Service Oriented Architecture (SOA)

Intelligent Infrastructure Design for the Internet of Things

Antonio Navarro

References

- B.V. Kumar, P. Narayan, T. Ng, *Implementing SOA Using Java EE*, Addison-Wesley, 2010.
- Mark D. Hansen, *SOA Using Java Web Services*, Prentice Hall, 2007
- Russel Butek, *Which style of WSDL should I use?, 2003, https://www.ibm.com/developerworks/library/ws-whichwsdl/*
- Thomas Erl, *SOA Principles of Service Design*, Prentice Hall, 2008.

Index

Introduction

- Today, there are many solutions for business applications with industrial dimensions.
- All have advantages and disadvantages
- The most important thing is to be clear about when to apply each one.

Type of application	Solution	Use of frames	J2EE implementation examples		
			Pres.	**Business**	**Integration**
Corporate	Multilayer architecture	No	JSP	SAs POJOs + transfers	DAOs POJOs
		Yes	JSF	SAs POJOs + entities	JPA
+ Distributed Logic	RPC (Remote Procedure Call)	No	JSP (client)	SAs POJOs + RMI + transfers	DAOs POJOs
		Yes	JSF (client)	Session EJBs + entities	JPA
Heterogeneous platforms	+ SOA	No	JSP (client)	SAs POJOs + transfers + JAX WS / JAX RS	DAOs POJOs
		Yes	JSF (client)	Session EJBs JAX WS / JAX RS + entities	JPA

Applicable solution based on application requirements

Introduction

- We have already seen the multilayer architecture
- Remote object invocation in practice is closely linked to implementation platforms.
- We will now focus on service-oriented architectures, more independent of the implementation platforms

SOA

- Service-oriented architecture (SOA) is an architectural model that attempts to improve the efficiency, agility and productivity of an enterprise by placing services as the primary method through which the solution logic is represented.
- In principle, the architecture itself should be differentiated from the elements of service-oriented computing, but in many contexts it is used interchangeably.

SOA

- Two uses of SOA must be distinguished:
  - As a logic invocation mechanism for external systems, i.e., as an API to be used by the integration layer of another system
  - In addition, as a logic invocation mechanism for its own internal system

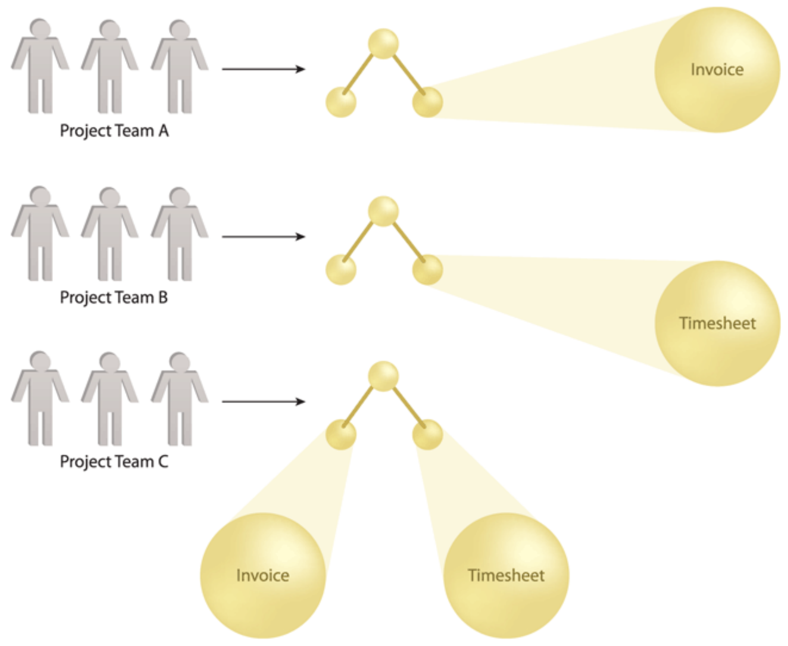
Objectives and benefits of SOA

- We can list the following:
  - Increased interoperability
  - Increase in the federation
  - Increasing platform diversification options
  - Greater alignment of business and technology
  - Increased return on investment
  - Increased organizational agility
  - Reduction of ICT burden

Objectives and benefits of SOA

- Increased interoperability
  - In this context, interoperability means data sharing between applications.
  - Exposing an application's services through SOA facilitates such interoperability.

Objectives and benefits of SOA

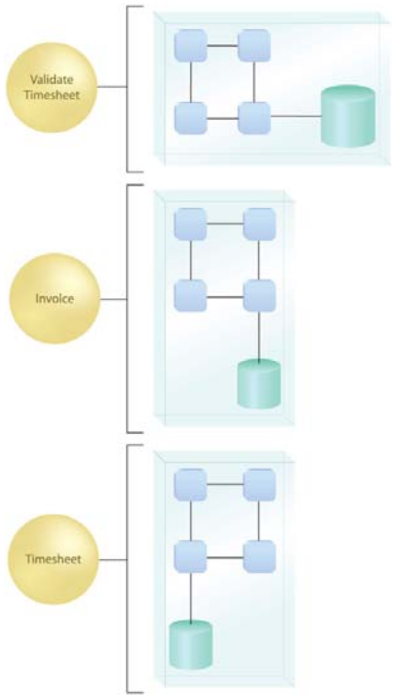


Increased interoperability in SOA

Objectives and benefits of SOA

- Increase in the federation
  - A federated computing environment is one where resources and applications are linked together while maintaining their autonomy and self-governance.
  - The exposure of services through SOA naturally favors the federation of functionalities.

