Karlsruhe Institute of Technology

aifb

CRITICAL INFORMATION INFRASTRUCTURES RESEARCHGROUP

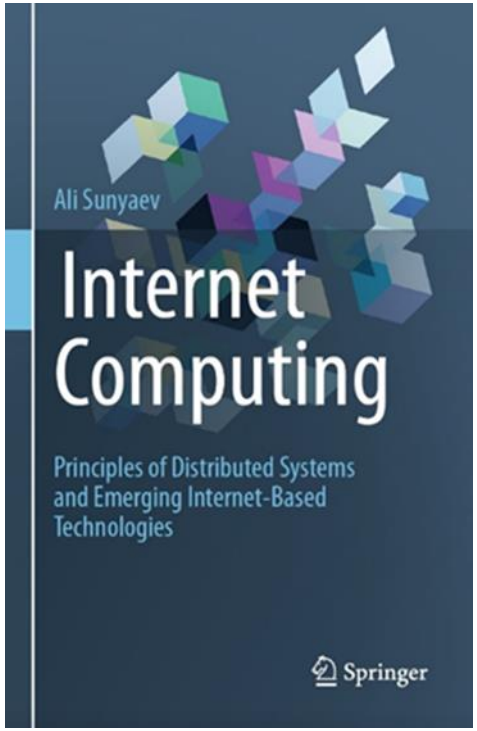
AI: Internet Computing

Lecture 6 — Web Services

Learning Objectives

- Understand the Web service concept from different perspectives
- Purpose of Web services and how they facilitate system integration efforts within and between separate organizations, as well as their technical architecture
- Understand SOAP Web services, RESTful Web services, their differences, and making informed choices when choosing between those two
- Learn about the two most important technologies on which Web services are built:
  - Hypertext Transfer Protocol (HTTP)
  - Extensible Markup Language (XML)

Reference to the Teaching Material Provided



Chapter 6
Web Services

Abstract
Based on the client-server principle, Web services are software systems that interact with client applications and other services through open Web standards. Consequently, heterogenous computer systems from all over the world can exchange information, regardless of their hardware configurations, operating systems, and software applications. Web services are, therefore, a very popular approach for facilitating automated intra-organizational and inter-organizational communication. This chapter provides a thorough introduction to the Web service concept and the different associated standards and technologies, such as Simple Object Access Protocol (SOAP), RESTful interfaces, and Web Services Description Language (WSDL). This chapter particularly provides a comprehensive introduction for two important Web technologies on which most Web services are based: The Hypertext Transfer Protocol (HTTP) and the Extensible Markup Language (XML). Then, the fundamental Web service architectural principles are explained and two common Web service variants are explored in more detail, namely RESTful and SOAP-based Web services. Using these example implementations, this chapter concludes by comparing the two Web service variants in terms of their different application areas.

Learning Objectives of this Chapter

Basics of Web Services

Web Services

Definition

Web services are self-contained, modular, distributed, dynamic applications that can be described, published, located, and invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web based.

Mohamed and Wijesekera, 2012

- Interact with client applications to exchange data
- Are based on the client-server principle
- Act as servers, which provide clients with functionalities
- Make these functions available
- Can be deployed on any network, not just the World Wide Web

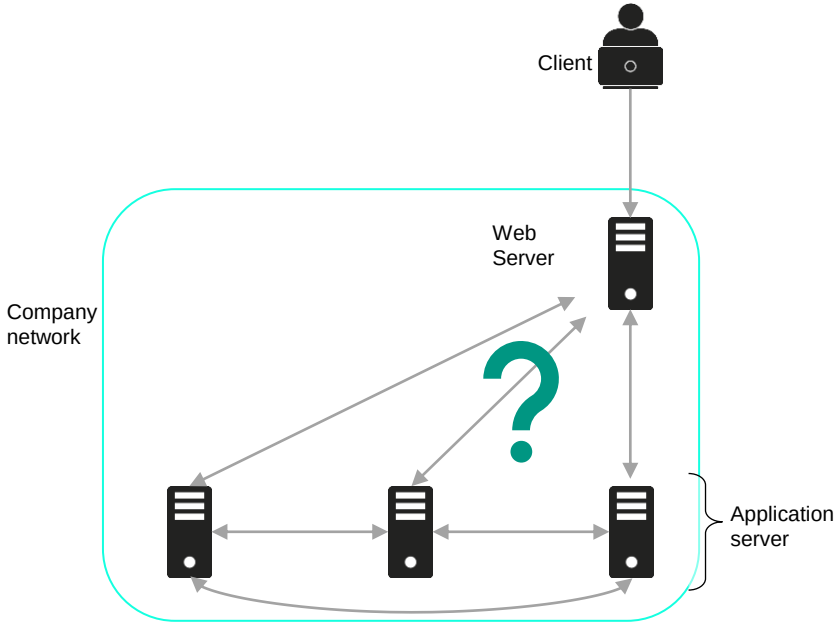
Source: Mohamed K, Wijsekera D (2012) Perfromance analysis of web services on mobile devices. Procedia Comput Sci 10:744-751

Web Services and XML/HTTP

- Web Services apply standardized Web technologies such as XML and HTTP
- Several benefits result from the use of standardized Web technologies:
  - Interoperability and extensibility allow connectivity between software applications
  - Less dependency on networks, operating systems, and platforms
  - Ideal for connecting software applications
  - Coupling of Web services
- Web services are a type of distributed system → service provider and service consumer are physically separated
- Example for a coupled web service:
  - Simple services that are chained together and provide value-added services
  - E-commerce Web service: external payment, warehousing, shipping services

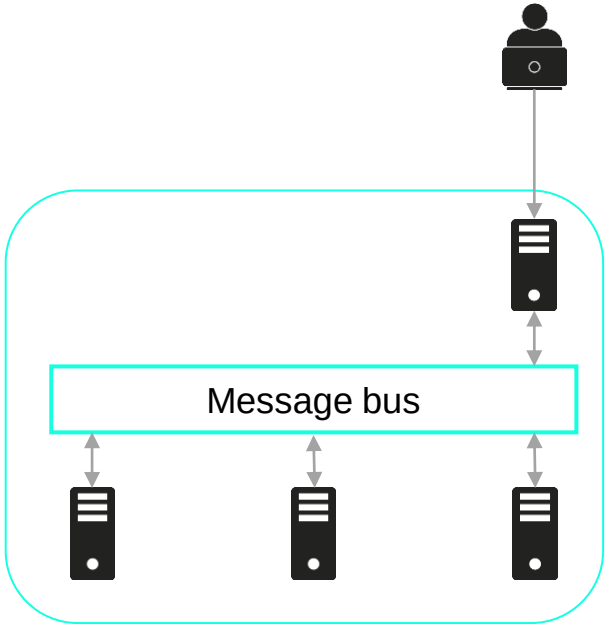
Web Services — Motivation

- Communication in (internal) distributed systems is complex



Web Services — Motivation

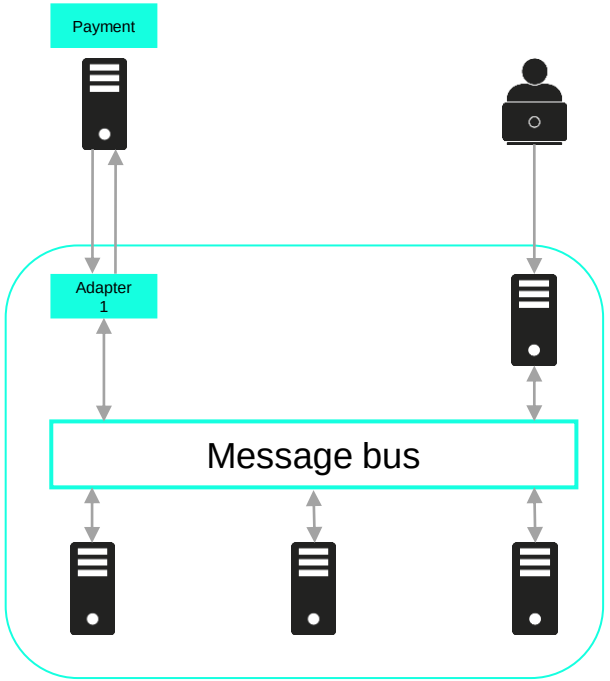
- Communication in (internal) distributed systems is complex



Web Services — Motivation

- Communication in (internal) distributed systems is complex
- What about the integration of external businesses?
- Examples:

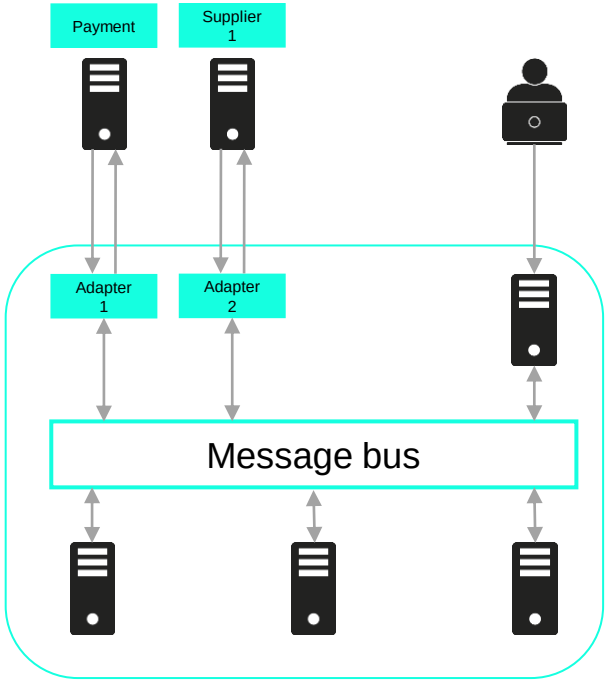
1. External payment service



Web Services — Motivation

- Communication in (internal) distributed systems is complex
- What about the integration of external businesses?
- Examples:

