Software Engineering and Service-Oriented Systems

– An Overview of (Web) Services –

Francesco Tiezzi

IMT - Institutions, Markets, Technologies

Institute for Advanced Studies Lucca

Lucca, Italy - September, 2012

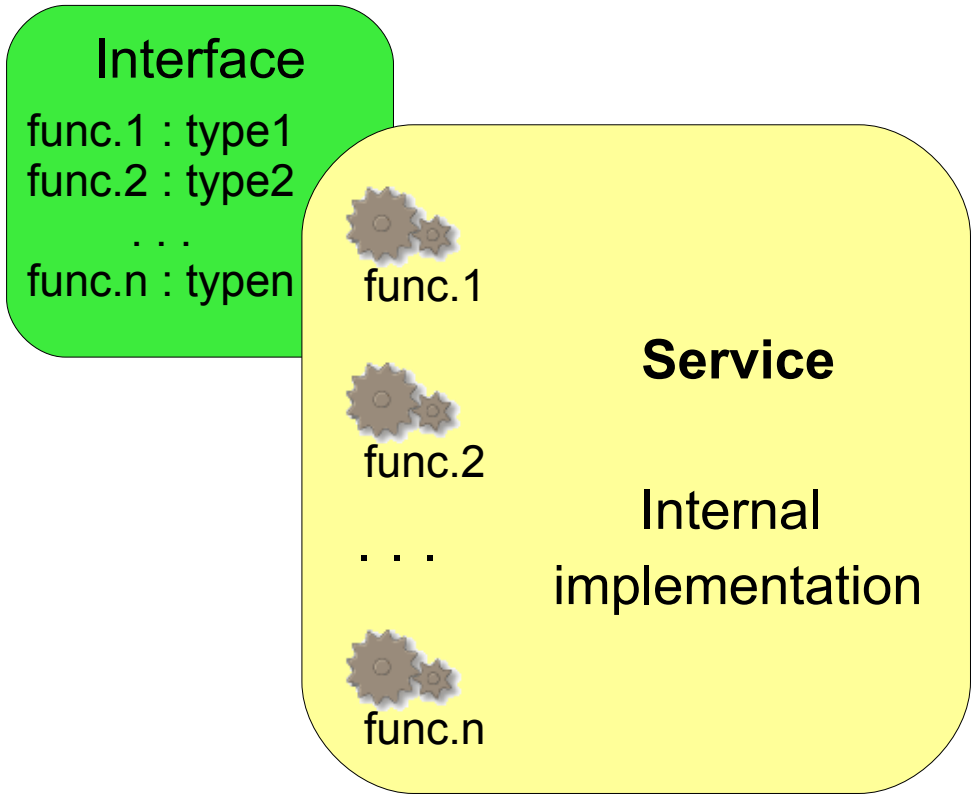
Setting the scene

- This course focusses on Service-Oriented Systems (SOSs)
- We will introduce the notion of:
  - Service-Oriented Computing as a paradigm for developing SOSs
  - Service as a basic block for building SOSs

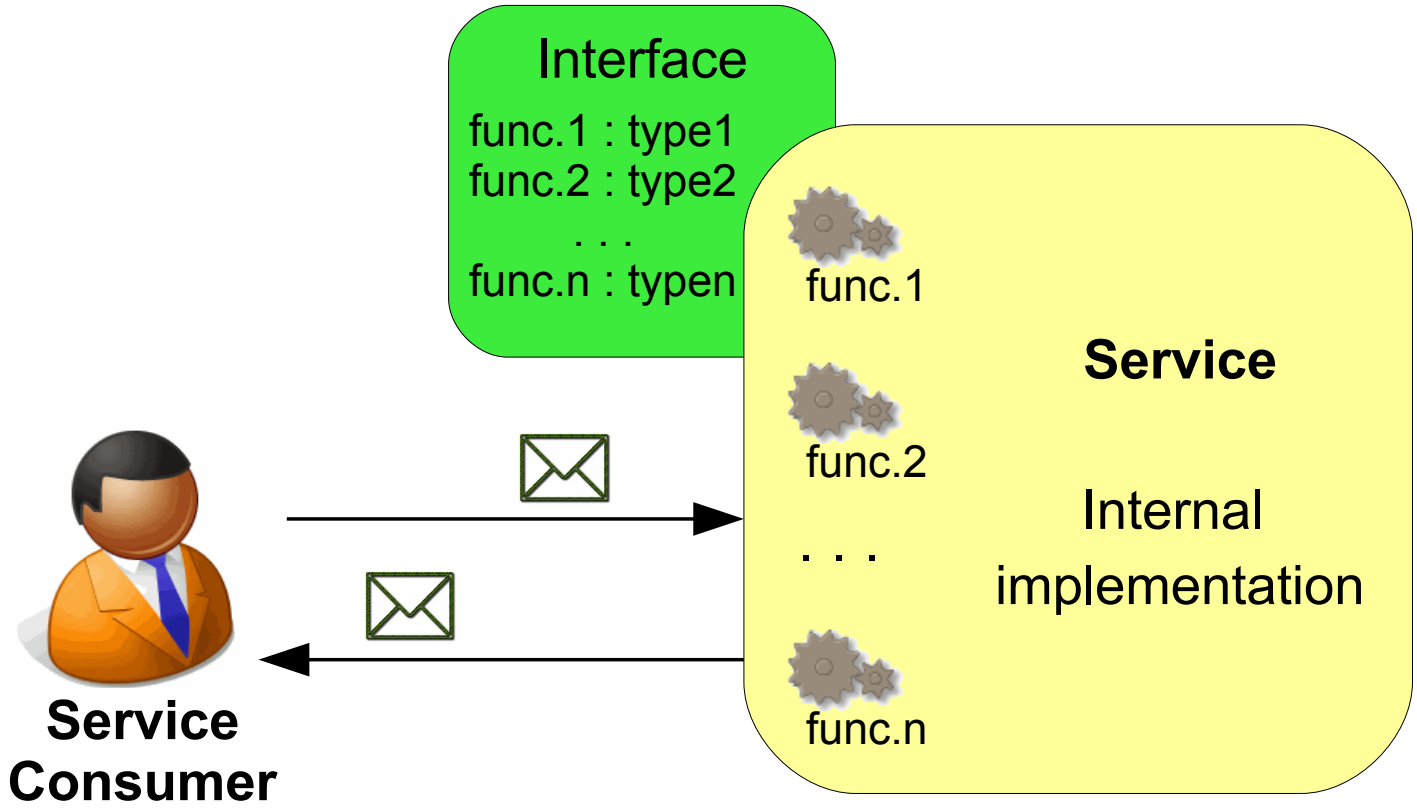
Service-Oriented Computing (SOC)

