Introduction to Web Services

Luis Rodero-Merino\textsuperscript{1} Agustín Santos Mendéz\textsuperscript{1}

\textsuperscript{1}Laboratorio de Algoritmia Distribuida y Redes, Departamento de Sistemas Telemáticos y Computación Universidad Rey Juan Carlos
Web Services (WS) are a kind of Distributed System (DS) that allows accessing to remote functionality through the web.

Definition (Web Services)

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

Created and maintained by the W3C, about 16 recommendations related (SOAP, WSDL, WS-Addressing, WS-Policy).
**Definition**

- Web Services (WS) are a kind of Distributed System (DS) that allows accessing to remote functionality through the web.

**Definition (Web Services)**

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

- Created and maintained by the W3C, about 16 recommendations related (SOAP, WSDL, WS-Addressing, WS-Policy).
Web Services (WS) are a kind of Distributed System (DS) that allows accessing to remote functionality *through the web*.

**Definition (Web Services)**

Web services provide a standard means of interoperating between different software applications, running on a variety of platforms and/or frameworks.

- Created and maintained by the W3C, about 16 recommendations related (SOAP, WSDL, WS-Addressing, WS-Policy).
Basic Characteristics of WS

- A WS is an application component.
- Data representation in the XML format.
- Message transport through Web protocols (HTTP).
- Resource identification by Uniform Resource Identifiers (URI).
- Although not specifically stated, messages are typically assumed to be stateless.
Basic Characteristics of WS

- A WS is an application component.
- Data representation in the XML format.
- Message transport through Web protocols (HTTP).
- Resource identification by Uniform Resource Identifiers (URI).
- Although not specifically stated, messages are typically assumed to be stateless.
Basic Characteristics of WS

- A WS is an application component.
- Data representation in the **XML** format.
- Message transport through Web protocols (**HTTP**).
- Resource identification by Uniform Resource Identifiers (**URI**).
- Although not specifically stated, messages are typically assumed to be stateless.
Basic Characteristics of WS

- A WS is an application component.
- Data representation in the **XML** format.
- Message transport through Web protocols (**HTTP**).
- Resource identification by Uniform Resource Identifiers (**URI**).

Although not specifically stated, messages are typically assumed to be stateless.
A WS is an application component.
Data representation in the XML format.
Message transport through Web protocols (HTTP).
Resource identification by Uniform Resource Identifiers URI.
Although not specifically stated, messages are typically assumed to be stateless.
The WS Working Group at the W3C defines WS as:

Definition (Web Service)
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.
The WS Working Group at the W3C defines WS as:

**Definition (Web Service)**

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.
A WSDL document defines the WS interface.

Calls to services and their results are sent by messages built using the SOAP protocol.

Messages are transported through HTTP, SMTP...

Extensions to basic or core WS:

- There are a lots of specifications that extend basic WS to define how to perform transactions (WS-Transaction), tasks coordination (WS-Coordination), etc. The whole set is often denoted as WS-*.
- Some are from organizations such as W3C or OASIS, others from private companies (BEA, IBM...).
WS Components

- A WSDL document defines the WS interface.
- Calls to services and their results are sent by messages built using the SOAP protocol.
- Messages are transported through HTTP, SMTP...
- Extensions to basic or core WS:
  - There are a lot of specifications that extend basic WS to define how to perform transactions (WS-Transaction), tasks coordination (WS-Coordination), etc. The whole set is often denoted as WS-*.
  - Some are from organizations such as W3C or OASIS, others from private companies (BEA, IBM...).
A WSDL document defines the WS interface.

Calls to services and their results are sent by messages built using the SOAP protocol.

Messages are transported through HTTP, SMTP...