Objectives and benefits of SOA

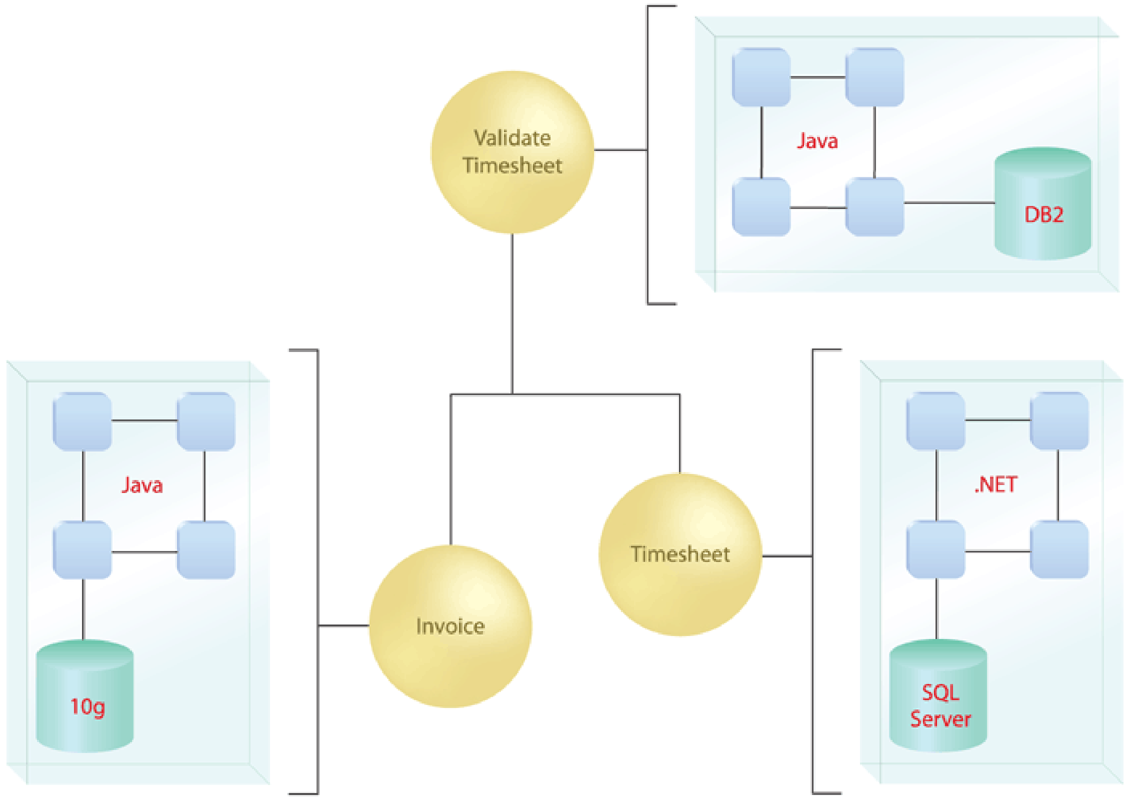


Services establishing a federated set of encapsulated functionalities

Objectives and benefits of SOA

- Increasing supplier diversification options
  - Diversification allows you to choose the best product from each supplier and use them in an integrated manner
  - SOA naturally allows the use of a vendor regardless of its implementation platform.

Objectives and benefits of SOA

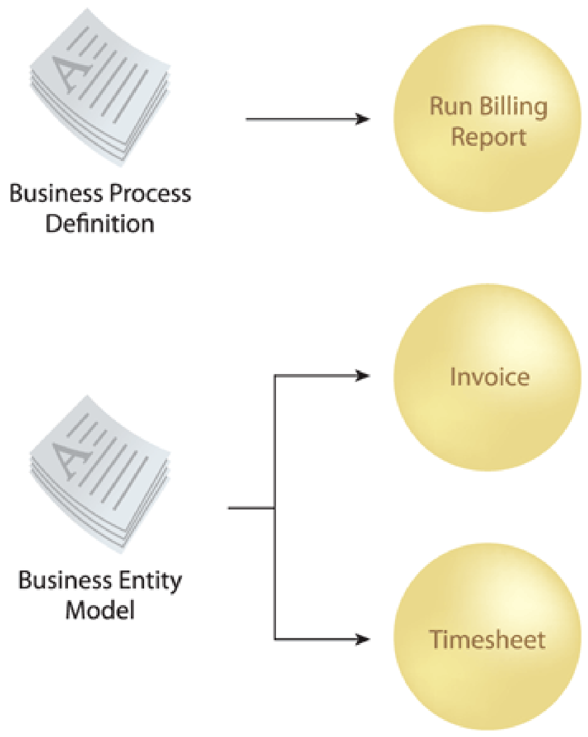


Services implemented on different platforms

Objectives and benefits of SOA

- Greater alignment of business and technology
  - Alignment aims to ensure that the IT system supports a company's changing requirements as quickly as possible.
  - To the extent that the system is designed around fine-grained services, the evolution of the system will be facilitated, and thus the alignment between business and technology will be facilitated.

Objectives and benefits of SOA

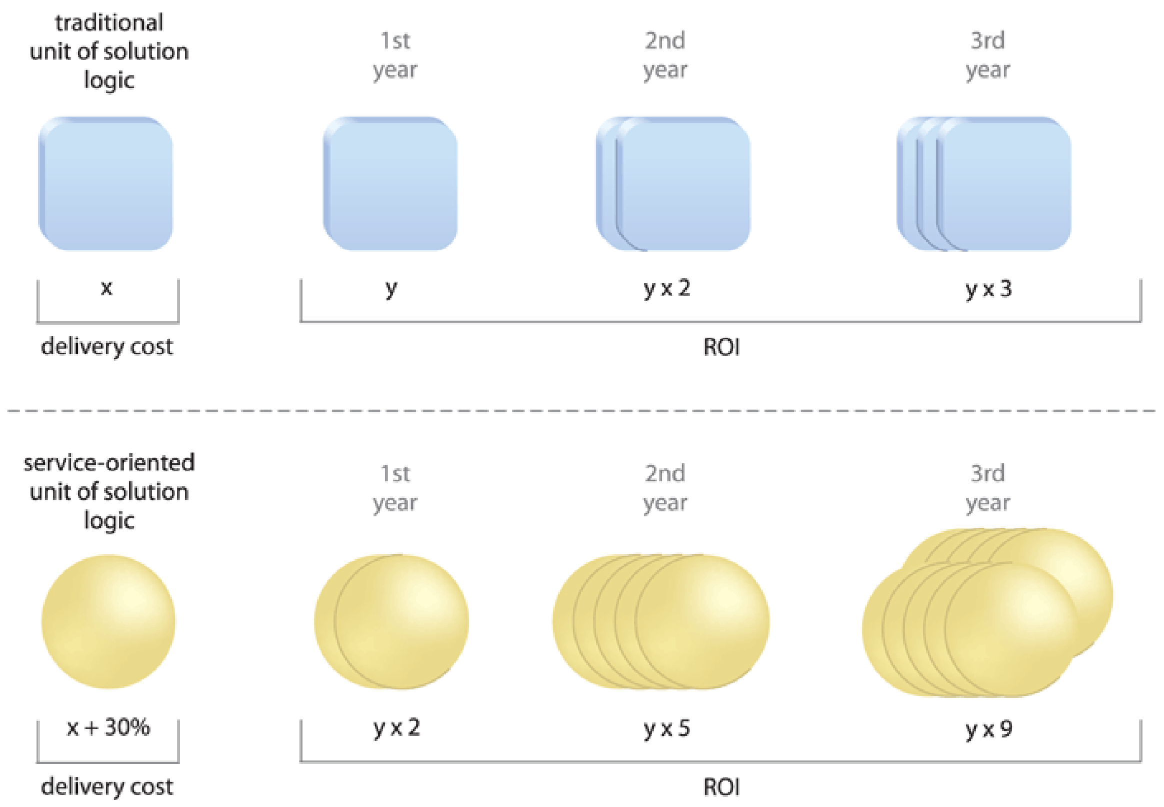


Fine-grained services favor evolution and hence alignment

Objectives and benefits of SOA

- Increased return on investment
  - The higher the return on investment, the higher the profitability of an organization.
  - The ability to reuse and evolve an SOA solution favors increased profits, despite a higher initial investment.