- A compute paradigm for distributed and e-business computing
- Aims at enabling developers to build networks of integrated and collaborative applications
  through the use of *loosely coupled*, *platform-independent*, *reusable* components (called **services**)
- A modern attempt to cope with old problems related to information interchange, software integration, and B2B
  - Finds its origin in object-oriented and component-based software development
- **Service-Oriented Architecture** (SOA): an architectural style to realize SOC

Service


Interface
func.1 : type1
func.2 : type2
. . .
func.n : typen
func.1
func.2
. . .
func.n
Service
Internal
implementation

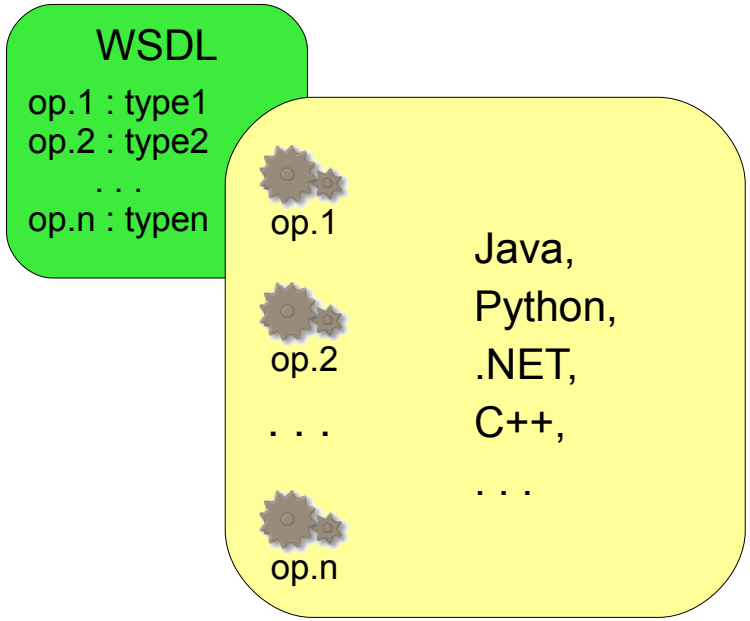
Service


Interface
func.1 : type1
func.2 : type2
. . .
func.n : typen
func.1
func.2
. . .
func.n
Service
Internal
implementation
Service
Consumer

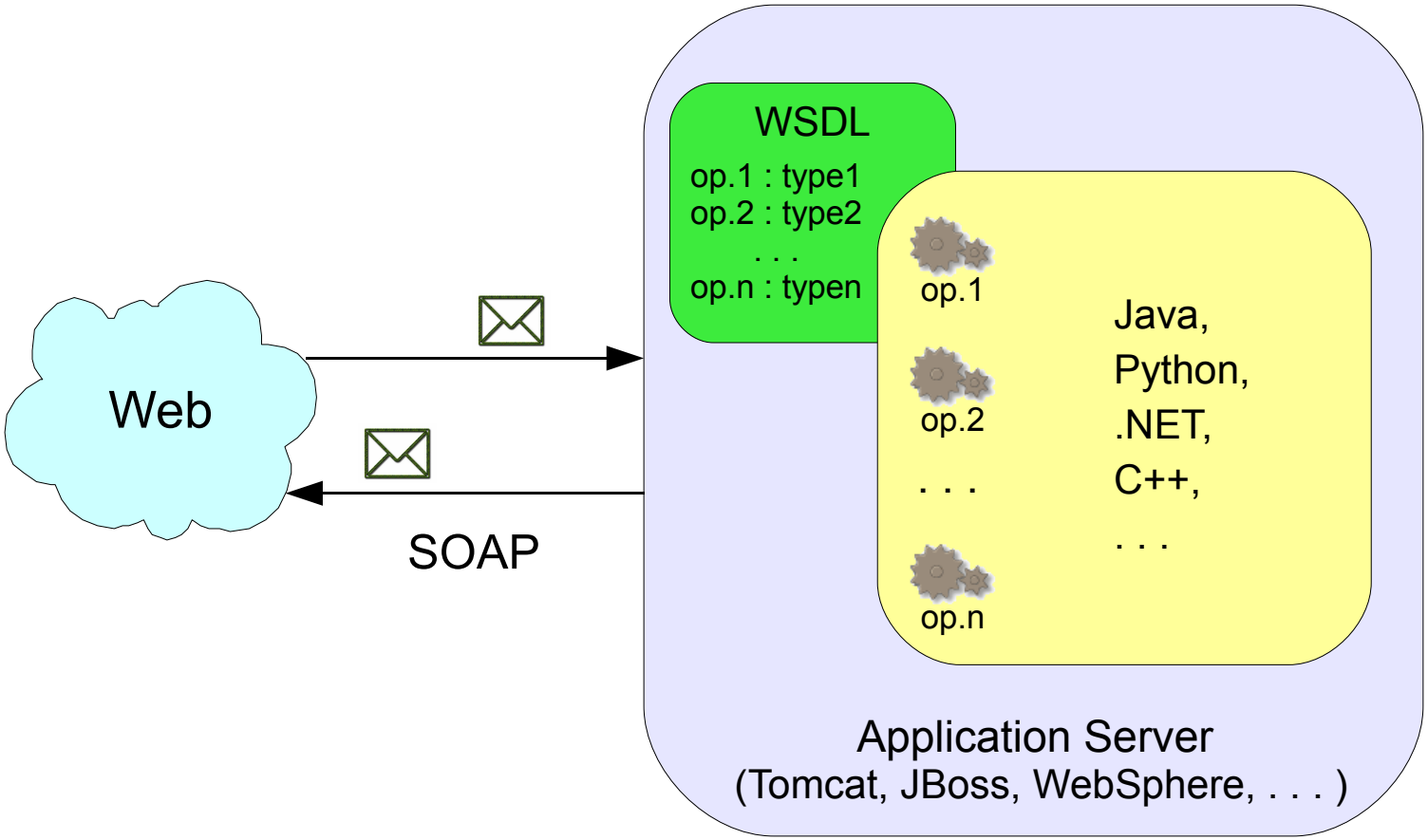
Web Services (WSs)