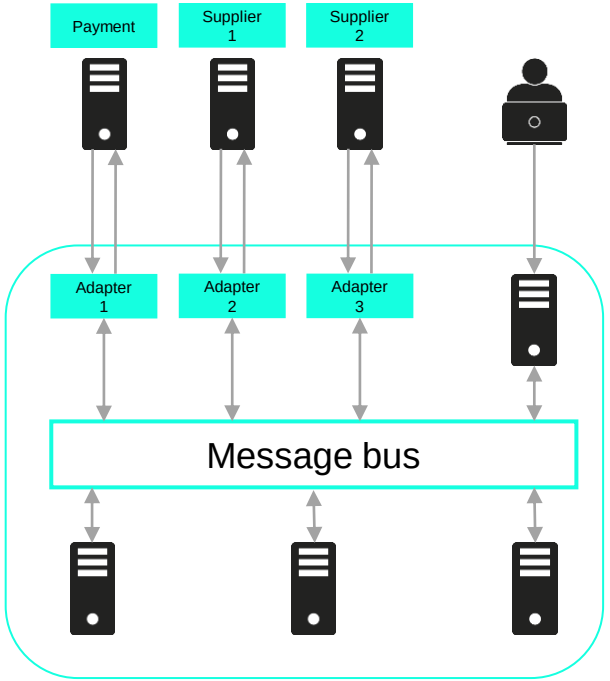
1. External payment service
2. Ordering out-of-stock products from third party suppliers



Web Services — Motivation

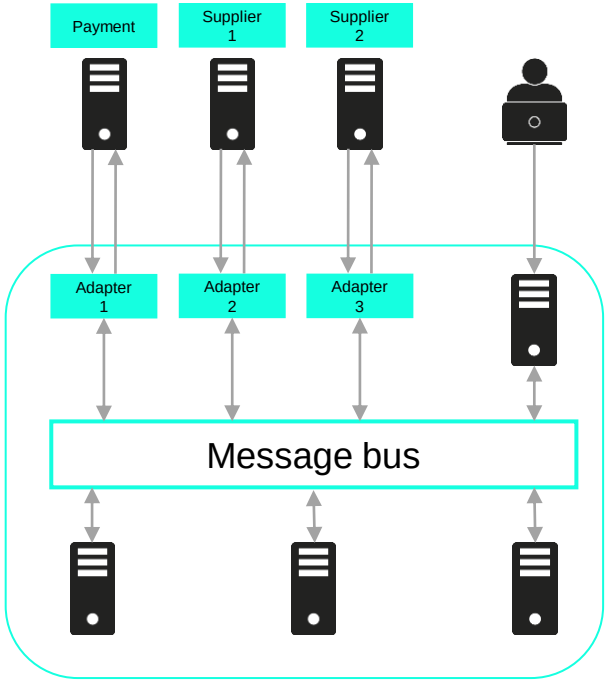
- Communication in (internal) distributed systems is complex
- What about the integration of external businesses?
- Examples:

1. External payment service
2. Ordering out-of-stock products from third party suppliers



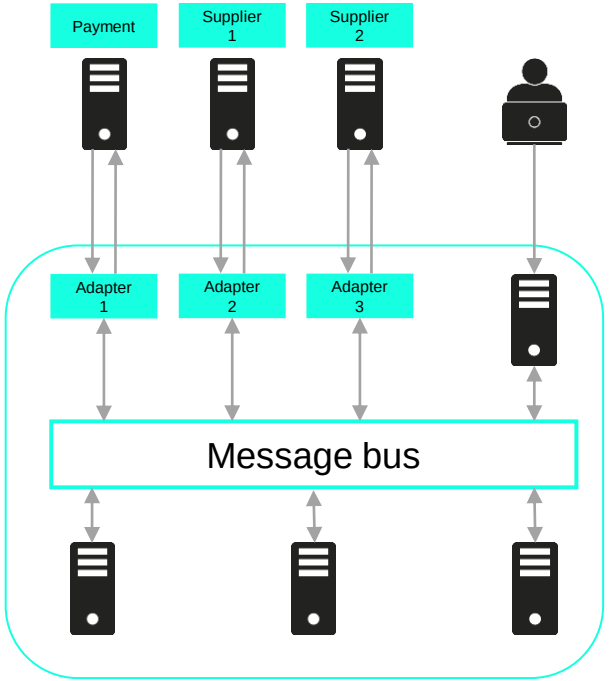
Web Services — Motivation

- Communication in (internal) distributed systems is complex
- What about the integration of external businesses?
- Examples:
1. External payment service
2. Ordering out-of-stock products from third party suppliers
- What technologies should we use for business-to-business (B2B) integration?



Web Services — Motivation

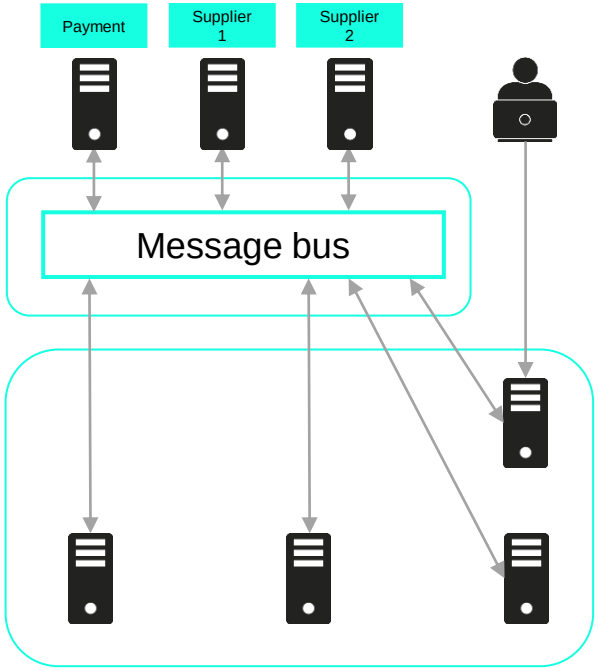
- How to achieve full B2B integration?



Web Services — Motivation

- How to achieve full B2B integration?

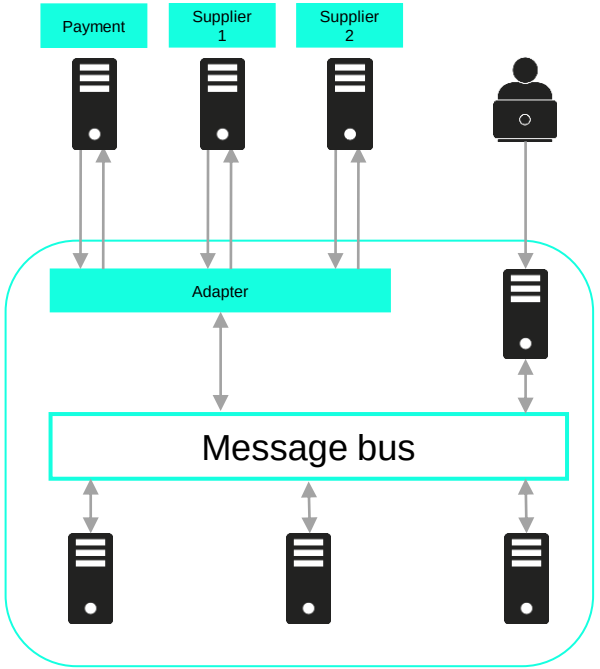
1. B2B service bus?
    1. Common middleware platform
    2. Probably hosted by a third party
    3. Advantages and disadvantages?



Web Services — Motivation

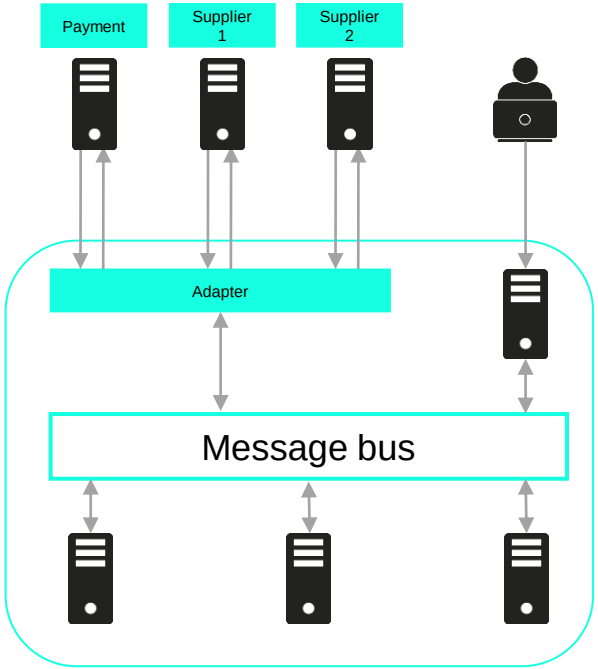
- How to achieve full B2B integration?