Extensions to basic or core WS:
- There are a lots of specifications that extend basic WS to define how to perform transactions (WS-Transaction), tasks coordination (WS-Coordination), etc. The whole set is often denoted as WS-*.
- Some are from organizations such as W3C or OASIS, others from private companies (BEA, IBM...).
WS Components

- A WSDL document defines the WS interface.
- Calls to services and their results are sent by messages built using the SOAP protocol.
- Messages are transported through HTTP, SMTP...
- Extensions to basic or core WS:
  - There are a lots of specifications that extend basic WS to define how to perform transactions (WS-Transaction), tasks coordination (WS-Coordination), etc. The whole set is often denoted as WS-*.
  - Some are from organizations such as W3C or OASIS, others from private companies (BEA, IBM...).
WS Components

- A WSDL document defines the WS interface.
- Calls to services and their results are sent by messages built using the SOAP protocol.
- Messages are transported through HTTP, SMTP...
- Extensions to basic or core WS:
  - There are a lots of specifications that extend basic WS to define how to perform transactions (WS-Transaction), tasks coordination (WS-Coordination), etc. The whole set is often denoted as WS-*.
  - Some are from organizations such as W3C or OASIS, others from private companies (BEA, IBM...).
New terms have appeared related with the creation of DS.

There is a lot of confusion around.

For example, some defend that WS is not only WSDL + SOAP + HTTP.

SOAP does not stand for SOA Protocol.

What is SOA?, what is REST?.

How do these terms relate with WS?.

New terms have appeared related with the creation of DS.

There is a lot of confusion around.

For example, some defend that WS is not only WSDL + SOAP + HTTP.

SOAP does not stand for SOA Protocol.

What is SOA?, what is REST?.

How do these terms relate with WS?.
New terms have appeared related with the creation of DS.
There is a lot of confusion around.
For example, some defend that WS is not only **WSDL + SOAP + HTTP**.

**SOAP** does not stand for **SOA Protocol**.
What is SOA?, what is REST?.
How do these terms relate with WS?
New terms have appeared related with the creation of DS.
There is a lot of confusion around.
For example, some defend that WS is not only WSDL + SOAP + HTTP.
SOAP does not stand for SOA Protocol.
What is SOA?, what is REST?.
How do these terms relate with WS?
New terms have appeared related with the creation of DS.
There is *a lot* of confusion around.
For example, some defend that WS is not only *WSDL + SOAP + HTTP*.
*SOAP* does not stand for *SOA Protocol*.
What is SOA?, what is REST?.
How do these terms relate with WS?
New terms have appeared related with the creation of DS.
There is a lot of confusion around.
For example, some defend that WS is not only WSDL + SOAP + HTTP.
SOAP does not stand for SOA Protocol.
What is SOA?, what is REST?.
How do these terms relate with WS?. 
SOA - Service Oriented Architecture

Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding. Services interact based on what functions they provide and how they deliver them.
- OO and procedural systems link elements together based on type or name.
SOA - Service Oriented Architecture

Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
Many DS, over all those that cross different administration domains, aim to achieve:

- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:

- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
Many DS, over all those that cross different administration domains, aim to achieve:
- Low-coupling.
- Interoperability.
- Platform neutrality.

The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:
- Rely on shared schemas, not shared code (interfaces and implementations are split).
- SOA differs from OO and RPC in one key aspect: binding.
  - Services interact based on what functions they provide and how they deliver them.
  - OO and procedural systems link elements together based on type or name.
SOA - Service Oriented Architecture

- Many DS, over all those that cross different administration domains, aim to achieve:
  - Low-coupling.
  - Interoperability.
  - Platform neutrality.
- The Service Oriented Architecture (SOA) is a proposal of a DS architecture that:
  - Rely on shared schemas, not shared code (interfaces and implementations are split).
  - SOA differs from OO and RPC in one key aspect: *binding*.
    - Services interact based on what functions they provide and how they deliver them.
    - OO and procedural systems link elements together based on type or name.
In SOA, functionality is exposed as *services*, where:

- Each *service* is an abstracted view of programs, DBs, etc.
- Each *service* interface is defined through the *messages* exchanged, that will follow a *standard format*.
- Each *service* is defined in a machine-processable format.

SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).