- Make available the functionalities that a company wants to expose over the Web, so that they can be exploited by other services
- Their underlying architecture is the World Wide Web
  - Widespread and extensively used platform
  - Suitable to connect different companies and customers
- Independently developed applications can be
  - exposed as services
  - interconnected by exploiting the Web infrastructure and the relative standards, e.g. HTTP, XML, SOAP, WSDL and UDDI
- Facilitate automated integration of newly built and legacy applications, both within and across organizational boundaries

Web Service (WS)



Web Service (WS)



Web Services: main features

- Autonomous
- Accesible via Web
- Uniquely identified by an URL
- Platform-independent and language-independent
- Self-contained
  - they can be deployed independently
- Self-describing
  - format of the exchanged messages are defined in their interfaces
- Composable
  - they can be dynamically assembled for developing distributed systems and applications (*business processes*)

Web Services: main features

- Stateless
  - They treat each service request as an independent transaction
  - This facilitates composability
  - How are sessions and transactions among services realized?
    It is up to the messages to guarantee such correlation
    (see *message correlation* in business processes)

Web Services: main features

- Stateless
  - They treat each service request as an independent transaction
  - This facilitates composability
  - How are sessions and transactions among services realized?

    It is up to the messages to guarantee such correlation
    (see *message correlation* in business processes)

Web Services: main features

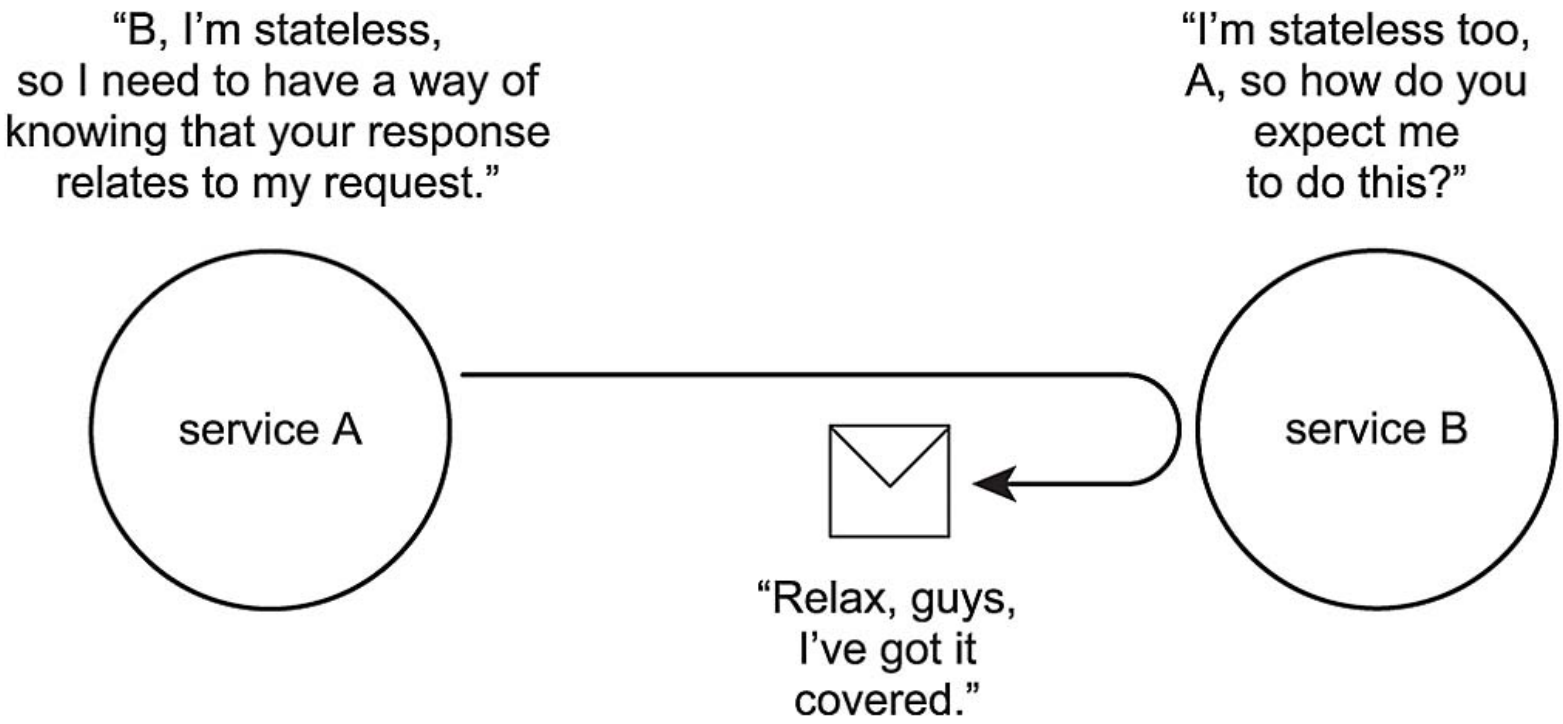
- Stateless
  - They treat each service request as an independent transaction
  - This facilitates composability
  - How are sessions and transactions among services realized?

