="en">
    WSDL description of the Hertz reservation service interface
  </description>
  <overviewDoc>
    <description xml:lang="en">
      WSDL source document.
    </description>
    <overviewURL>
      http://mach3.ebphost.net/wsdl/hertz_reserve.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4"
      keyName="uddi-org:types" keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```

## WSDL

The *Web Services Description Language* is an XML-based interface description language that is used for describing the functionality offered by a web service.

WSDL provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns

## Example of a WSDL

```
<definitions name="HelloService"
  targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <portType name="Hello_PortType">
    <operation name="sayHello">
      <input message="tns:SayHelloRequest"/>
      <output message="tns:SayHelloResponse"/>
    </operation>
  </portType>
  <message name="SayHelloRequest">
    <part name="firstName" type="xsd:string"/>
  </message>
  <message name="SayHelloResponse">
    <part name="greeting" type="xsd:string"/>
  </message>
```

## SOAP

The *Simple Object Access Protocol* protocol specification for exchanging structured information over the web.

Let's suppose we want to get the quotation of the “MicroSoft” stock from a stock-exchange service

```
<SOAP-ENV:Envelope xmlns:
  SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle="http://www.w3.org/2001/12/soap-encoding" >
<SOAP-ENV:Body xmlns:m="http://www.xyz.org/quotations" >
      <m:GetQuotation>
         <m:QuotationsName>MicroSoft</m:QuotationsName>
      </m:GetQuotation>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The verbosity comes from the need to exactly define the schema of the parameters to pass to the remote function.

# How to Use SOAP

- Get Hold of the WDSL URL

  http:// www.webservicex.com/globalweather.asmx?

- Build a client based on this information (there are automated tools that do that)
- Execute a function (here a JavaScript function)

```
require("soap").createClient(
    "http://www.webservicex.com/globalweather.asmx?WSDL",
    function(err, client) {
   client.GlobalWeather.GlobalWeatherSoap12.GetCitiesByCountry ({
        CountryName : "Australia"
      }, function(err, result) {
        console.log(result.GetCitiesByCountryResult);
      });
    });
```

## References

- *UML Deployment Diagrams*
  http://agilemodeling.com/artifacts/deploymentDiagram.htm

- *Draw.io* (a web-based drawing tool)
  https://www.draw.io

- *UMLDesigner* (an Eclipse plugin)
  https://marketplace.eclipse.org/content/uml-designer