Objectives and benefits of SOA

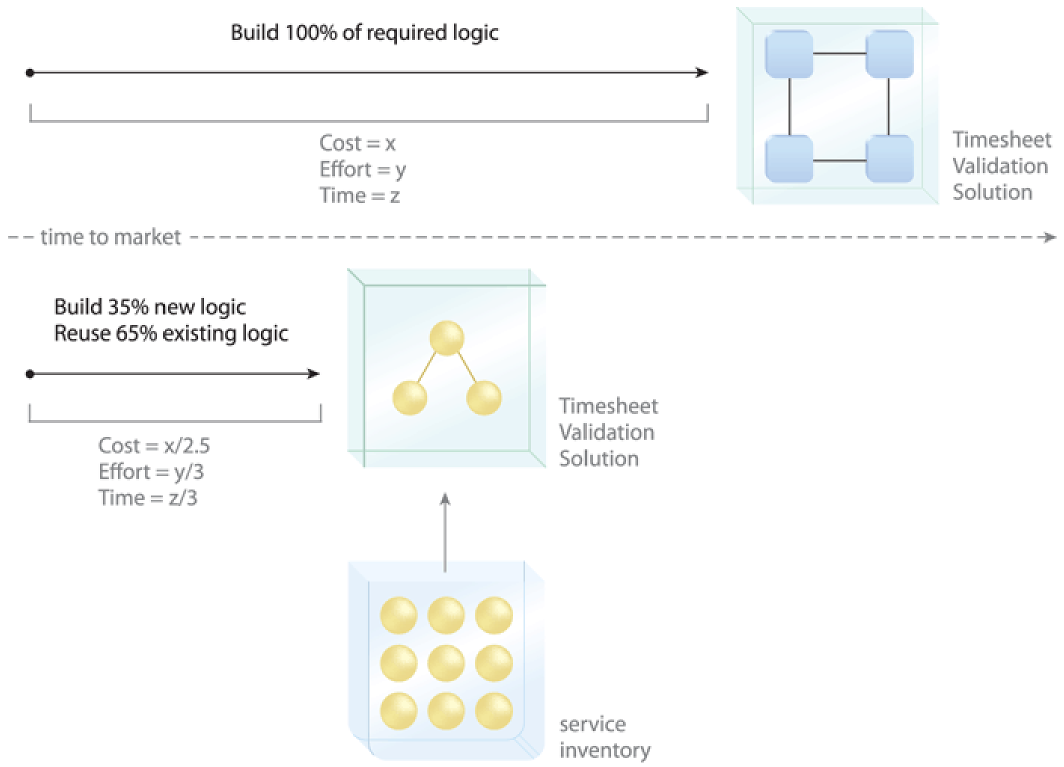


Comparison of return on investment in classic and SOA solutions

Objectives and benefits of SOA

- Increased organizational agility
  - Agility is the efficiency with which an organization can respond to change.
  - As long as there is a good IT support and alignment between business and system, things that favor SOA, the organization will be more agile.

Objectives and benefits of SOA

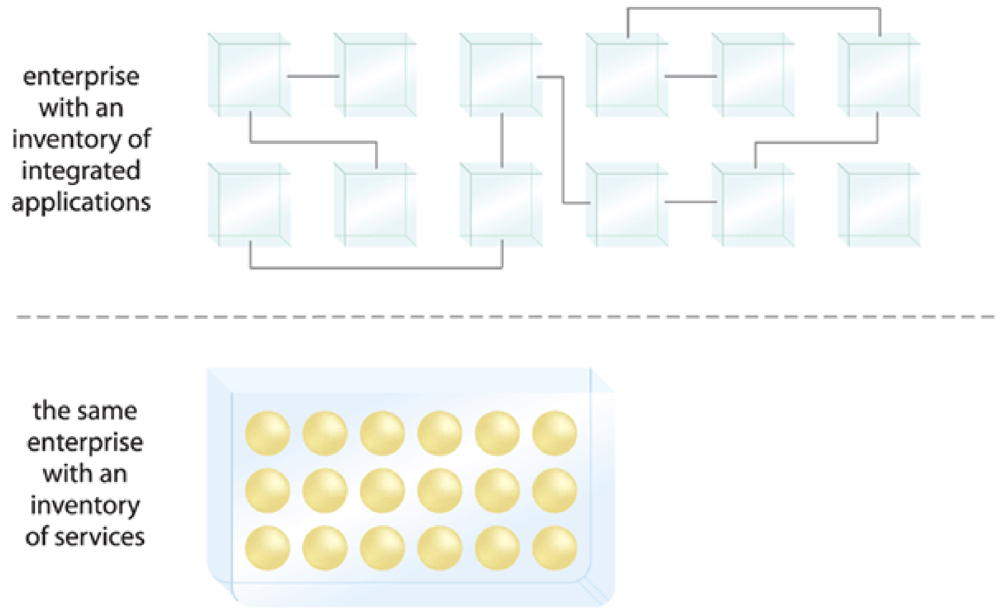


Comparison of changeover time in a traditional and SOA application

Objectives and benefits of SOA

- Reduction of the ICT burden
  - The lower the load of any component of a company, the higher the sales.
  - Service-oriented architecture allows for a more agile ICT department that reduces the burden on the organization.

Objectives and benefits of SOA


enterprise with an inventory of integrated applications
the same enterprise with an inventory of services

Service organization allows for less costly IT departments

SOA design principles

- There are a number of principles when designing services:
  - Standard service contract
  - Low service coupling
  - Service abstraction
  - Service reusability
  - Service autonomy
  - Stateless services
  - Ability to find services
  - Ability to compose services

SOA design principles

  - Ability to find the service
  - Service Componentization
- Standard service contract
  - Services express their purpose and capabilities through a service contract. For example:
    - WSDL
    - XML Schema
    - Description WS-Policy
  - This is the most important principle

SOA design principles

- Low coupling of services
  - Coupling is the level of connection or relationship between two elements.
  - Low coupling of services promotes no dependency between services, nor access to service implementations by jumping their interface.
- Service abstraction
  - This principle promotes concealing as many details of the service implementation as possible.
  - In other words, it promotes minimal and abstract interfaces

SOA design principles

- Reusability of services
  - This principle promotes the positioning of services as business resources independent of specific functional contexts.
  - Thus, the services are useful for more than one purpose
- Autonomy of services
  - This principle determines that services must be able to perform their functionality consistently and reliably.
  - For this, the implementation of such services must have a significant degree of control over its environment and resources.

SOA design principles

- Stateless services
  - Managing excessive status information can compromise the availability of a service and limit its potential for scale.
  - Therefore, the services should only have status in very specific and determined cases.