    It is up to the messages to guarantee such correlation
    (see *message correlation* in business processes)

Web Services: main features

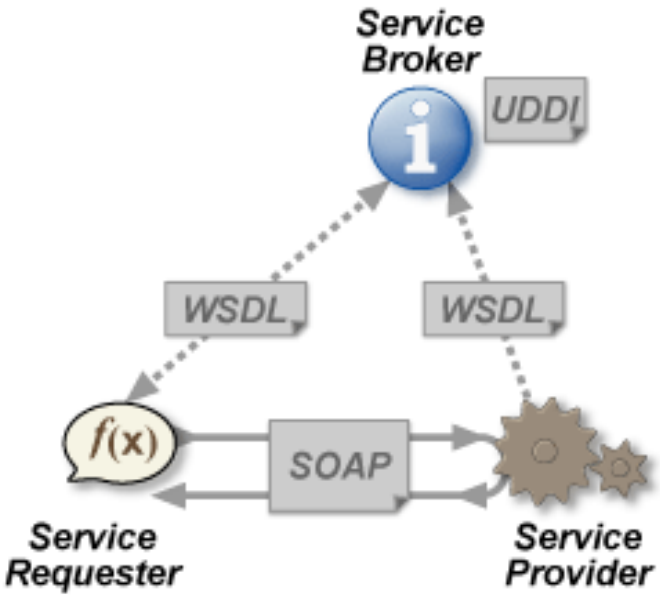
- Stateless
  - They treat each service request as an independent transaction
  - This facilitates composability
  - How are sessions and transactions among services realized?

    It is up to the messages to guarantee such correlation
    (see *message correlation* in business processes)



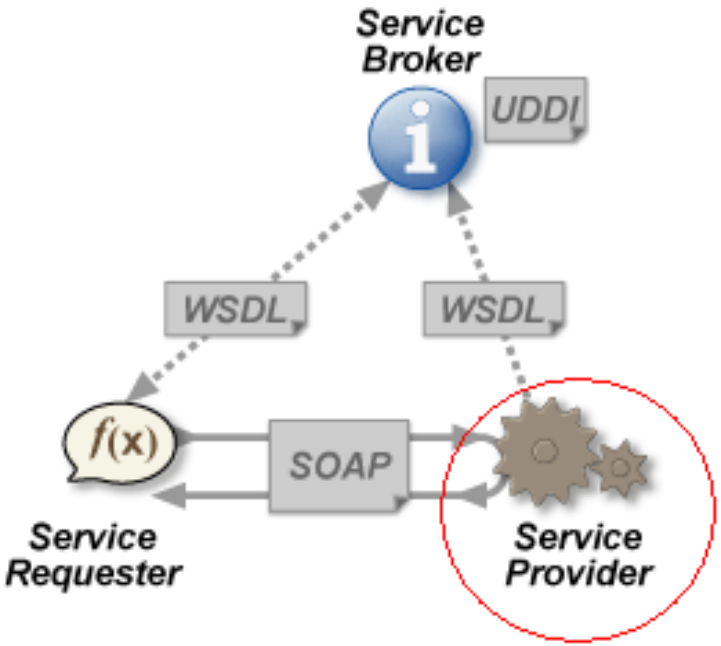
Web Services: main features

- WSs can play three roles in a service-oriented architecture



Web Services: main features

- WSs can play three roles in a service-oriented architecture

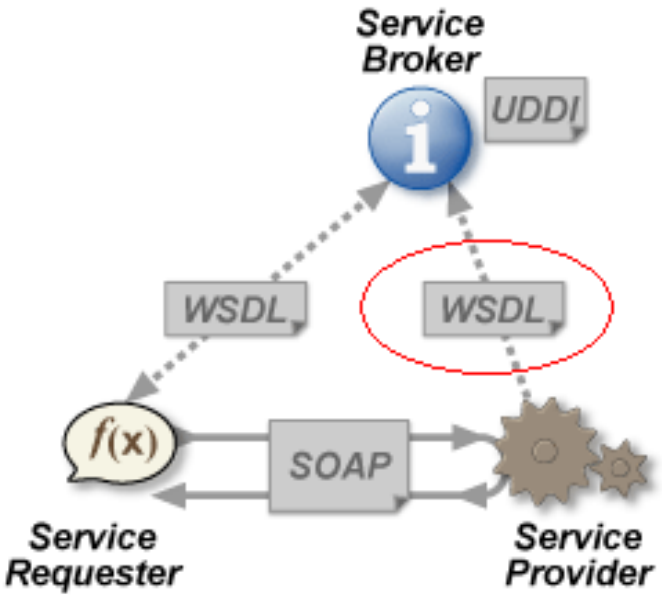


Providers

- *offer* functionalities
- *publish* machine-readable service descriptions on broker registries