1. B2B service bus?
   1. Common middleware platform
   2. Probably hosted by a third party
   3. Advantages and disadvantages?
2. Point-to-point integration?
   1. Common data formats and protocols
   2. Ideally accepted by all business partners
   3. Requires only a single adapter



Web Services — Motivation

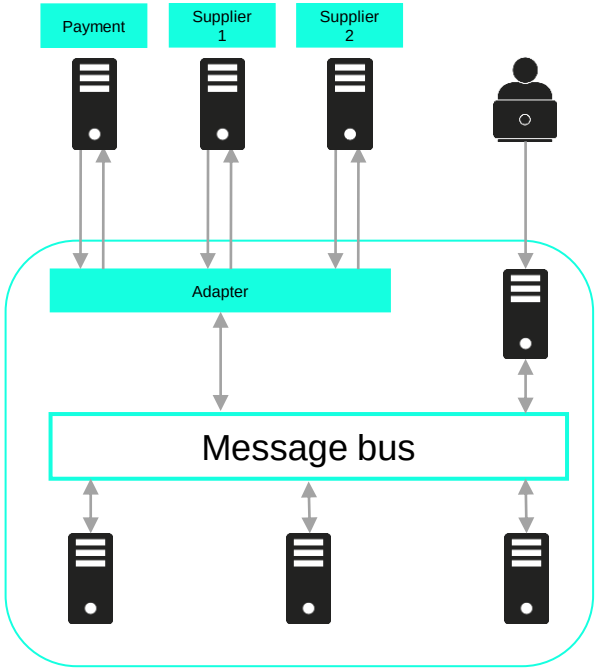
- Globally accepted data formats and protocols already exist in the World Wide Web



Web Services — Motivation

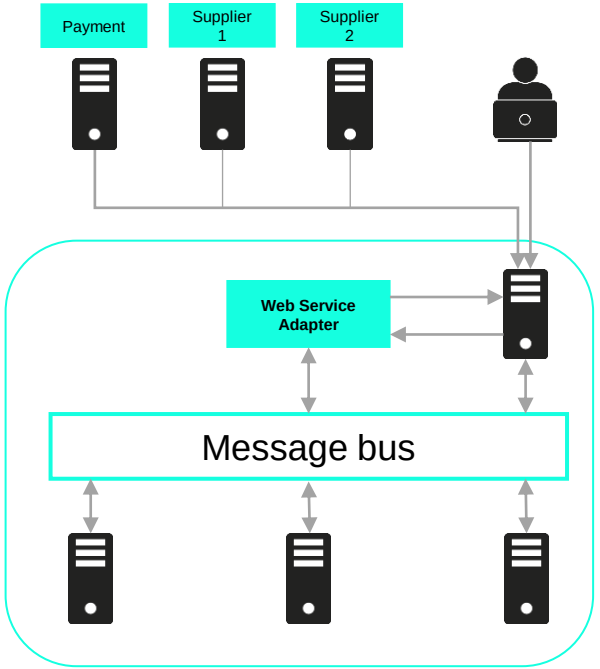
- Globally accepted data formats and protocols already exist in the World Wide Web

- Important examples:
  - HTTP: standard protocol for message exchange between browser and web servers
  - (X)HTML: standard document format for web pages



Web Services — Motivation

- Globally accepted data formats and protocols already exist in the World Wide Web
- Important examples:
  - HTTP: standard protocol for message exchange between browser and web servers
  - (X)HTML: standard document format for web pages
- Idea: Use Web technologies for B2B integration

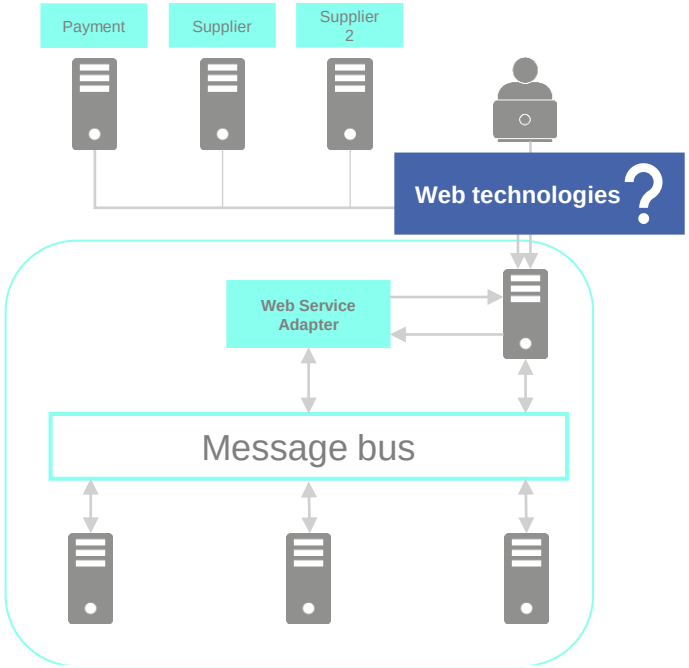


Web Services — Motivation

- Globally accepted data formats and protocols already exist in the World Wide Web

- Important examples:
  - HTTP: standard protocol for message exchange between browser and web servers
  - (X)HTML: standard document format for web pages

- Idea: Use Web technologies for B2B integration



Excursion: Basic Web Technologies

HTTP

HTTP — Definition

Definition

The **Hypertext Transfer Protocol** (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems.

Fielding and Reschke, 2014

- HTTP protocols are used for:
  - The communication and interaction with Web Services
  - For text-based request-responses
  - Data exchange over network based on rules

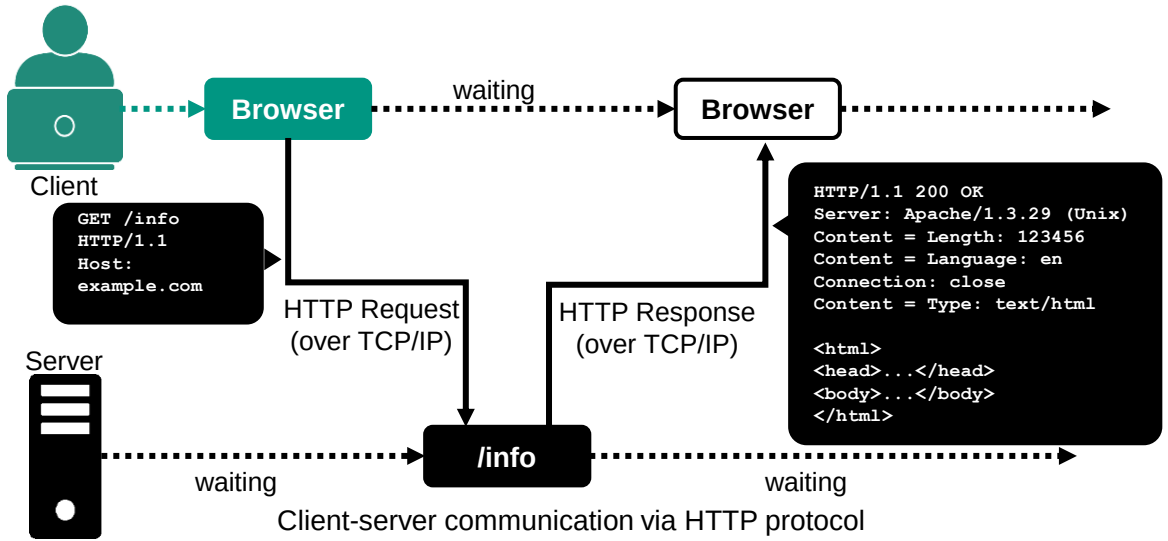
Source: Fielding R, Reschke J (2014) Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. https://tools.ietf.org/html/rfc7230. Accessed 17 September 2019

HTTP — Basics

- The data is encapsulated in a message format
- The structure of a HTTP message contains different parts
  - The start line shows the requested method or response status
  - The optional message header consists out of name-value pairs that describe additional arguments needed for data transmission
  - The optional message body carries the data payload
- HTTP messages are in plaintext. Encryption possible, resulting in HTTPS (Hypertext Transfer Protocol Secure)
- Transport Layer Security (TLS) is used for encryption