SOA design principles

- Ability to find services
  - To increase ROI and leverage services, services should make clear what their functionality is:
    - Purpose
    - Capabilities
    - Resource constraints
  - Such functionality should be made clear regardless of the existence of mechanisms such as service registries.

SOA design principles

- Ability to compose services
  - As the sophistication of SOA solutions grows, so does the complexity of service composition configurations.
  - Therefore, the ability to compose services is essential in advanced applications.

SOA design principles

- The above principles lack the concept of *interoperability.*
- It has not been made explicit, as it is assumed to be a fundamental requirement of SOA and each of the above principles.

SOA design principles

- Consistent application of these eight design principles produces designs with:
  - Improved consistency in the representation of functionality and data
  - Reduced dependencies between logical solution units
  - Reduced knowledge of logic design and its implementation
  - Increased ability to reuse a logical solution element for multiple purposes

SOA design principles

- Increased opportunity to combine logical solution units in different configurations
- Higher level of behavioral prediction
- Increased availability and scalability
- Increased knowledge of available logical solutions

SOA or not SOA

- There are some problems that non-SOA architectures can present:
  - There may be redundant functionality
  - This makes the development process inefficient.
  - It causes the existence of unnecessary functionality in the company.
  - This leads to heterogeneous platforms, complex infrastructures and convoluted architectures.
  - Integration becomes a constant challenge

SOA or not SOA

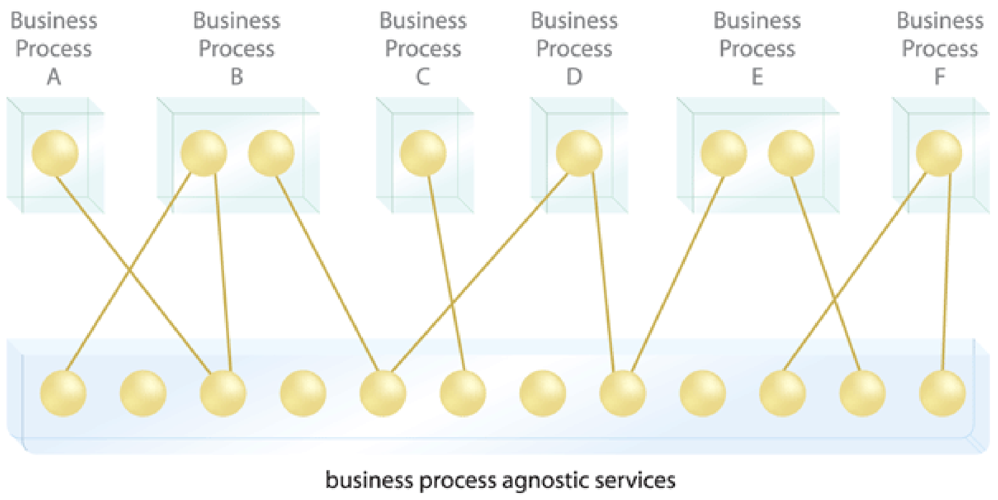
- The SOA architecture has several advantages:
  - Increased amount of solution-independent logic
  - Less application-specific logic
  - Reduction in logic volume
  - Inherent interoperability

Intelligent Infrastructure Design for IoT - MiOT UCM
Antonio Navarro

SOA or not SOA

- Increased amount of solution-independent logic
  - Within an SOA solution, units of logic (services) encapsulate functionality that is not specific to an application or business service.
  - These elements are reusable TICS elements that are independent of the solution.

SOA or not SOA

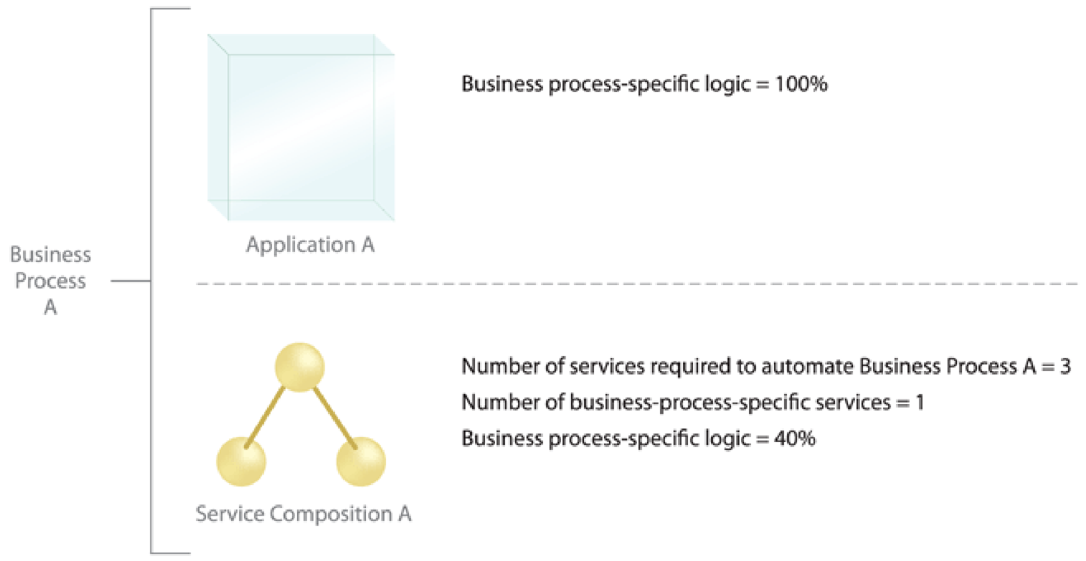


Services as reusable logic elements

SOA or not SOA

- Less application-specific logic
  - By making services that are not specific to an application or business, the amount of application-specific logic is decreased.
  - This blurs the concept of independent application.

SOA or not SOA


Business Process A
Application A
Business process-specific logic = 100%
Service Composition A
Number of services required to automate Business Process A = 3
Number of business-process-specific services = 1
Business process-specific logic = 40%