Web Services: main features

- WSs can play three roles in a service-oriented architecture



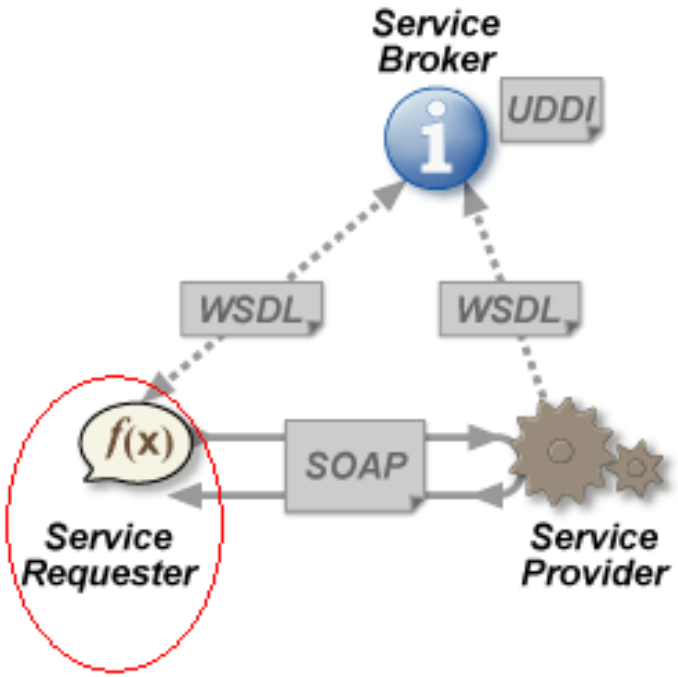
Providers

- *offer* functionalities
- *publish* machine-readable service descriptions on broker registries

Service descriptions should include both functional and non-functional aspects (*Quality of Service*: response time, availability, reliability, security, performance, etc.)