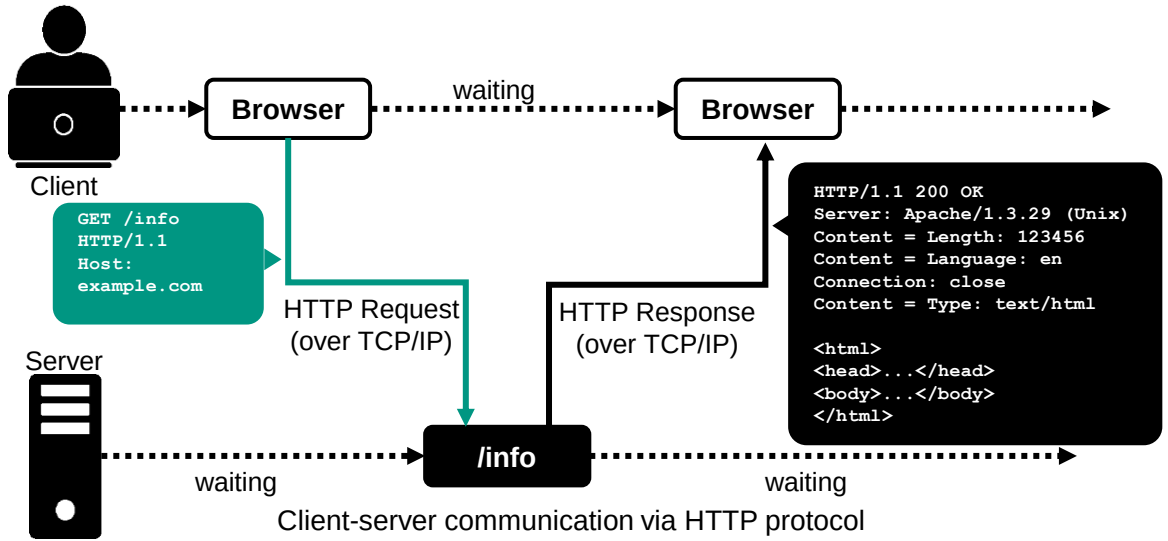
HTTP — Process

1. Client retrieves website with the address *http://example.com/info* from a Web server



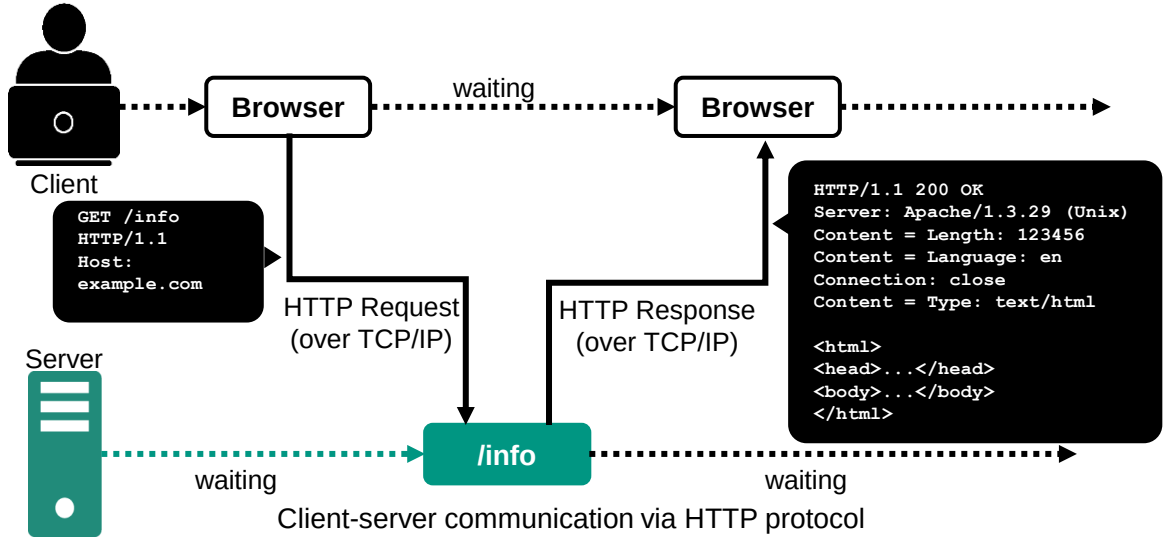
HTTP — Process

1. Client retrieves website with the address *http://example.com/info* from a Web server
2. A HTTP request message is sent over a TCP/IP connection to the Web server associated with the domain *example.com*



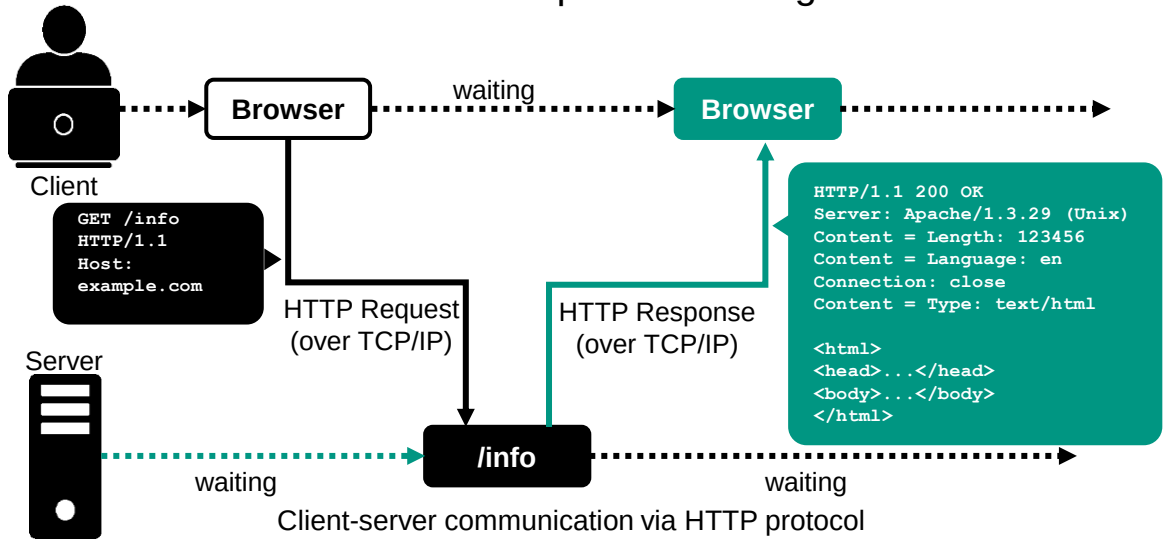
HTTP — Process

1. Client retrieves website with the address *http://example.com/info* from a Web server
2. A HTTP request message is sent over a TCP/IP connection to the Web server associated with the domain *example.com*
3. With *info*, the identified resource is accessed



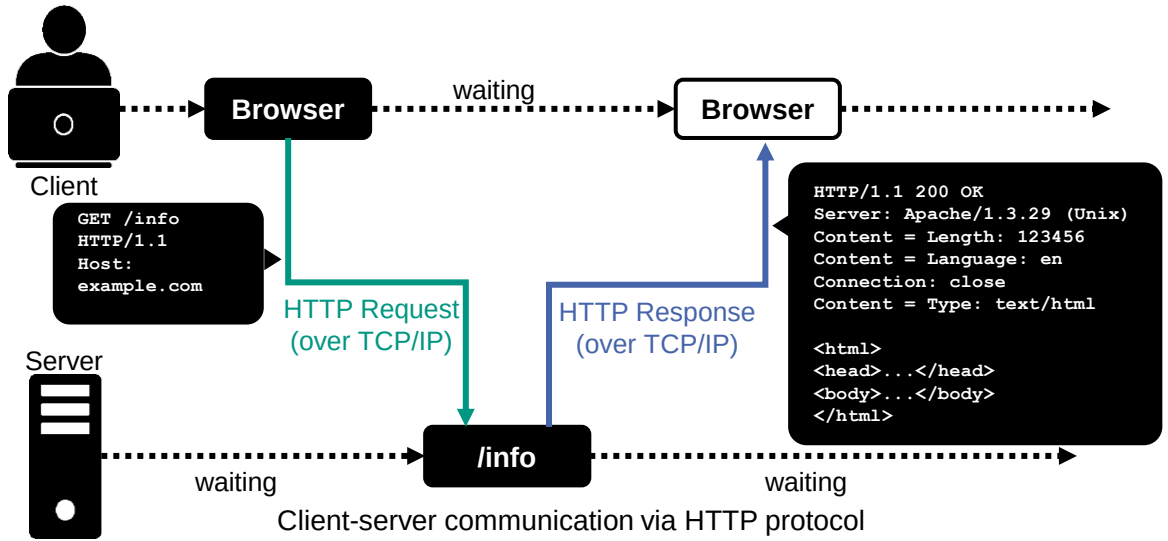
HTTP — Process

1. Client retrieves website with the address *http://example.com/info* from a Web server
2. A HTTP request message is sent over a TCP/IP connection to the Web server associated with the domain *example.com*
3. With *info*, the identified resource is accessed
4. The Web site data is returned via a HTTP response message



HTTP — Request vs. Response Message

- The **HTTP request** message is sent from client to server
- The **HTTP response** message is sent from server to client, as a response



HTTP — Request Message

- The starting line specifies which operation the server is supposed to perform

- There are several different request methods:
  - GET requests the resource's representation
  - HEAD is identical to GET, but only returns the response message's header
  - PUT stores payload data in a request body as a resource, specified by the given URI

- URI (Uniform Resource Identifier) has the (simplified) form: scheme://host/path/to/resource?query

HTTP — Response Message

- The starting line of an HTTP Response contains:
  - The protocol version
  - A status code
  - A message indicating corresponding HTTP request status

- There are several HTTP status codes:
  - Status code 200 with the message OK is the standard response if an operation was successful
  - Status code 301 with the message Moved Permanently states that the resource asked for, was moved to another URI
  - Status code 403 with the message Forbidden states that the client has no access rights

HTTP — Caching

- One major advantage of HTTP is caching
- Caching allows to satisfy large amounts of requests, that enables scalable Web services
- Functionality:
  - Previous HTTP responses are stored
  - The stored responses answer future HTTP requests
  - This leads to a faster response
- Caching works for common requests
  - It only works with methods, which do not modify the resource, like GET

- Example:
  - Amazon receives huge amounts of requests every second
  - If the whole HTTP process must be performed for each request, Amazon would not be able to satisfy all of those in time
  - Caching allows Amazon to store popular requests
  - Large chunk of the requests can be answered from the cache very fast

XML

XML — Definition

Definition

Extensible Markup Language, commonly abbreviated **XML**, describes a **class of data objects** called XML documents and partially describes the behavior of the computer programs which process them.