Web Services fit well the SOA requirements.
In SOA, functionality is exposed as *services*, where:

- Each *service* is an abstracted view of programs, DBs, etc.
- Each *service* interface is defined through the *messages* exchanged, that will follow a *standard format*.
- Each *service* is defined in a machine-processable format.

SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).

Web Services fit well the SOA requirements.
In SOA, functionality is exposed as *services*, where:
- Each *service* is an abstracted view of programs, DBs, etc.
- Each *service* interface is defined through the *messages* exchanged, that will follow a *standard format*.
- Each *service* is defined in a machine-processable format.

- SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).
- Web Services fit well the SOA requirements.
WS and SOA

- In SOA, functionality is exposed as services, where:
  - Each service is an abstracted view of programs, DBs, etc.
  - Each service interface is defined through the messages exchanged, that will follow a standard format.
  - Each service is defined in a machine-processable format.

- SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).

- Web Services fit well the SOA requirements.
In SOA, functionality is exposed as services, where:
- Each service is an abstracted view of programs, DBs, etc.
- Each service interface is defined through the messages exchanged, that will follow a standard format.
- Each service is defined in a machine-processable format.

SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).

Web Services fit well the SOA requirements.
In SOA, functionality is exposed as services, where:

- Each service is an abstracted view of programs, DBs, etc.
- Each service interface is defined through the messages exchanged, that will follow a standard format.
- Each service is defined in a machine-processable format.

SOA pursues fault tolerance through asynchrony and communication models that support partial failures (transactions).

Web Services fit well the SOA requirements.
REST - Representation State Transfer

- An ‘arquitectural style’, a proposal of how to offer remote funcionality through the web.
- Main characteristics:
  - Everything is identified by an URI:
    - http://example.com/customers/1234
  - Only four standard methods can be used: GET(retrieve), PUT(update), POST(create) and DELETE(remove).
    - => GET /costumers/1234
    - => Host: example.com
    - => Accept: application/mycompany.customer+xml
    - ...
    - <= <costumer>...</costumer>
  - Stateless communication.
REST - Representation State Transfer

- An ‘arquitectural style’, a proposal of how to offer remote funcionality through the web.
- Main characteristics:
  - Everything is identified by an URI:
    http://example.com/customers/1234
    http://example.com/orders/2007/10/776654
  - Only four standard methods can be used: GET(retrieve), PUT(update), POST(create) and DELETE(remove).
    => GET /costumers/1234
    => Host: example.com
    => Accept: application/mycompany.customer+xml
    ...
    <= <costumer>...</costumer>

- Stateless communication.
REST - Representation State Transfer

- An ‘architectural style’, a proposal of how to offer remote functionality through the web.

**Main characteristics:**

- Everything is identified by an URI:
  
  http://example.com/customers/1234  
  http://example.com/orders/2007/10/776654

- Only four standard methods can be used: **GET**(retrieve), **PUT**(update), **POST**(create) and **DELETE**(remove).

  => GET /customers/1234

  => Host: example.com

  => Accept: application/mycompany.customer+xml

  ...

  <= <customer>...</costumer>

- Stateless communication.
REST - Representation State Transfer

- An ‘architectural style’, a proposal of how to offer remote functionality through the web.
- Main characteristics:
  - Everything is identified by an URI:
    
    http://example.com/customers/1234
    http://example.com/orders/2007/10/776654
  
  - Only four standard methods can be used: GET(retrieve), PUT(update), POST(create) and DELETE(remove).
  
    => GET /customers/1234
    => Host: example.com
    => Accept: application/mycompany.customer+xml
    ...
    <= <customer>...</costumer>

- Stateless communication.
REST are WS?

REST supporters advocate for what they call ‘REST Web Services’.

In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

SOA and REST are opposing proposals?

Some assume that SOA and SOAP + WS-* are equivalent (although they are not).

Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.