Web Services: main features

- WSs can play three roles in a service-oriented architecture

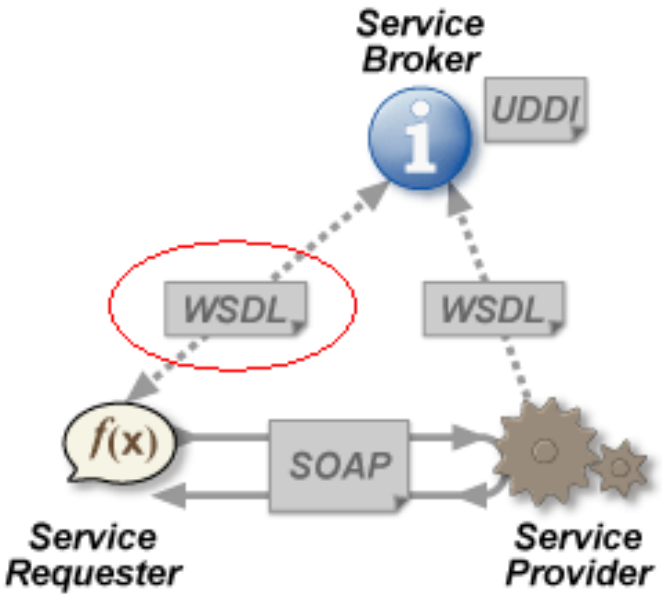


Requesters/Consumers

- *discover* functionalities
- *invoke* providers

Web Services: main features

- WSs can play three roles in a service-oriented architecture

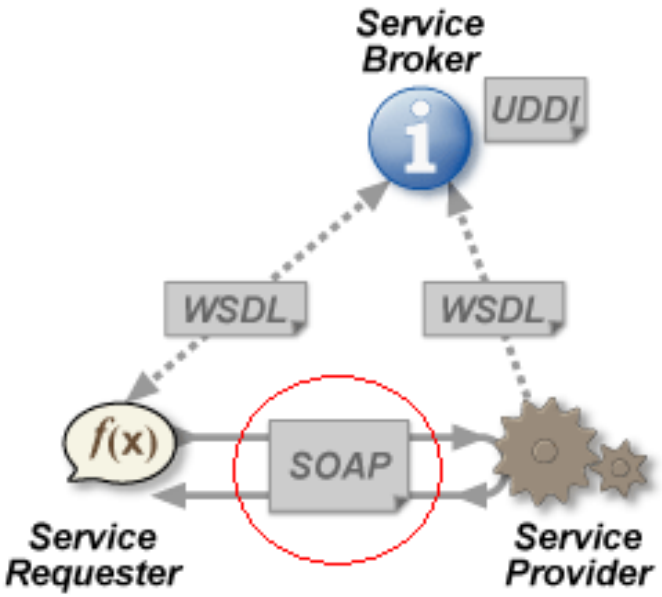


Requesters/Consumers

- *discover* functionalities
- *invoke* providers

Web Services: main features

- WSs can play three roles in a service-oriented architecture

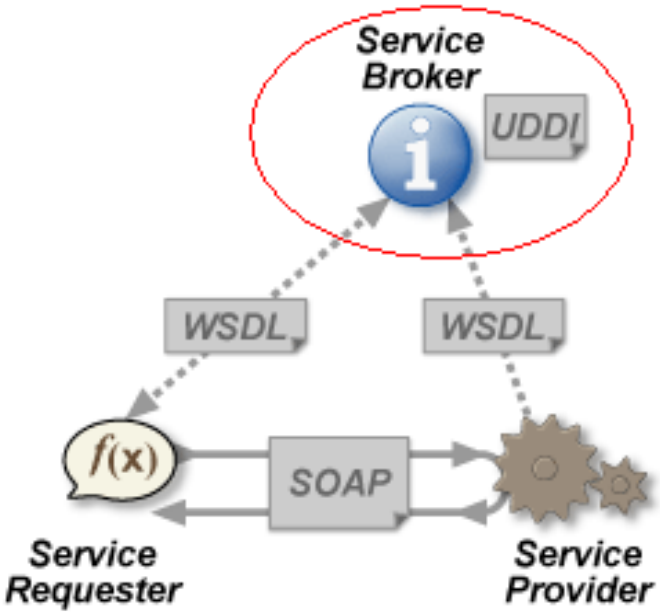


Requesters/Consumers

- *discover* functionalities
- *invoke* providers

Web Services: main features

- WSs can play three roles in a service-oriented architecture

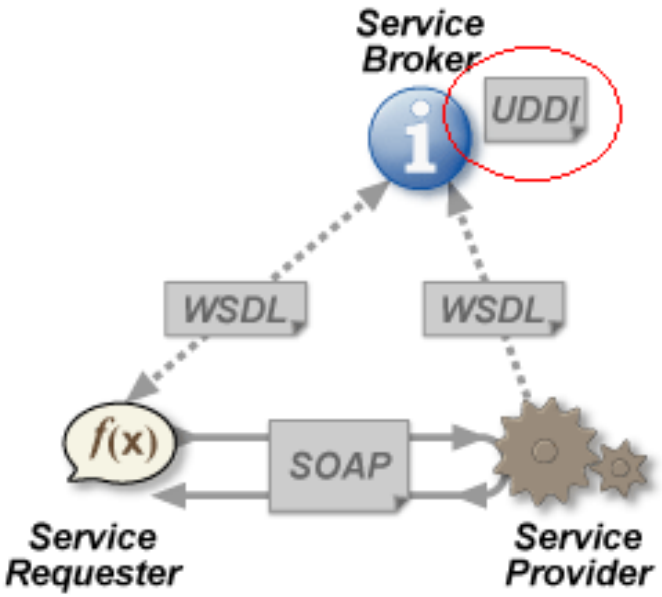


Brokers

- allow *automated* publication and discovery
- rely on *registries*

Web Services: main features

- WSs can play three roles in a service-oriented architecture



Brokers

- allow *automated* publication and discovery
- rely on *registries*

Web Services: advantages

- Interoperability
  - WSs allow different applications running on different platforms to interact in a loosely coupled way
- Reusability
  - WSs are component that can be (re)used in different systems and domains
- Standardization
  - WSs rely on open, standard protocols

Web Services: disadvantages

- Performances
  - They can be lower than other approaches for distributed computing, due to the use of XML
- Security issues
  - The use of HTTP allows WSs to avoid security measures such as firewalls
- Critical systems
  - Currently, there are no mature standards for relevant aspects of critical applications, e.g. distributed transactions

WSDL

What is WSDL?

- WSDL stands for **W**eb **S**ervices **D**escription **L**anguage
- WSDL is an XML document
- WSDL is used to describe the public interface of a Web service
- WSDL is used to define the location of a Web service
- Version 1.1 is the W3C standard most widely used
- Version 2.0 is a W3C Recommendation, but it is not widely adopted yet

WSDL documents

- A WSDL document defines the features of a Web service by means of:
  - `portType`, describes the operation provided by the service
  - `message`, describes the messages exchanged by the service
  - `types`, defines the data types used by the service
  - `binding`, defines the communication protocol for a portType

- Notation:
  - `?` indicates that an element/attribute can be omitted
  - `+` indicates that an element can be present more than once, but cannot be omitted
  - `*` indicates that an element can be present more than once or omitted

WSDL document structure

<definitions>
  <types> ?
    data type definitions ...
  </types>
  <message> *
    message format definitions ...
  </message>
  <portType> *
    definition of the interface (i.e. set of operations) ...
  </portType>
  <binding> *
    protocol and data format specifications ...
  </binding>
</definitions>

< portType > element

```
<definitions>
  ...
  <portType>
    definition of the interface (i.e. set of operations) ...
  </portType>
  ...
</definitions>
```

- The < portType > element is the most relevant WSDL element and describes
  - a *Web service*
  - the *operations* that can be performed
  - the *messages* that are involved
- < portType > can be compared to a function library (a module, a class,. . .) in a traditional programming language

< message > element

```
<definitions>
  ...
  <message>
    message format definitions ...
  </message>
  ...
</definitions>
```

- The < message > element defines the data format of an operation
- Each message can consist of one or more parts
  - Message parts can be compared to the parameters of a function in a traditional programming language

< types > element

```
<definitions>
  ...
  <types>
    data type definitions ...
  </types>
  ...
</definitions>
```

- The < types > element defines the data types udsed by the Web service
- WSDL uses XML Schema to define data types
  - this implies the maximum platform independence

The < binding > element

```
<definitions>
  ...
  <binding>
    binding definitions...
  </binding>
  ...
</definitions>
```

- The < binding > element defines the format of the messages and the details of the protocol used by each WSDL port