W3C, 2008

- XML is widely used for machine-, OS- and middleware-independent data exchange across the web
- XML is different from HTML
- It is designed for storage and exchange of data
- XML supports Unicode characters

Source: W3C (2008) Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/xml/. Accessed 17 September 2019

XML — Structure

Markup consists of XML tags of the following form:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Structure

Markup consists of XML tags of the following form:

1. XML element:
   The content is enclosed by a starting and an ending tag.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Structure

Markup consists of XML tags of the following form:

1. XML element:
   The content is enclosed by a starting and an ending tag.
2. XML attributes:
   Optional key-value pairs are embedded in element tags.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Structure

Markup consists of XML tags of the following form:

1. XML element:
   The content is enclosed by a starting and an ending tag.
2. XML attributes:
   Optional key-value pairs are embedded in element tags.
3. XML comment:
   Text which is ignored by software processing XML documents.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Structure

Markup consists of XML tags of the following form:

1. XML element:
   The content is enclosed by a starting and an ending tag.
2. XML attributes:
   Optional key-value pairs are embedded in element tags.
3. XML comment:
   Text which is ignored by software processing XML documents.
4. Plain text file:
   Binary data must be additionally encoded.

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Document

- An XML document can be portrayed as a tree of elements
  - Prolog declaration carries processing instructions (e.g., version, encoding)
  - Each XML element can have multiple child elements

- An XML document is well-formed if:
  1. It contains only legal Unicode text
  2. It has a single root element
  3. Each tag is opened and closed
  4. Tags are correctly nested
  5. Elements have no duplicate attributes

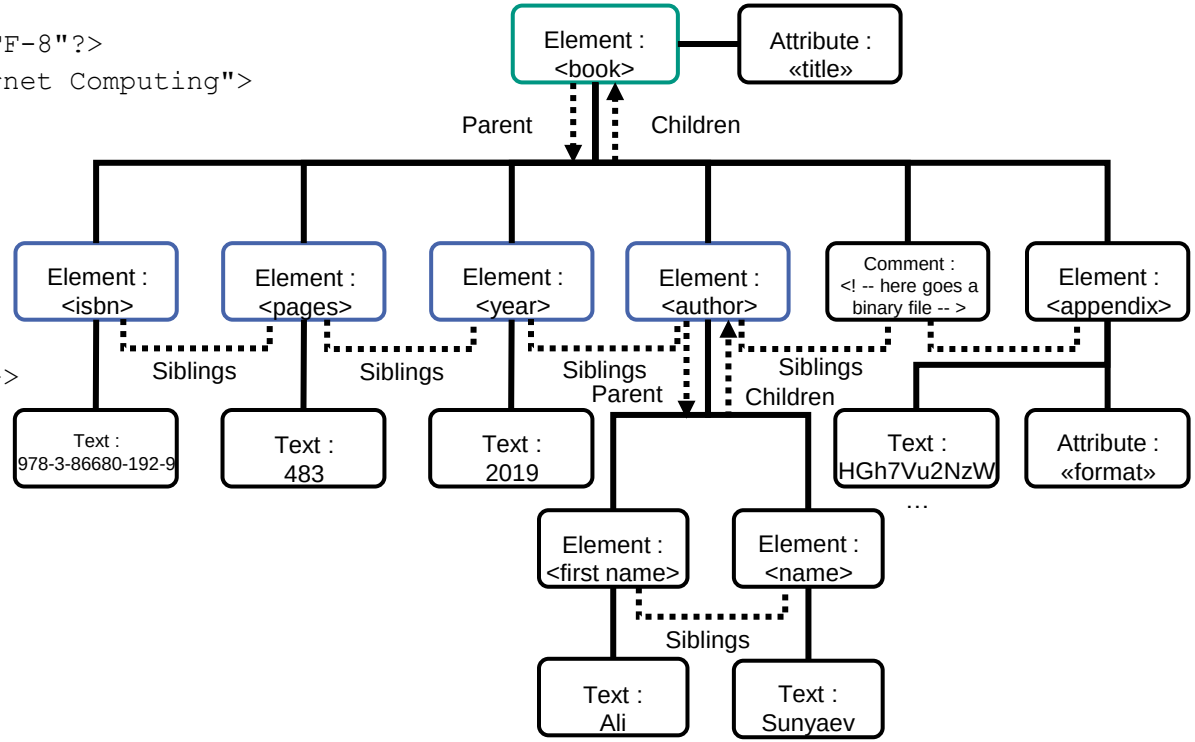
XML — Tree: Parent, Child, and Sibling

- Besides as code, an XML document can be portrayed as a tree
- Elements can have content, text or child elements
- Example: The element **<book>** has content. This content consists of **six child elements**
- Vice versa, the element <pages> has the parent <book>
- Two child elements, with the same parent element, are siblings

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```

XML — Tree: Parent, Child, and Sibling

```
1   <?xml version="1.0" encoding="UTF-8"?>
2   <book title="Principles of Internet Computing">
3    <isbn>978-3-86680-192-9</isbn>
4    <pages>483</pages>
5    <year>2019</year>
6    <author>
7     <firstname>Ali</firstname>
8     <name>Sunyaev</name>
9    </author>
10   <!-- here goes a binary file -->
11   <appendix format="pdf">
12    HGh7Vu2NzW…
13   </appendix>
14  </book>
```



XML — XSD Definition

Definition

An **XML schema definition (XSD)** aims at defining and describing a class of XML documents via schema components to constrain and document the meaning, usage and relationships of their constituent parts: datatypes, elements, as well as their content, attributes and values.

W3C, 2012

- XML parse only accepts documents that comply with XML syntax rules
- Besides correct syntax, there are additional rules documents have to fulfill
- The XML schema definition specifies those additional rules

Source: W3C (2012) W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. https://www.w3.org/TR/xmlschema11-1/. Accessed 17 September 2019

Web Service Architecture

Service

Definition

A **service** is a logical representation of a repeatable business activity that has a specific outcome, is self-contained, may consist of other services, and is a black box to the service's consumers.

The Open Group, 2009

Example:

- Credit Card company's service that checks a credit card number's validity
  ➔ repeatable business activity
- The outcome is the assessment of the credit card
  ➔ specific outcome, self-contained
- There is no information about which databases are accessed
  ➔ black box

Source: The Open Group (2009) SOA source book. Van Haren Publishing, Zaltbommel, Netherlands

Service Oriented Architecture

Definition

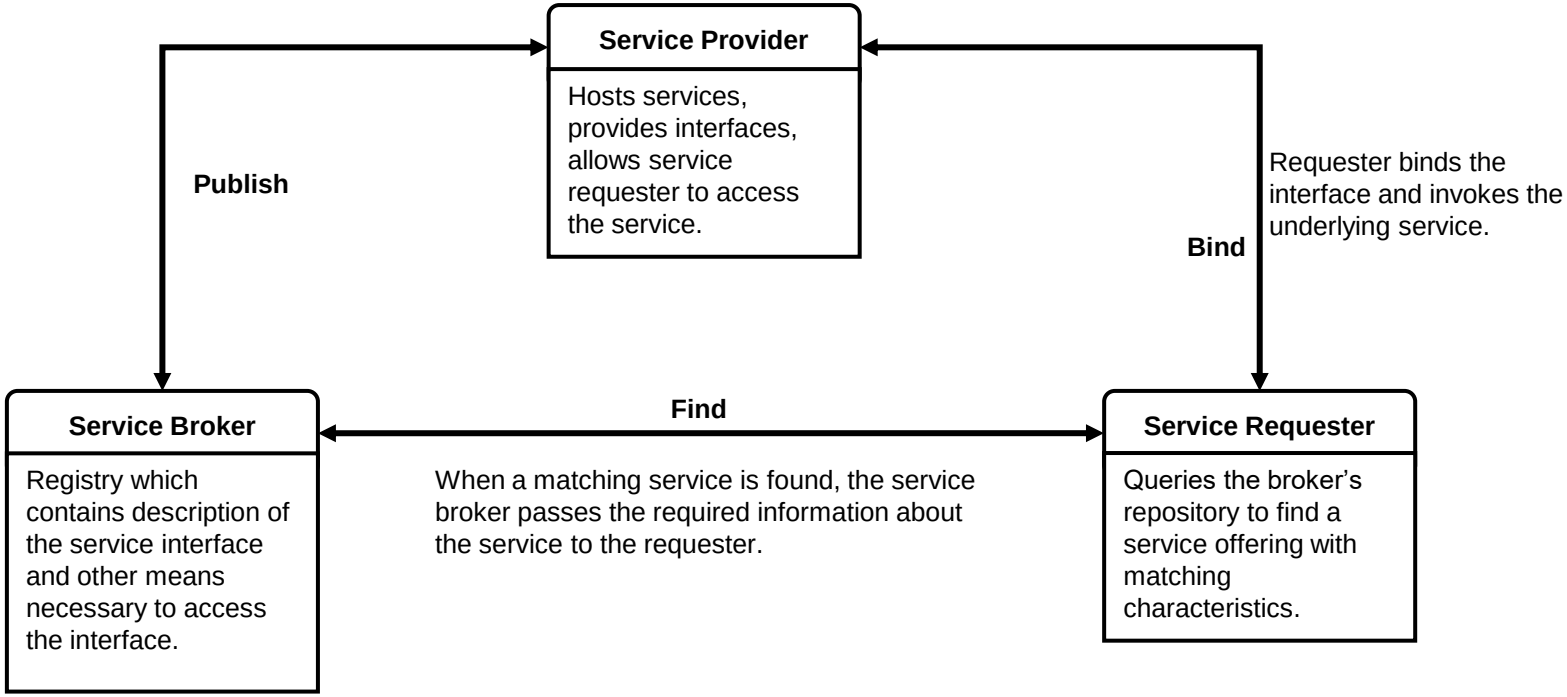
Service Oriented Architecture (SOA) represents a set of principles and methodologies for designing and developing software in the form of interoperable services. These services are well-defined business functions that are built as software components i.e., discrete pieces of code and/or data structures that can be reused for different.