However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
REST are WS?
- REST supporters advocate for what they call ‘REST Web Services’.
- In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

SOA and REST are opposing proposals?
- Some assume that SOA and SOAP + WS-* are equivalent (although they are not).
- Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.
- However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
REST are WS?
- REST supporters advocate for what they call ‘REST Web Services’.
- In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

SOA and REST are opposing proposals?
- Some assume that SOA and SOAP + WS-* are equivalent (although they are not).
- Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.
- However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
Terms are Messed Up

REST are WS?
- REST supporters advocate for what they call ‘REST Web Services’.
- In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

SOA and REST are opposing proposals?
- Some assume that SOA and SOAP + WS-* are equivalent (although they are not).
  - Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.
  - However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
Terms are Messed Up

- **REST are WS?**
  - REST supporters advocate for what they call ‘REST Web Services’.
  - In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

- **SOA and REST are opposing proposals?**
  - Some assume that SOA and SOAP + WS-* are equivalent (although they are not).
  - Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.
  - However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
Terms are Messed Up

- **REST are WS?**
  - REST supporters advocate for what they call ‘REST Web Services’.
  - In contrast, some associate SOAP + WS-* standards as the only technologies that offer what can be called WS.

- **SOA and REST are opposing proposals?**
  - Some assume that SOA and SOAP + WS-* are equivalent (although they are not).
  - Often, SOAP + WS-* (and so SOA) are presented as a competing proposal to REST.
  - However, the W3C identifies REST as a valid model to build WS, that can be implemented also by using SOAP and WSDL.
REST applies the Web principles to develop distributed services. It advocates for simplicity.

SOAP + WS-* standards are seen by many as a more complete and flexible solution.
REST and SOAP+WS-* Debate

- REST applies the Web principles to develop distributed services. It advocates for simplicity.

- SOAP + WS-* standards are seen by many as a more complete and flexible solution.
REST and SOAP+WS-* Debate

- REST applies the Web principles to develop distributed services. It advocates for simplicity.

- SOAP + WS-* standards are seen by many as a more complete and flexible solution.
REST and SOAP+WS-* Debate

- REST applies the Web principles to develop distributed services. It advocates for simplicity.

- SOAP + WS-* standards are seen by many as a more complete and flexible solution.
**Finding WS: UDDI**

- **UDDI** stands for Universal Description, Discovery and Integration.
  - UDDI is a directory for storing information (metadata) about web services.
  - UDDI is a directory of web service interfaces described by WSDL.
  - UDDI communicates via SOAP (it is a WS itself, with its own WSDL).
Finding WS: UDDI

- **UDDI** stands for Universal Description, Discovery and Integration.
- UDDI is a directory for storing information (metadata) about web services.
  - UDDI is a directory of web service interfaces described by WSDL.
  - UDDI communicates via SOAP (it is a WS itself, with its own WSDL).
Finding WS: UDDI

- **UDDI** stands for Universal Description, Discovery and Integration.
- UDDI is a directory for storing information (metadata) about web services.
- UDDI is a directory of web service interfaces described by WSDL.
- UDDI communicates via SOAP (it is a WS itself, with its own WSDL).
Finding WS: UDDI

- **UDDI** stands for Universal Description, Discovery and Integration.
- UDDI is a directory for storing information (metadata) about web services.
- UDDI is a directory of web service interfaces described by WSDL.
- UDDI communicates via SOAP (it is a WS itself, with its own WSDL).
Calling a WS