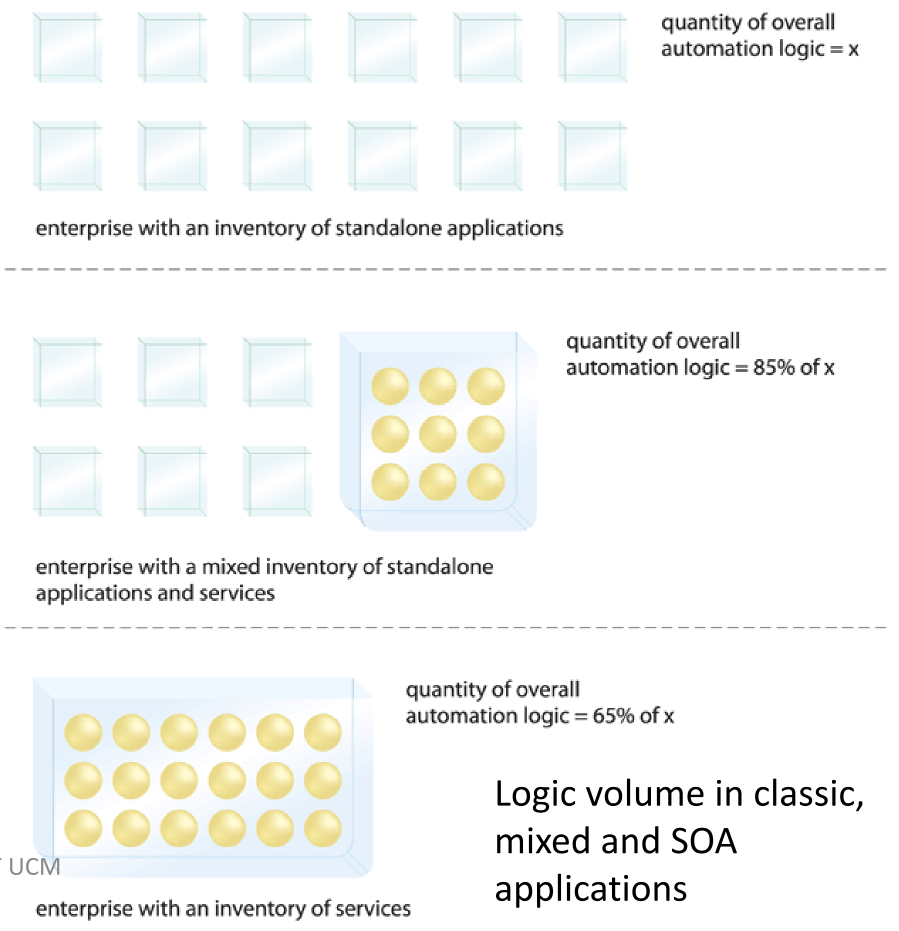
Solution-specific logic reduction

Intelligent Infrastructure Design for IoT - MiOT UCM
Antonio Navarro

SOA or not SOA

- Reduction in logic volume
  - The volume of the solution logic is reduced as services are shared and reused by different business processes.

SOA or not SOA

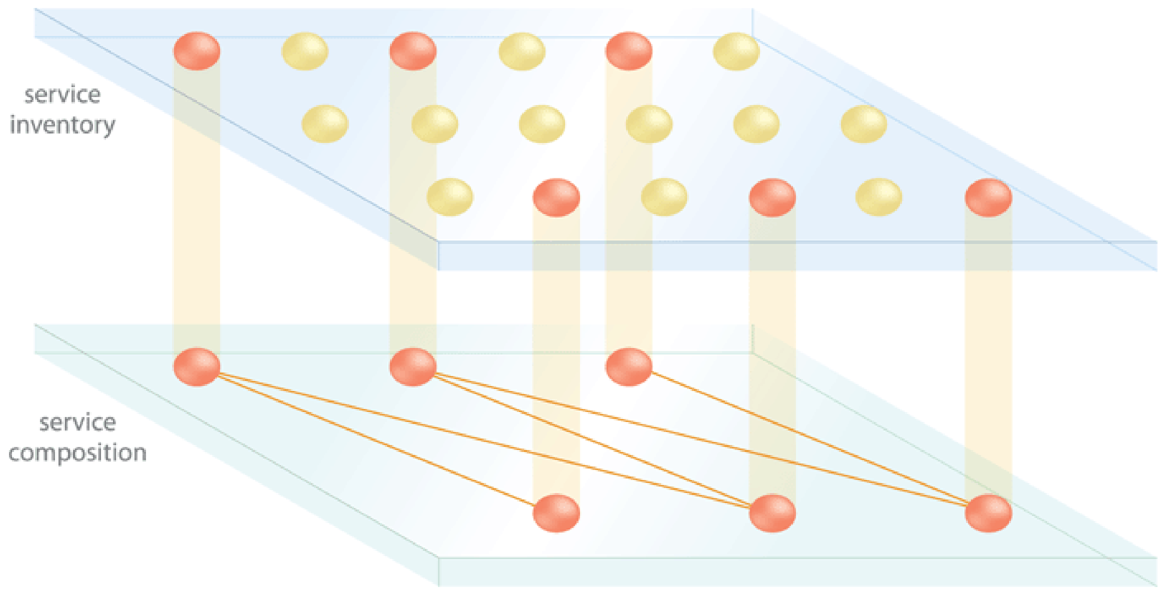

quantity of overall automation logic = x
enterprise with an inventory of standalone applications
quantity of overall automation logic = 85% of x
enterprise with a mixed inventory of standalone applications and services
quantity of overall automation logic = 65% of x
enterprise with an inventory of services

Logic volume in classic, mixed and SOA applications

SOA or not SOA

- Inherent interoperability
  - SOA solutions involve aligned logic solutions
  - Thus, service contracts and their underlying data models provide a mechanism to increase interoperability.

SOA or not SOA



Consistent set of services increases interoperability and composability

Intelligent Infrastructure Design for IoT - MiOT UCM
Antonio Navarro

SOA or not SOA

- SOA architecture also presents a number of challenges:
  - Increased design complexity
  - Need for design and architectural standards
  - Pre-listing of all services prior to their implementation
  - May hinder the application of agile methodologies.
  - Forces the presence of an authority in the management of services
- Therefore, an SOA solution should only be applied when strictly necessary.

Web services

- SOA is a generic architecture independent of concrete implementations.
- CORBA can be an SOA implementation
- It is common to speak of web services as a mechanism for SOA implementation.

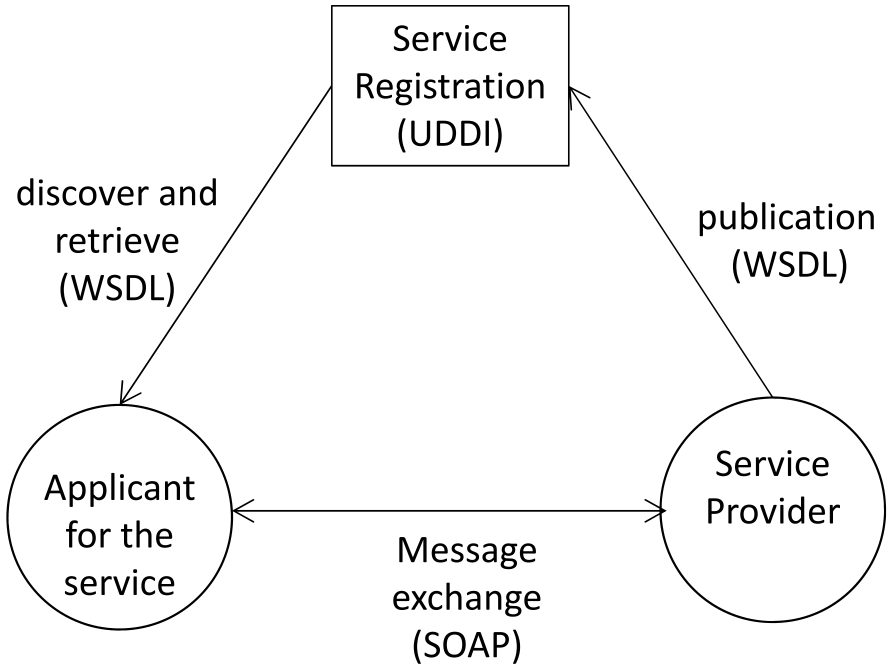
Web services

- There are several definitions, but I am not convinced by any of them
- I would say that a *web service* is an element of logic invocable over the network and whose client only knows the type of the input information output, and the implemented functionality, i.e., it does not know its implementation platform