NIST, 2013

- The segregation of service providers and requesters makes SOA highly service centered.
- It becomes possible to have multiple providers offering the same service,
- and to add or replace different service providers that offer the same service, while not affecting the service requester.
- Service requesters can automatically locate services and providers.

Source: NIST (2013) Security and Privacy Controls for Federal Information Systems and Organizations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf. Accessed 17 September 2019.

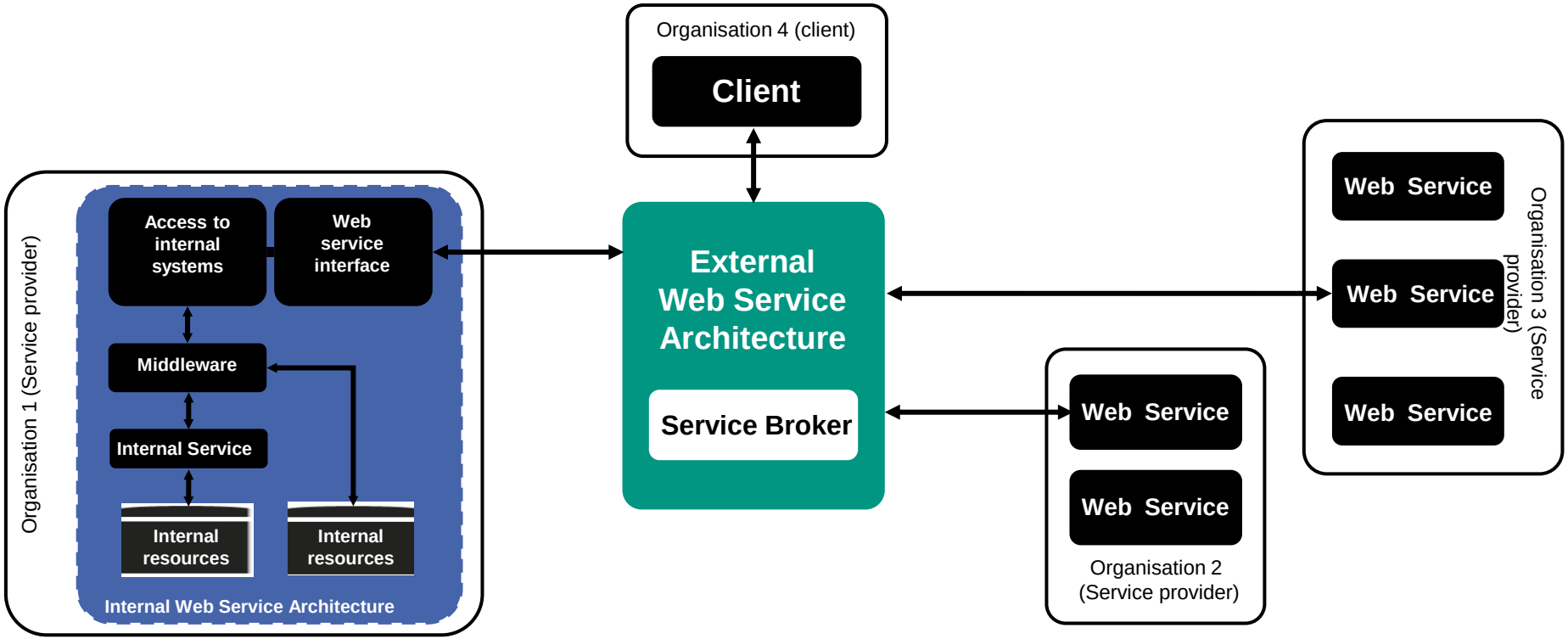
Service Oriented Architecture



Internal and External Web Service Architecture

- Web services: Allow external clients to access internal operations via a network
  - **Internal** and **external** perspective
- Two middleware architectures that share a common service interface
- **Internal Web services**, contain internal operations and resources
- **External Web services**, facilitate external clients' access to the interface
- For the interaction between internal and external architecture Web standards like XML, HTTP are used

Internal and External Web Service Architecture



SOAP

SOAP — Basics

- Web service use a XML messaging architecture
- SOAP standards are used for the message format
- Originally: Simple Object Access Protocol
- Requests and responses follow a standardized message format

- Advantages:
  - High security
  - Standardized approach
  - Extensible functionality

- Disadvantages:
  - Requires more network bandwidth
  - More complex interfaces

SOAP — Visualized

- SOAP Web service architecture stack
- Illustrates the layering between different concepts
- Messages are centered, SOAP is based on messages
- WSDL is used for service description.
- Additional standards define means for:
  - Secure message exchanges
  - Service management
  - Service discovery

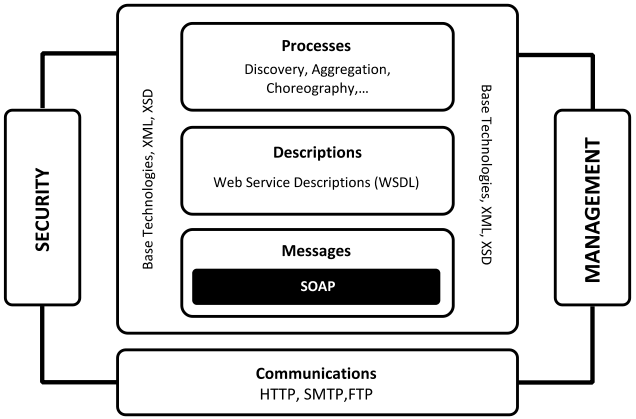


Image source: Adapted from W3C (2004) Web Service Architecture. https://www.w3.org/TR/ws-arch/. Accessed 17 September 2019

SOAP — Basics

- Transport of SOAP messages
  - SOAP provides a mechanism, which allows to use any underlying transport protocol, for example HTTP
- SOAP Message Processing
  - Consist of several components
  - Clients generate messages that are sent to Web services
  - Messages have a body an optional header
  - Messages pass multiple intermediaries
  - A forward intermediary relays the message and can change headers
  - An active intermediary can change the body of a message
- Example:
  - An organization employs a service
  - The service attaches a digital signature header to each outbound message leaving the intranet
  - This allows the receiver to verify a message's integrity

SOAP — Message Format

- SOAP Message Format
  - Client request or Web server response
  - XML document
  - Root element is the SOAP envelope, it closes the header and body
  - SOAP Header: optional, application specific information (authentication, routing data)
  - SOAP Body: mandatory, encloses the actual message
  - SOAP Fault: informs Web service clients about errors that occurred while transmitting or processing the preceding service request

```
1   POST /Library HTTP/1.1
2   Host:example.org
3   Content-Type: application/soap+xml; charset=utf-8
4   Content-Length: 1234
5
6   <?xml version="1.0"?>
7   <soap:Envelope
8   xmlns:soap="http://www.w3.org/2003/05/soap-envelope/">
9    <soap:Body xmlns:lib="http://www.example.org/library">
10    <lib:GetFirstAuthor>
11     <lib:ISBN>978-3-86680-192-9</lib:ISBN>
12    </lib:GetFirstAuthor>
13   </soap:Body>
14  </soap:Envelope>
```

Example of a SOAP HTTP Request
Header: row 1 – 4
Body: row 6 – 14
This code represents a successful transition, therefore there is no fault

SOAP — Service Descriptions with WSDL

- An understanding of the functionality of a Web Service is necessary
- Interface descriptions can be used for that
- This is implemented through WSDL

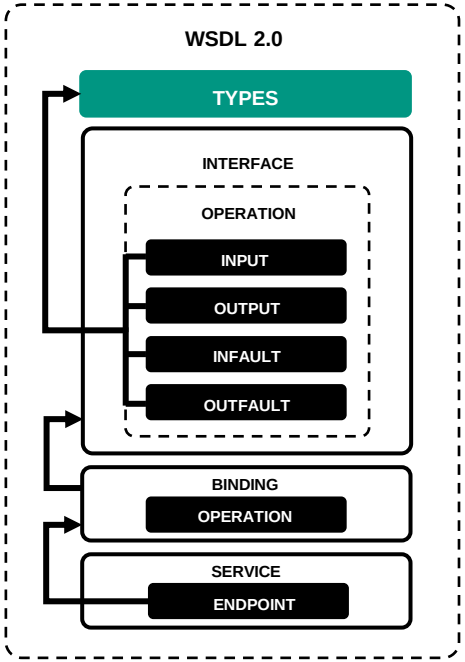
- WSDL:
  - It is not necessary for clients to have any knowledge about the Web service's actual implementation
  - Clients only need to know, what messages can be sent and what responses can be expected
  - This is stated in a WSDL file
  - A WSDL file is a XML document

- SOAP and WSDL provide:
  - Flexible XML-based messaging infrastructure
  - Standard service description method

- Example:
  - Web Services Management: Communication protocol. It simplifies administrative data access and exchange within IT infrastructure and shows the client common operations