Example

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
 <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
 </operation>
</portType>
```

Example

- W.r.t. traditional programming languages:
  - “glossaryTerms” is a function library
  - “getTerm” is a function
  - “getTermRequest” is the input parameter of the function
  - “getTermResponse” is the return parameter

Operation types

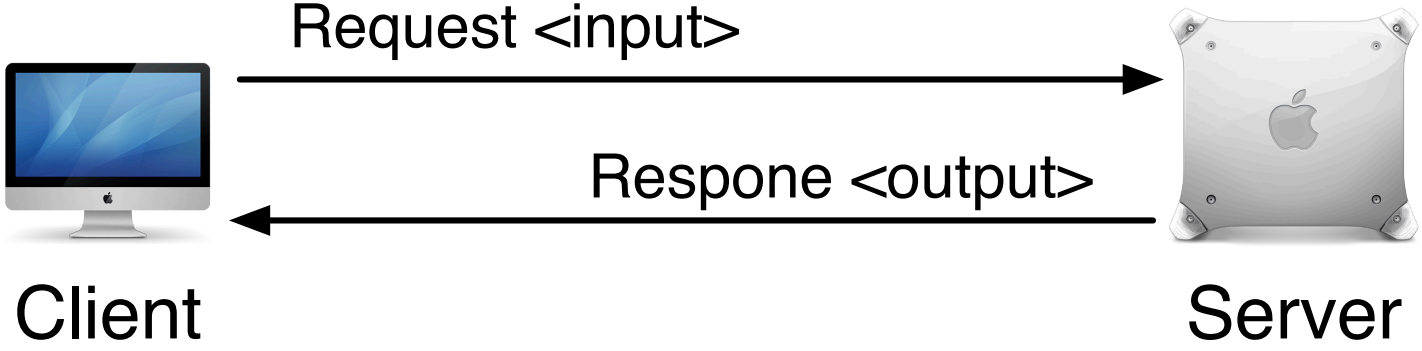
- The most common operation type is *request-response* (see previous example)

- WSDL defines 4 types of operation:

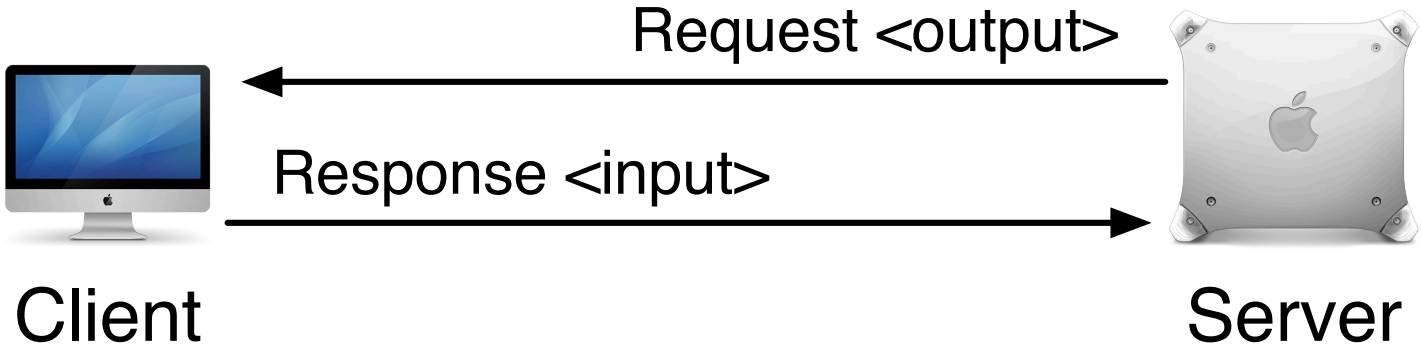
Types	**Definitions**
Request-response	The operation receives a request and will return a response
Solicit-response	The operation sends a message and waits for a response
One-way	The operation receives a message but will not return a response
Notification	The operation sends a message but will not wait for a response

Operation types

Request-Response

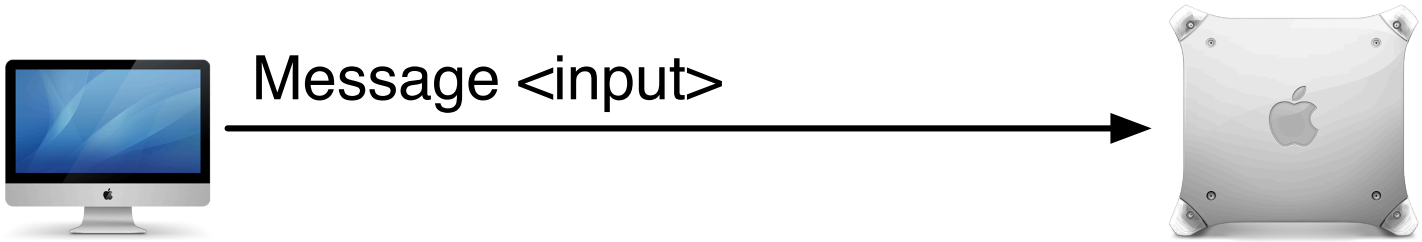


Solicit-Response

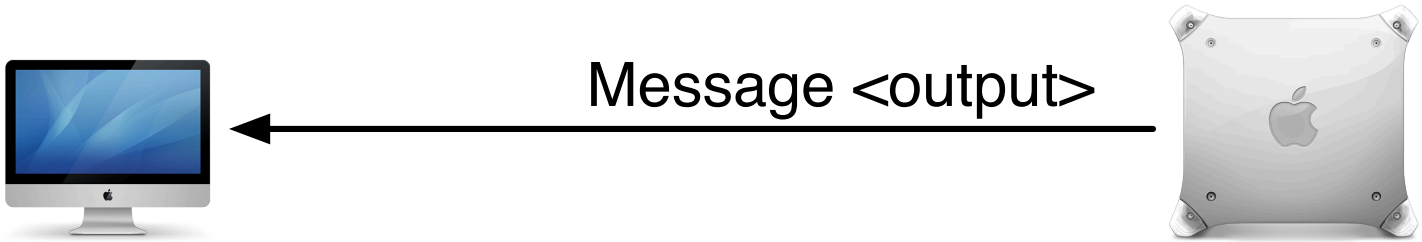


Operation types

One-way



Notification



Operation types: syntax

< portType > syntax

```
<portType name="ncname">
  <operation name="ncname" .... /> *
</portType>
```

Syntax of a *request-response* operation

```
<operation name="n_name">
  <input name="n_name"? message="m_name"/>
  <output name="n_name"? message="m_name"/>
  <fault name="n_name" message="m_name"/> *
</operation>
```

Operation types: syntax

Syntax of a *solicit-response* operation

```
<operation name="n_name">
  <output name="n_name"? message="m_name"/>
  <input name="n_name"? message="m_name"/>
  <fault name="n_name" message="m_name"/> *
</operation>
```

Syntax of a *one-way* operation

```
<operation name="n_name">
  <input name="n_name"? message="m_name"/>
</operation>
```

Operation types: syntax

Syntax of a *notification* operation

```
<operation name="n_name">
  <output name="n_name"? message="m_name"/>
</operation>
```

- Request-response and solicit-response operations can specify zero or more elements `< fault.../ >`
  - they indicate the format of possible *error messages* sent back as operation result
- Interaction modality:
  - *synchronous*: the client is blocked
    - ★ it is defined by a request-response operation
      (the response should be close to the request)
  - *asynchronous*: the client can perform other activities while waiting
    - ★ it is defined by a pair of one-way operations:
      - request op. is provided by server and invoked by client
      - response op. (*callback*) is provided by client and invoked by server

One-way: example