UDDI
Service
Register

Web
Service
Client

Web
Service
Provider
Calling a WS

1. Publish Service by UDDI

UDDI
Service Register

Web Service Provider

Web Service Client
Calling a WS

1. Publish Service by UDDI

2. Query for Service by UDDI

UDDI Service Register

Web Service Provider

Web Service Client
Calling a WS

1. Publish Service by UDDI
2. Query for Service by UDDI
3. Get the service WSDL
Calling a WS

1. Publish Service by UDDI
2. Query for Service by UDDI
3. Get the service WSDL
4. Call the Service
Calling a WS

1. WS Client

TCP PIPE

2. WS Server

Middleware - Introduction to Web Services
Calling a WS

Middleware - Introduction to Web Services
Calling a WS

HTTP Message

Head
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

Body
<SOAP-ENV:Envelope>...
</SOAP-ENV:Envelope>

SOAP request

WS Client

HTTP Message

Head
HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

Body
<SOAP-ENV:Envelope>...
</SOAP-ENV:Envelope>

SOAP reply

WS Server
WS Reliable Messaging

- Ensures that messages are delivered as wished by sender.
- Given by `WS-ReliableMessaging` protocol.
- Takes care of acknowledgements by sender, and warns when the message could not be sent.
- It is not a ‘TCP for SOAP’. No session is kept between Source and Destination.
- Specially suitable for asynchronous communication.
WS Reliable Messaging

- Ensures that messages are delivered as wished by sender.
- Given by **WS-ReliableMessaging** protocol.
- Takes care of acknowledgements by sender, and warns when the message could not be sent.
- It is not a ‘TCP for SOAP’. No session is kept between Source and Destination.
- Specially suitable for asynchronous communication.
WS Reliable Messaging

- Ensures that messages are delivered as wished by sender.
- Given by **WS-ReliableMessaging** protocol.
- Takes care of acknowledgements by sender, and warns when the message could not be sent.
- It is not a ‘TCP for SOAP’. No session is kept between Source and Destination.
- Specially suitable for asynchronous communication.
WS Reliable Messaging

- Ensures that messages are delivered as wished by sender.
- Given by **WS-ReliableMessaging** protocol.
- Takes care of acknowledgements by sender, and warns when the message could not be sent.
- It is not a ‘TCP for SOAP’. No session is kept between Source and Destination.
- Specially suitable for asynchronous communication.
WS Reliable Messaging

- Ensures that messages are delivered as wished by sender.
- Given by **WS-ReliableMessaging** protocol.
- Takes care of acknowledgements by sender, and warns when the message could not be sent.
- It is not a ‘TCP for SOAP’. No session is kept between Source and Destination.
- Specially suitable for asynchronous communication.
WS Reliable Messaging (2)

- Can specify the *Delivery Assurance*.
  - AtLeastOnce. If there is an error, the Sender is warned.
  - AtMostOnce. Avoids duplicates.
  - ExactlyOnce. Previous assurances combined.
  - InOrder. Messages will be received by the Destination exactly in the same order they were sent by the Source.
WS Reliable Messaging (2)

- Can specify the *Delivery Assurance*.
  - AtLeastOnce. If there is an error, the Sender is warned.
  - AtMostOnce. Avoids duplicates.
  - ExactlyOnce. Previous assurances combined.
  - InOrder. Messages will be received by the Destination exactly in the same order they were sent by the Source.
WS Reliable Messaging (2)

- Can specify the *Delivery Assurance*.
  - AtLeastOnce. If there is an error, the Sender is warned.
  - AtMostOnce. Avoids duplicates.
  - ExactlyOnce. Previous assurances combined.
  - InOrder. Messages will be received by the Destination exactly in the same order they were sent by the Source.
Can specify the *Delivery Assurance*.  
- **AtLeastOnce.** If there is an error, the Sender is warned.  
- **AtMostOnce.** Avoids duplicates.  
- **ExactlyOnce.** Previous assurances combined.  
- **InOrder.** Messages will be received by the Destination exactly in the same order they were sent by the Source.
WS Reliable Messaging (2)

Can specify the *Delivery Assurance*.
- **AtLeastOnce.** If there is an error, the Sender is warned.
- **AtMostOnce.** Avoids duplicates.
- **ExactlyOnce.** Previous assurances combined.
- **InOrder.** Messages will be received by the Destination exactly in the same order they were sent by the Source.
The transport protocol (HTTP) is in charge of delivering the message and sending the reply.

When using HTTP, a POST command is used to send the SOAP request, and the SOAP response is sent by the HTTP reply.

What happens if the requester wants another kind of interaction?