Web services

- There is a first generation of web services characterized by the use of:
  - *Web Services Description Language* (WSDL)
  - *Simple Object Access Protocol* (SOAP)
  - *Universal Description, Discovery and Integration* (UDDI)
  - *WS-I Basic Profile*

Web services


Service Registration (UDDI)
discover and retrieve (WSDL)
publication (WSDL)
Applicant for the service
Service Provider
Message exchange (SOAP)

First generation of web services

Web services

- There is a second generation of web services based on extensions to the first generation web services. Among others:
  - *WS-Security Specification and Frameworks*
  - *WS-Addressing Specification*
  - *WS-Reliable Messaging Specifications*
  - *WS-Business Process Execution Language*
  - *WS-Choreography Definition Language*
  - *WS-Metadata Exchange Specifications*

Web services

- There are two implementations of web services:
  - REST Web Services
  - SOAP web services

	REST	SOAP
Message format	XML, JSON	XML within SOAP
Interface definition	it is not necessary	WSDL
Transportation	HTTP	HTTP, JMS, FTP, etc.

Differences between REST and SOAP web services

REST Web Services

- REST web services are based on Roy Fielding's Ph.
- REST-style, or RESTful, web services are referred to as RESTful
- They are based on a series of principles:
  - *RESTful services are stateless*. Each request from the client to the server must contain all the information necessary to understand the request, and cannot take advantage of any context stored on the server.

REST Web Services

– *Web services have a common interface*. This usually means that the only operations allowed are those provided by the HTTP protocol: `GET`, `POST`, `PUT` and `DELETE`.
– *REST architectures are built on resources that are uniquely identified by URIs*. Thus, each URI represents a different function and, by extension, a different data.
– *REST components manipulate resources by exchanging representations of them*. Thus, information is exchanged in XML format.

REST Web Services

- Example of REST web service invocation:

```
URL url = new URL("http://localhost:8080/RESTplano/saluda
                           ?
firstname=Charlton&lastname=Heston");
HttpURLConnection with = (HttpURLConnection)
url.openConnection();
con.setRequestMethod("GET");
InputStream in = con.getInputStream();
byte[] b = new byte[1024];
int result = in.read(b);
while (result != -1) {
                      System.out.write(b,0,result);
                      result =in.read(b); }
in.close();
con.disconnect();
```

REST Web Services

- Example of REST web service implementation:

```
public class Saluda extends HttpServlet {
 public void doGet(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
     String name = req.getParameter("name");
     String lastName= req.getParameter("lastName");

     GreetingWSB greetingWSB= new GreetingWSB();
      String reply= greetingWSB.greet(firstname, lastname);
res.getWriter(). write(reply);
     }
}
```

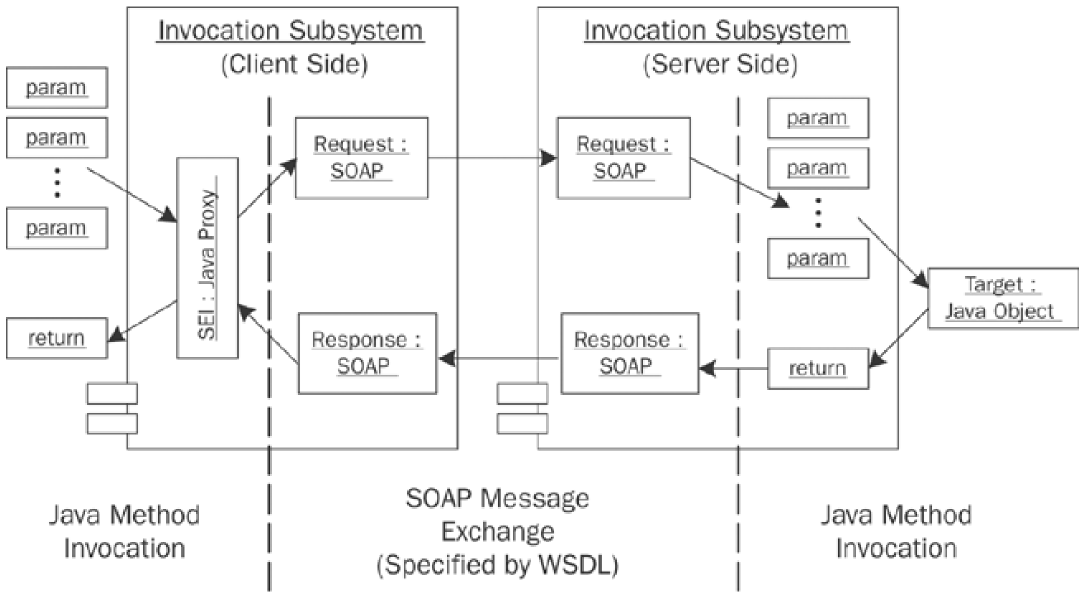
REST Web Services

- The above example does not use Java-specific frameworks for the implementation of REST web services, such as:
  - JAX-RS (*Java API for RESTful Web Services*)
  - JAXB (*Java Architecture for XML Binding*)