SOAP — Structure of WSDL

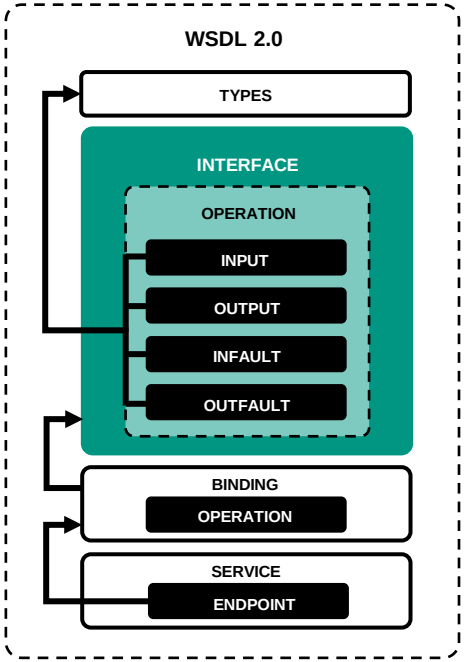
- Types:
  Describe the complex data types a Web service understands and uses to exchange data with its clients



SOAP — Structure of WSDL

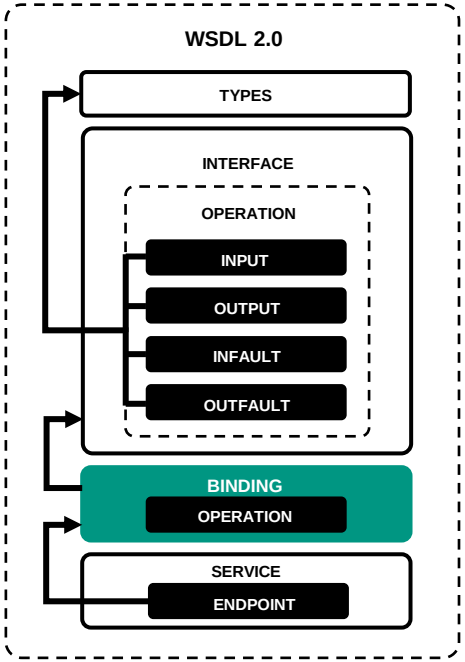
- Types:
  Describe the complex data types a Web service understands and uses to exchange data with its clients

- Interface:
  Describes all operations that a Web service offers to clients



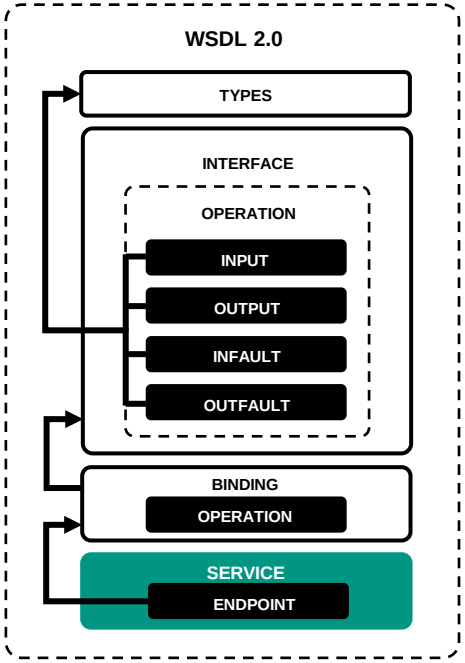
SOAP — Structure of WSDL

- Types:
  Describe the complex data types a Web service understands and uses to exchange data with its clients
- Interface:
  Describes all operations that a Web service offers to clients
- Binding:
  Necessary technical details clients require to access interface operations



SOAP — Structure of WSDL

- Types:
  Describe the complex data types a Web service understands and uses to exchange data with its clients
- Interface:
  Describes all operations that a Web service offers to clients
- Binding:
  Necessary technical details clients require to access interface operations
- Service:
  Specify the service's name and defines endpoints



SOAP — WSDL Code Example

- Here, a Web service by the name **glossaryTerms** is defined
- It defines a single operation called **getTerm**
  - The operation expects the input message **getTermRequest**
  - The output message is **getTermResponse**
- Involved messages and used datatypes are also defined

```
1  <message name="getTermRequest">
2     <part name="term" type="xs:string"/>
3  </message>
4
5  <message name="getTermResponse">
6     <part name="value" type="xs:string"/>
7  </message>
8
9  <portType name="glossaryTerms">
10    <operation name="getTerm">
11       <input message="getTermRequest"/>
12       <output message="getTermResponse"/>
13    </operation>
14 </portType>
```

Simplified fraction of a WSDL document

Source: https://www.w3schools.com/xml/xml_wsdl.asp

SOAP Summary

	SOAP Web service
Design	Standardized SOAP protocol framework
Approach	Function-driven
Statefulness	Stateless or stateful
Caching	Requests cannot be cached by default
Security	WS-Security with TLS support
Performance	Large message overhead (SOAP envelope)
Transfer protocol	E.g., HTTP, SMTP, or UDP
Message format	Mainly XML (SOAP)
Pros	High security, standardized approach, extensible functionality
Cons	Requires more network bandwidth, more complex interfaces

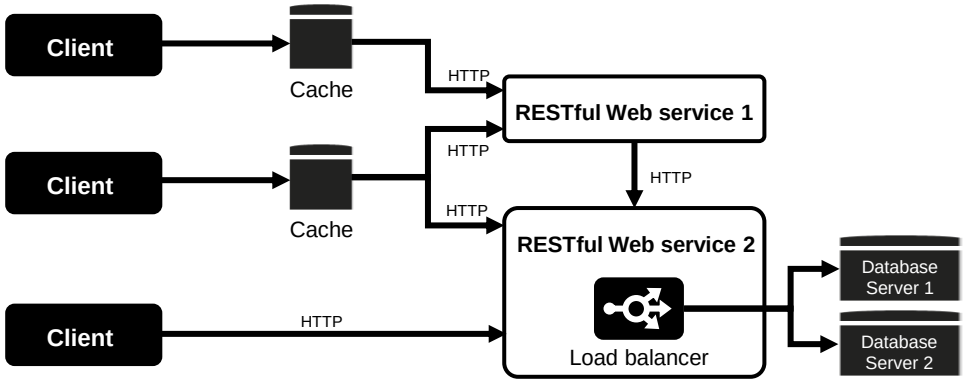
RESTful

RESTful Web Services

- Representational State Transfer (REST)
  - Architectural style for designing loosely coupled applications over HTTP
  - When REST is used as basis for Web services, they are referred to as RESTful

- RESTful Web Services
  - Build on the client-server architecture pattern
  - Are inherently stateless and focused on providing basic read and write operations to the exposed resources
  - A resource can be any coherent and meaningful concept like structured data and texts
  - Clients usually interact with the resources' representations
  - RESTful Web services can convert resources between different representational states
  - Clients do not have to differentiate between the resources' actual formats and their representations

RESTful Web Services



Six Constraints of RESTful Web Services

- RESTful Web services are defined by six constraints (Fielding RT, 2000) which are explained in the following slides:
  1. Built on the client-server architecture pattern
  2. Provides a uniform communication interface
  3. Interactions with RESTful Web services are stateless
  4. Capability to cache server responses
  5. RESTful systems are layered
  6. RESTful Web services may provide downloadable code

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

1. **Built on the client-server architecture pattern:**
   - Client-server architecture separates clients and servers and decouples user interface concerns from data storage and data processing concerns
   - Loose coupling allows each component to scale and evolve independently
   - Server provides access to information over an efficient and standardized interface
   - Client retrieves this information and presents it appropriately to users or other systems
   - Relies heavily on the provision of a reliable interface between components

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

2. Provides a uniform communication interface:

- RESTful Web services provide a uniform communication interface between components
- Interfaces are typically implemented via URIs, which allow for accessing, creating, and manipulating server resources
- Client requests and service responses are self-descriptive and provide information to explain how requests and responses should be processed
- The resulting communication simplification between components facilitates understanding interactions between components of a RESTful Web service

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

3. Interactions with RESTful Web services are stateless:

- The server neither stores client-specific context information, nor maintains a persistent session state in the server
- Therefore, all context-specific and client-specific information that are required must be included in the request
- Example: when a Web service requires clients to authenticate themselves with a username and password, each client request must include the login credentials

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

4. Capability to cache server responses:

- RESTful systems have the capability to cache server responses in order to improve network efficiency, scalability, and performance
- A response can be cached by a client and reused for consecutive equivalent requests without resending the request to the Web service
- Consequently, cacheable responses require requests to be idempotent

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

5. RESTful systems are layered:

- RESTful systems consist of two or more component layers, including separate layers for clients and servers
- Each component can only interact with the components of the layers immediately above or below it
- This facilitates replacing and extending individual components, as resulting changes for other components will be limited to adjacent layers

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

Six Constraints of RESTful Web Services

6. RESTful Web services may provide downloadable code (optional):

- RESTful Web services may provide downloadable code
- A requesting application's functionality can be extended via program code that is executed on the client system
- Example: Extension for message decryption and encryption functionalities
  - →Clients could download this extension from the Web service and invoke its functions locally to enable secure end-to-end communication with or over the Web service
- RESTful Web services have little to no control over a client's ability to actually execute the distributed program code

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

RESTful Web Service Interactions

- A major benefit of the basic set of HTTP operations is that it makes service interactions simpler, more predictable, and idempotent
- The responses that implement an HTTP interface can be easily cached.
  - → Facilitates the service's scalability and reliability
- The RESTful Web services' uniform interface design only provides basic operations for creating, deleting, reading and updating resources (CRUD)
- HTTP request methods are suitable for these operations
- HTTP is, therefore, the preferred communication protocol for RESTful Web service interactions