WS-Addressing gives a more powerful way to refer to entities: endpoints references.
The transport protocol (HTTP) is in charge of delivering the message and sending the reply.

When using HTTP, a POST command is used to send the SOAP request, and the SOAP response is sent by the HTTP reply.

What happens if the requester wants another kind of interaction?

WS-Addressing gives a more powerful way to refer to entities: endpoints references.
WS-Addressing

- The transport protocol (HTTP) is in charge of delivering the message and sending the reply.
- When using HTTP, a POST command is used to send the SOAP request, and the SOAP response is sent by the HTTP reply.
- What happens if the requester wants another kind of interaction?
- WS-Addressing gives a more powerful way to refer to entities: endpoints references.
The transport protocol (HTTP) is in charge of delivering the message and sending the reply.

When using HTTP, a POST command is used to send the SOAP request, and the SOAP response is sent by the HTTP reply.

What happens if the requester wants another kind of interaction?

WS-Addressing gives a more powerful way to refer to entities: endpoints references.
WS-Addressing

HTTP Message
POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: nnnn
SOAPAction: ""

Body
<SOAP-ENV:Envelope>...
</SOAP-ENV:Envelope>

SOAP request

WS Client

HTTP Message
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: xxxx

Body
<SOAP-ENV:Envelope>...
</SOAP-ENV:Envelope>

SOAP reply

WS Server
WS-Addressing

HTTP Message

POST /StockQuote HTTP/1.1
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: ""

Body

<SOAP-ENV:Envelope>...
</SOAP-ENV:Envelope>

SOAP request

What if the request reply must be sent to other agent?
What if the request failure must be reported to other agent?

Fault Handler

Reply Handler
WS Resource Framework

- WS are typically stateless.
- However, some services require to have state: data that persist across calls.
- **WS Resource Framework** sets how to refer to stateful resources.
  - Each message carries the ID of the stateful resources to use (as input or output).
  - Notation: WS + stateful resource = WS-Resource.
  - WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
WS are typically stateless.
However, some services require to have state: data that persist across calls.

**WS Resource Framework** sets how to refer to stateful resources.
- Each message carries the ID of the stateful resources to use (as input or output).
- Notation: WS + stateful resource = WS-Resource.
- WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
WS are typically stateless.

However, some services require to have *state*: data that persist across calls.

**WS Resource Framework** sets how to refer to stateful resources.

- Each message carries the ID of the stateful resources to use (as input or output).
- Notation: WS + stateful resource = *WS-Resource*.
- WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
WS are typically stateless.

However, some services require to have state: data that persist across calls.

**WS Resource Framework** sets how to refer to stateful resources.

- Each message carries the ID of the stateful resources to use (as input or output).
- Notation: WS + stateful resource = *WS-Resource*.
- WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
WS are typically stateless.

However, some services require to have state: data that persist across calls.

**WS Resource Framework** sets how to refer to stateful resources.

- Each message carries the ID of the stateful resources to use (as input or output).
- Notation: WS + stateful resource = *WS-Resource*.
- WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
WS Resource Framework

- WS are typically stateless.
- However, some services require to have state: data that persist across calls.
- **WS Resource Framework** sets how to refer to stateful resources.
  - Each message carries the ID of the stateful resources to use (as input or output).
  - Notation: WS + stateful resource = \textit{WS-Resource}.
  - WS-Addressing allows to add the id of the resource inside the WS endpoint reference, so we can address a WS-Resource.
Let’s imagine a WS that keeps a counter. Each time, the client passes a number that is added to the counter, and the WS returns the new value of the counter.
But the WS has no state!. The counter will be reset with each call.
To solve this, a stateful resource is created. The server sends back the id of that resource so it can be used afterwards.
Next calls will be able to reference the resource.
Next calls will be able to reference the resource.

Add 5
5, res id = 3445
Add 7, res is=3445
12, res id = 3445
Add 4, res is=3445
16, res id = 3445

Id 3445
12
16