SOAP web services

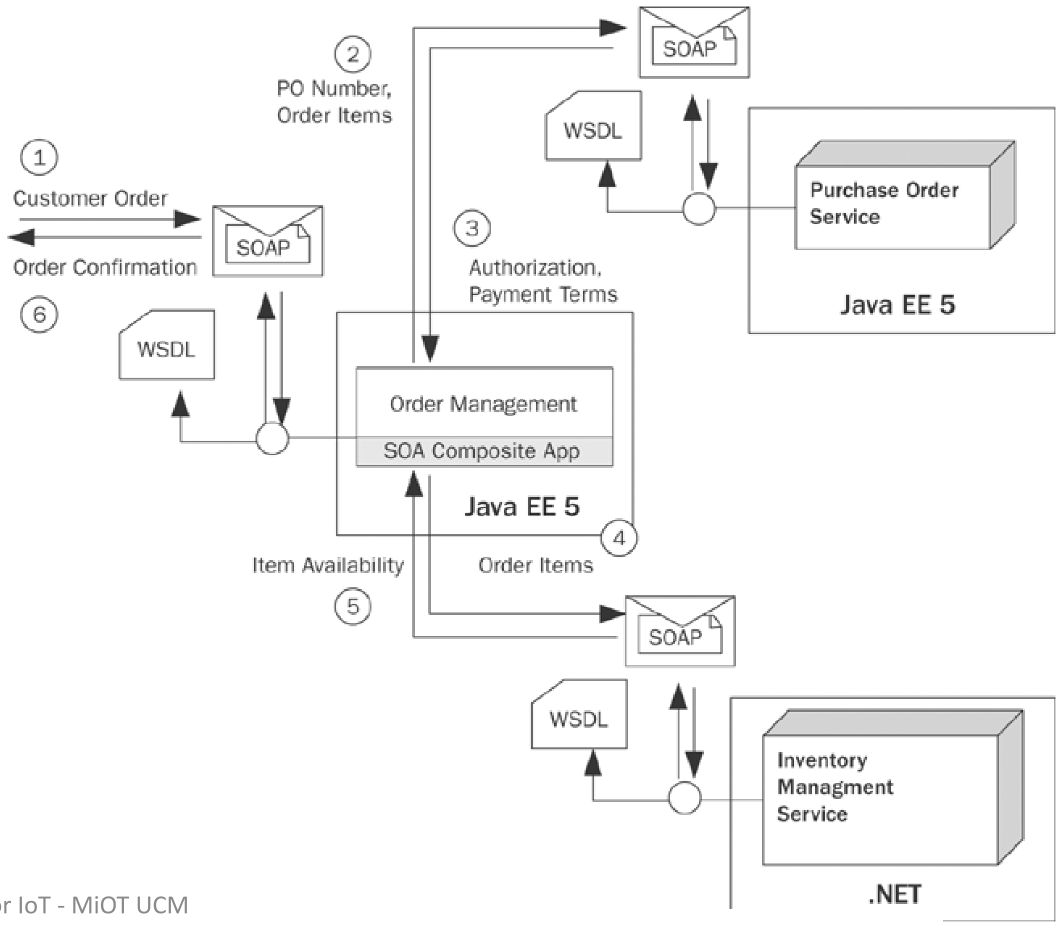
- SOAP web services use SOAP to
  - Invoking web services described through WSDL interfaces
  - Encode the input and output parameters of the invoked service in XML format.
- They require a mechanism to translate:
  - Service requests in logic elements
  - Parameters of a programming language in other languages

SOAP web services



SOAP web services invocation and response

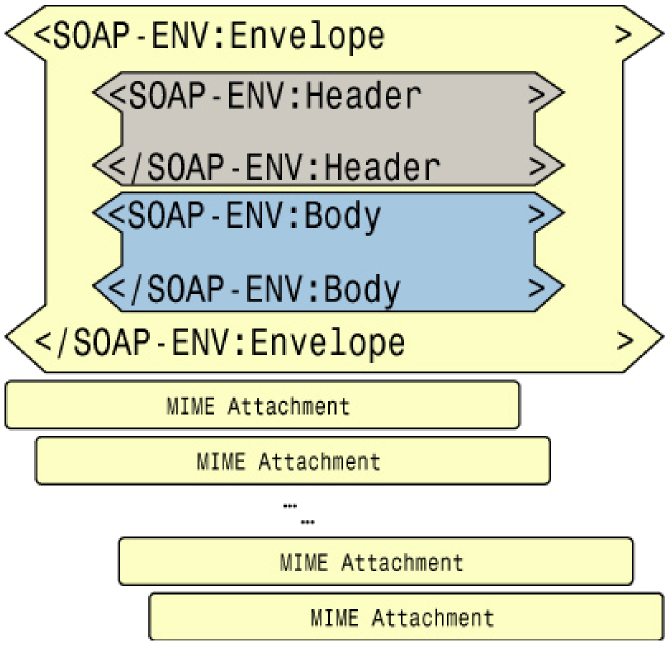
SOAP web services



Example of an application based on SOAP web services

SOAP web services

- The basic structure of a SOAP message is:



Basic structure of a SOAP message

SOAP web services

- Where:
  - `Envelope`: it is the root element. It stores namespace and coding style information.
  - `Header`: optional, but not usually omitted. They provide services to the cargo carried in the `Body` element.
  - `Body`: is mandatory. Contains the payload of the SOAP message to be processed by the destination endpoint.

SOAP web services

- SOAP request example

```
<env1:Envelope xmlns:env1="http://schemas.xmlsoap.org/soap/envelope">
 <env1:Body>
          < getord:getOrdersDates
          xmlns:getord="http://www.example.com/oms/getorders">
       <getord:startDate>2005-11-19</getord:startDate>
        <getord:endDate>2005-11-22</getord:endDate>
           </getord:getOrdersDates>.
 </env1:Body>
</env1:Envelope>
```

SOAP web services

- Example of SOAP response

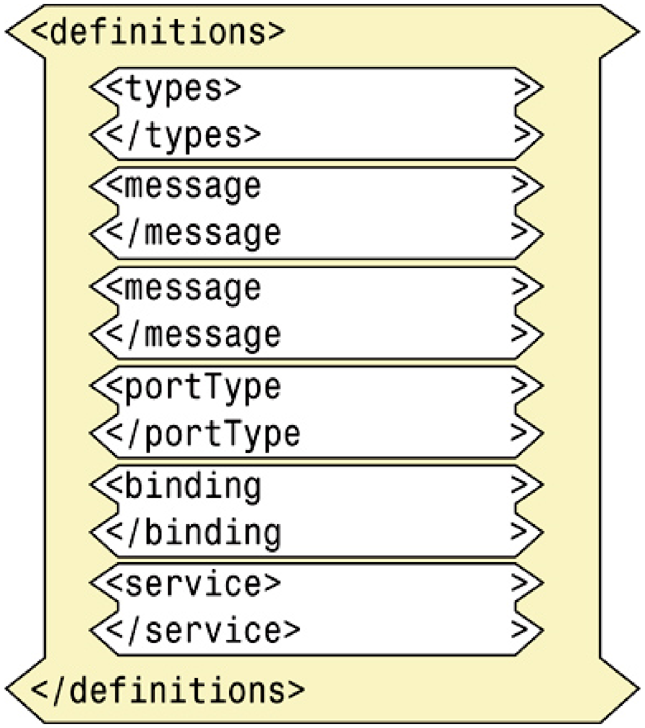
```
<env1:Envelope
xmlns:env1="http://schemas.xmlsoap.org/soap/envelope">
  <env1:Body>
        < getord:getOrdersDatesResponse
       xmlns:getord="http://www.example.com/oms/getorders">
              < Orders xmlns="http://www.example.com/oms"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://www.example.com/oms
          http://soabook.com/example/oms/orders.xsd">
```

SOAP web services

```
< Order>
      < OrderKey>ENT1234567</OrderKey>.
            < OrderHeader>
         <SALES_ORG>NE</SALES_ORG>.
         <PURCH_DATE>2005-11-20</PURCH_DATE>
         <CUST_NO>ENT0072123</CUST_NO>
         <PYMT_METH>PO</PYMT_METH>.
         <PURCH_ORD_NO>PO-72123-0007</PURCH_ORD_NO>
        <WAR_DEL_DATE>2006-12-20</WAR_DEL_DATE>
       </OrderHeader>
………………………………………………………………
```

SOAP web services

- The structure of a WSDL interface is:



WSDL interface structure

SOAP web services

- Where:
  - `definitions`: is the root element and contains the elements that define a web service.
  - `types`: acts as a container for data type definitions using XML schema definitions. These schemas define the data types of the information using `message` elements.