Source: Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA

RESTful Web Service Interactions

- The basic idea of RESTful HTTP interactions is that each resource can be identified using a unique URI, usually of the following format:
- <protocol>://<service-name>/<ResourceType>/<ResourceID>

- Client sends an HTTP request to the URI:
  - Which operation the service performs is determined by the request's HTTP request method specified in its header (CRUD)
  - The request's body may contain additional parameters (e.g., username and password)
  - The HTTP accept header field specifies the desired resource representation in the Web service's response

```
1  GET /books/author/sunyaev HTTP/1.0
2  Host: example.com
3  Accept: application/json
```

RESTful Web Service Interactions

- Example of a RESTful HTTP Request:
  - Shows a simple HTTP message that requests a list of books written by the author with the name *Sunyaev*
  - Desired service operation in this case is read → HTTP request method is GET
  - The resource, i.e., the list of books, is identified by a self-explanatory URI: http://example.com/books/author/sunyaev
  - The request specifies that the resource's representation should be JSON (JavaScript Object Notation)

RESTful Web Service Interactions

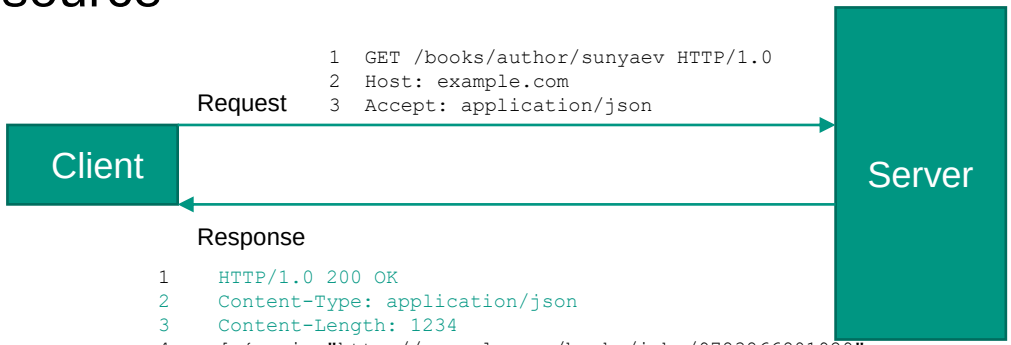
- Example of a RESTful HTTP Request:
  - Shows a simple HTTP message that requests a list of books written by the author with the name *Sunyaev*
  - Desired service operation in this case is read → HTTP request method is GET
  - The resource, i.e., the list of books, is identified by a self-explanatory URI: http://example.com/books/author/sunyaev
  - The request specifies that the resource's representation should be JSON (JavaScript Object Notation)

RESTful Web Service Interactions

- Example of a RESTful HTTP Request:
  - Shows a simple HTTP message that requests a list of books written by the author with the name *Sunyaev*
  - Desired service operation in this case is read → HTTP request method is GET
  - The resource, i.e., the list of books, is identified by a self-explanatory URI: http://example.com/books/author/sunyaev
  - The request specifies that the resource's representation should be JSON (JavaScript Object Notation)

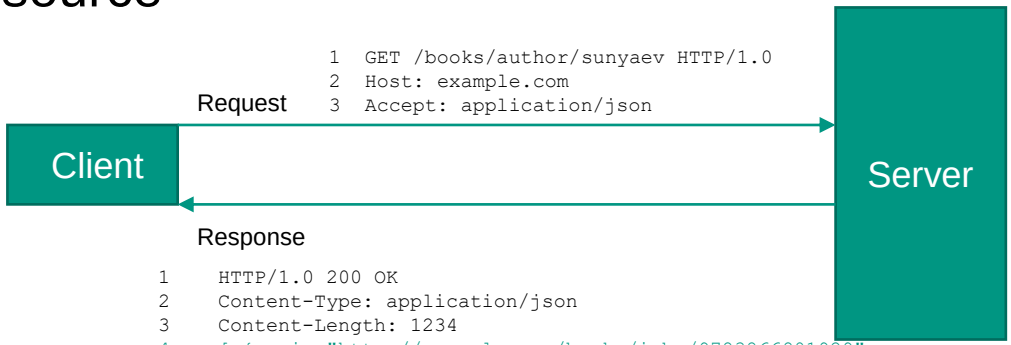
RESTful Web Service Interactions

- The RESTful Web service returns an HTTP response with its payload containing the list of books in JSON notation, which matches the specified resource



RESTful Web Service Interactions

- The RESTful Web service returns an HTTP response with its payload containing the list of books in JSON notation, which matches the specified resource



Exposing RESTful Web Services

- In order to use a RESTful Web service, clients require knowledge about the available resources, request parameters, data formats, and operations
- SOAP Web services use WSDL to expose their functionality to clients
- The RESTful Web services' interfaces are usually more simplistic, this results in less need for complex machine-readable service descriptions
- RESTful interactions are often generated and processed by the client via light-weight libraries

Exposing RESTful Web Services

- A sufficiently large number of resources or customized functionalities could lead to even RESTful Web services requiring more documentation
- This led to the development of XML-based languages to describe RESTful Web service interfaces
- The most important example is the Web Application Description Language (WADL), which is a machine-readable XML variation for HTTP-based Web services

WADL Example:

```
<application
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://exmple.com/wadl.xsd"
  xmlns:lib="http://example.com/library"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://example.com/library">
  <resources base="http://example.com">
   <resource path="orderService">
    <method name="POST" id="orderBook">
     <request>
      <param name="isbn" type="xsd:string"
       style="query" required="true"/>
      <param name=”quanitity” type="xsd:int"
       style="query" default="1"/>
     </request>
     <response status="200">
      <representation mediaType="application/xml"
       element="lib:OrderResponse"/>
      <response status="400">
       <representation mediaType="application/xml"
        Element="lib:Error"/>
      </response>
     </response>
    </method>
   </resource>
  </resources>
</application>
```

Characteristics of RESTful Web Services

- Architectural style with six nonspecific constraints
- Approach focuses on the resources' representation
- Interactions with RESTful Web services are stateless
- Responses can be cached by the client
- The Messages have a small message overhead
- HTTP as transfer protocol
- Message format is flexible (e.g., HTML, XML, JSON, plain text)

- Advantages:
  - Scalable
  - Requires fewer resources
  - Requests cacheable

- Disadvantages:
  - Less security and flexibility

Differences Between RESTful and SOAP

	SOAP Web Service	RESTful Web Service
Design	Standardized SOAP protocol framework	Architectural style with nonspecific constraints
Approach	Function-driven	Resource-driven
Statefulness	Stateless or stateful	Stateless
Caching	Requests cannot be cached by default	Cacheable
Security	WS-Security with TLS support	Supports HTTPS and TLS
Performance	Large message overhead (SOAP envelope)	Small message overhead
Transfer protocol	E.g., HTTP, SMTP, or UDP	Mainly HTTP
Message format	Mainly XML (SOAP)	E.g., HTML, XML, JSON, or plain text
Pros	High security, standardized approach, extensible functionality	Scalable, requires fewer computing and networking resources, requests cacheable by default
Cons	Requires more network bandwidth, more complex interfaces	Less secure, less flexible, and less functional flexibility

References

- Fielding R, Reschke J (2014) Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. https://tools.ietf.org/html/rfc7230. Accessed 17 September 2019
- Fielding RT (2000) Architectural Styles and the Design of Network-based Software Architectures. Dissertation, University of California, Irvine, CA, USA
- Mohamed K, Wijesekera D (2012) Performance Analysis of Web Services on Mobile Devices. Procedia Computer Science 10:744-751
- NIST (2013) Security and Privacy Controls for Federal Information Systems and Organizations. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r4.pdf. Accessed 17 September 2019
- The Open Group (2009) SOA source book. Van Haren Publishing, Zaltbommel, Netherlands
- W3C (2004) Web Service Architecture. https://www.w3.org/TR/ws-arch/. Accessed 17 September 2019
- W3C (2008) Extensible Markup Language (XML) 1.0 (Fifth Edition). https://www.w3.org/TR/xml/. Accessed 17 September 2019
- W3C (2012) W3C XML Schema Definition Language (XSD) 1.1 Part 1: Structures. https://www.w3.org/TR/xmlschema11-1/. Accessed 17 September 2019

Questions

Questions

1. What is a Web service?
2. What are the design principles behind HTTP?
3. What is the difference between a valid and a well-formed XML document?
4. What is XSD and why is it needed?
5. What is SOAP and what functionalities does it provide?
6. What is the purpose of a WSDL document?
7. Explain how XSD can be used in the context of SOAP-based Web services and WSDL?
8. What are RESTful Web services and how do they differ from SOAP Web services?

Further Reading

- Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services. In: Alonso G, Casati F, Kuno H, Machiraju V (eds) Web services: concepts, architectures and applications. Data-centric systems and applications, 1st edn. Springer, Berlin.
- Møller, A., & Schwartzbach, M. (2006). An Introduction to XML and Web Technologies, chapter Schema Languages. Addison-Wesley, Harlow, England, 92-187.
- Papazoglou MP (2012) Web services and SOA: principles and technology, 2nd edn. Pearson, Harlow.
- Pautasso, C., Wilde, E., & Alarcon, R. (Eds.). (2013). REST: advanced research topics and practical applications. Springer Science & Business Media.
- The Open Group (2009) SOA source book. Van Haren Publishing, Zaltbomme