```
<message name="newTermValues">
  <part name="term" type="xs:string"/>
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="setTerm">
    <input name="newTerm" message="newTermValues"/>
  </operation>
</portType >
```

- The “setTerm” operation permits inserting a new term in the glossary
- The input message “newTermValues” is composed of a new “term” and the corresponding “value”

Binding

- We have seen the definition of the *abstract* interface of a Web service
  - it is not bound to a concrete network address (i.e. an URL)
  - it is not bound to any protocol for data transmission
  - it can be used for different implementations of the service
- The `< binding >` element defines the *concrete* part of the service interface

Syntax of the `< binding >` element

```
<binding type="n_type" name="n_name">
```

- `< binding >` has two attributes:
  - *name*: defines the name of the binding
  - *type*: specifies the portType for the binding

Binding to SOAP: example

```
<binding type="glossaryTerms" name="b1">
<soap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http" />
  <operation>
    <soap:operation
     soapAction="http://example.com/getTerm"/>
    <input>
      <soap:body use="literal"/>
    </input>
    <output>
      <soap:body use="literal"/>
    </output>
  </operation>
</binding>
```

Binding to SOAP

- The < `binding` > element contains:

```
<soap:binding style="document"
              transport="http://schemas.xmlsoap.org/soap/http" />
```

  - *style* can be either “document” (the messages contain documents) or “rpc” (the messages contain parameters and return values);
  - *transport*: specifies the transport protocol used by SOAP

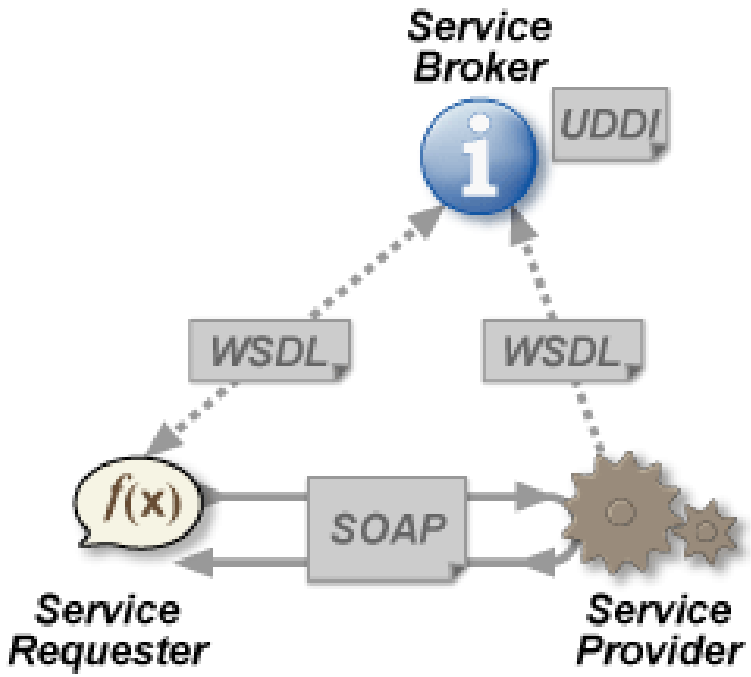
- The < `soap` : `binding` > element is followed by the binding definitions of the operation provided by the port:
  - it must be specified how the input and output are encoded:
    “literal” (no encoding) or
    “encoded” (the encoding is specified by the *encodingStyle* attribute)

WSDL vs. Contracts

- WSDL can be thought of as a simple notion of contract between the provider and the clients of a Web service
- *Problem:* it is a notion of contract too poor of information
  - e.g., a Web service for goods delivery could be reply to an order request after 20 yerars!
- The following aspects should be taken into account:
  - quality of service (QoS)
  - time
  - execution order of the operations
  - ...

WSDL and UDDI

- UDDI: Universal Description, Discovery and Integration
  - it relies on a directory (*registry*) that stores information about Web services
    - ★ basically, such information consists of WSDL descriptions
  - it is based on the SOAP protocol
  - it is like a telephone book
  - the search of Web services is mostly performed manually



SOAP

What is SOAP?

- SOAP stands for **S**imple **O**bject **A**ccess **P**rotocol
- SOAP is a protocol for the communication between applications
- SOAP defines the format for sending messages
- SOAP communicates via Internet
  - in particular, to access Web services
- SOAP is platform independent
- SOAP is language independent
- SOAP is based on XML
- SOAP is extensible
- SOAP is a W3C standard (v1.1 and v1.2)

SOAP elements

- A SOAP message is an XML document containing the following elements:
  - an *Envelope*, that identifies the XML document as a SOAP message
  - an *Header* (optional), that contains supporting information
    - ⋆ e.g., info for routing, security, long running transactions, . . .
  - a *Body*, that contains call and response information
  - a *Fault* (optional), that provides information (a code and an explanation) about possible errors occurred during the message processing

Skeleton of a SOAP message

```
<?xml version="1.0"?>
<soap:Envelope
 xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
 soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
   <soap:Header>
       ...
   </soap:Header>
   <soap:Body>
       ...
       <soap:Fault>
           ...
       </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

SOAP over HTTP: example

Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: ...

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

SOAP over HTTP: example

Response

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: ...

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
  <soap:Body xmlns:m="http://www.example.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

Demo...

References

Some references

- http://www.w3.org/
- http://www.w3.org/TR/xml/
- http://www.w3schools.com/
- http://www.w3schools.com/xml/
- http://www.w3schools.com/webservices/
- http://www.w3schools.com/wsdl/
- http://www.w3schools.com/soap/
- ...