SOAP web services

- `message`: these elements characterize abstract typed definitions of the exchanged data. These elements reference the data types defined in `types`
- `portType`: this element describes one or more set of abstract operations supported by one or more endpoints. It uses `operation` elements to describe these operations, which refer to the values defined in `message` to characterize the function parameters.

SOAP web services

- `binding`: specifies a particular protocol for transport binding and a data format specification for a `portType` element. (Butek, 2005) is a good guide.
- `service`: identifies the web service in its `name` attribute. It models multiple web services through a collection of `port` elements, which indicate endpoints as a combination of transport bindings and network addresses.
  - `port`: provides the name and location of the system on which the web service is located. Models an individual web service

SOAP web services

- Example of WSDL interface:

```
<? xml version="1.0" encoding="UTF-8"?>
< wsdl:definitions name="SumaImplementationService"
targetNamespace="http://servicioAplicacion.negocio/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://servicioAplicacion.negocio/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  < wsdl:types>
        < schema xmlns="http://www.w3.org/2001/XMLSchema">
<import namespace="http://servicioAplicacion.negocio/"
schemaLocation="suma_schema1.xsd"/>
    </schema>
    </wsdl:types>
```

SOAP web services

```
< wsdl:message name="add">
       < wsdl:part name="parameters" element="tns:sum">
   </wsdl:part>
 </wsdl:message>
   < wsdl:message name="addResponse">
       < wsdl:part name="parameters" element="tns:sumResponse">
   </wsdl:part>
 </wsdl:message>
```

SOAP web services

```
  < wsdl:portType name="Sum">
        < wsdl:operation name="add">
            < wsdl:input name="sum" message="tns:sum">
            </wsdl:input>
            < wsdl:output name="sumarResponse"
message="tns:sumarResponse">
            </wsdl:output>
        </wsdl:operation>
    </wsdl:portType>
```

SOAP web services

```
 < wsdl:binding name="SumaImplementationServiceSoapBinding"
type="tns:Sum">
        < soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        < wsdl:operation name="add">
            < soap:operation soapAction="" style="document"/>
            < wsdl:input name="sum">
                < soap:body use="literal"/>
            </wsdl:input>
            < wsdl:output name="addResponse">
                < soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
```

SOAP web services

```
 < wsdl:service name="SumAddImplementationService">
        < wsdl:port name="SumaImplementacionPort"
binding="tns:SumaImplementacionServiceSoapBinding">
            < soap:address
location="http://localhost:55555/servicioSuma/services/SumaImplement
acionPort"/>
    </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

SOAP web services

- Example of WSDL interface

```
<? xml version="1.0" encoding="UTF-8"?>
< wsdl:definitions name="CalculoImplementacionService"
targetNamespace="http://servicioAplicacion.negocio/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:tns="http://servicioAplicacion.negocio/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">

 < wsdl:types>
        < schema xmlns="http://www.w3.org/2001/XMLSchema">
        < import namespace="http://servicioAplicacion.negocio/"
schemaLocation="calculo_schema1.xsd"/>
    </schema>
    </wsdl:types>
```

SOAP web services

```
< wsdl:message name="add">
       < wsdl:part name="parameters" element="tns:sum">
   </wsdl:part>
   </wsdl:message>
   < wsdl:message name="subtract">
       < wsdl:part name="parameters" element="tns:subtract">
   </wsdl:part>
   </wsdl:message>
```

SOAP web services

```
< wsdl:message name="addResponse">
       < wsdl:part name="parameters" element="tns:sumResponse">
   </wsdl:part>
   </wsdl:message>
   < wsdl:message name="subtarResponse">
       < wsdl:part name="parameters" element="tns:subtarResponse">
   </wsdl:part>
</wsdl:message>
```

SOAP web services

```
 < wsdl:portType name="Calculation">
        < wsdl:operation name="subtract">
            < wsdl:input name="subtract" message="tns:subtract">
      </wsdl:input>
            < wsdl:output name="subtarResponse"
message="tns:subtarResponse">
      </wsdl:output>
        </wsdl:operation>
        < wsdl:operation name="add">
            < wsdl:input name="sum" message="tns:sum">
      </wsdl:input>
            < wsdl:output name="sumarResponse"
message="tns:sumarResponse">
      </wsdl:output>
        </wsdl:operation>
</wsdl:portType>
```

SOAP web services

```
 < wsdl:binding name="CalculationImplementationServiceSoapBinding"
type="tns:Calculation">
        < soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        < wsdl:operation name="subtract">
            < soap:operation soapAction="" style="document"/>
            < wsdl:input name="subtract">
                < soap:body use="literal"/>
            </wsdl:input>
            < wsdl:output name="subtractResponse">
                < soap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
```

SOAP web services

```
< wsdl:operation name="add">
         < soap:operation soapAction="" style="document"/>
         < wsdl:input name="sum">
             < soap:body use="literal"/>
         </wsdl:input>
         < wsdl:output name="addResponse">
             < soap:body use="literal"/>
         </wsdl:output>
     </wsdl:operation>
 </wsdl:binding>
```

SOAP web services

```
 < wsdl:service name="CalculoImplementacionService">
        < wsdl:port name="CalculoImplementacionPort"
binding="tns:CalculoImplementacionServiceSoapBinding">
            < soap:address
location="http://localhost:55555/servicioCalculo/services/CalculoImple
mentacionPort"/>
    </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

SOAP vs. REST

- This is one of the most frequently asked questions in enterprise applications.
- The use of specific frameworks has brought the ease of use of REST much closer to SOAP, however it is easier to use SOAP services than REST, as long as the client language allows or facilitates it.
- Despite the use of frameworks, the SOAP middleware layer is more burdensome to applications than REST.

SOAP vs. REST

- Depending on whether REST is used (orthodoxly or flexibly), REST can perform only four types of operations or many. SOAP, it can always perform many
- SOAP and REST have transport layer (*point to point*) security using HTTPS
- SOAP has application layer security (*end to end*) and REST does not. However, it puts a lot of overhead on the server
- SOAP has distributed transactions and REST does not.

Service Oriented Architecture (SOA)

Intelligent Infrastructure Design for the Internet of Things

Antonio Navarro