Parallel & Distributed Computing (WQD7008)

Week 5

Service Oriented Architecture

2019/2020 Semester 1

Dr. Hamid Tahaei

Service-oriented architecture (SOA) Introduction

- SOA is about how to design a software system that makes use of services of new or legacy applications through their published or discoverable interfaces.
- These applications are often distributed over the networks.
- SOA aims to make service interoperability extensible and effective.
- It prompts architecture styles such as loose coupling, published interfaces, and a standard communication model in order to support this goal.
- Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects.
- World Wide Web Consortium (W3C) defines SOA as a form of distributed systems architecture characterized by the following properties:
  - Logical view
  - Message orientation
  - Description orientation
  - Granularity
  - Network orientation
  - Platform-neutral

Service-oriented architecture (SOA)

- SOA, as the name says is an **architectural concept** which focuses on having different services communicating with each other to carry out a bigger job.
- Thus, a web service is a basic building block in a SOA. When multiple services are combined, we have an application that falls under SOA.
- SOA is a structure that allows services to communicate with each other across different platforms and languages by implementing what is known as a "loose coupling" system.
- SOA is Machine (service) to machine (service) communication
- In an SOA paradigm, software capabilities are delivered and consumed via loosely coupled, reusable, coarse-grained, discoverable, and self-contained services interacting via a message-based communication model.

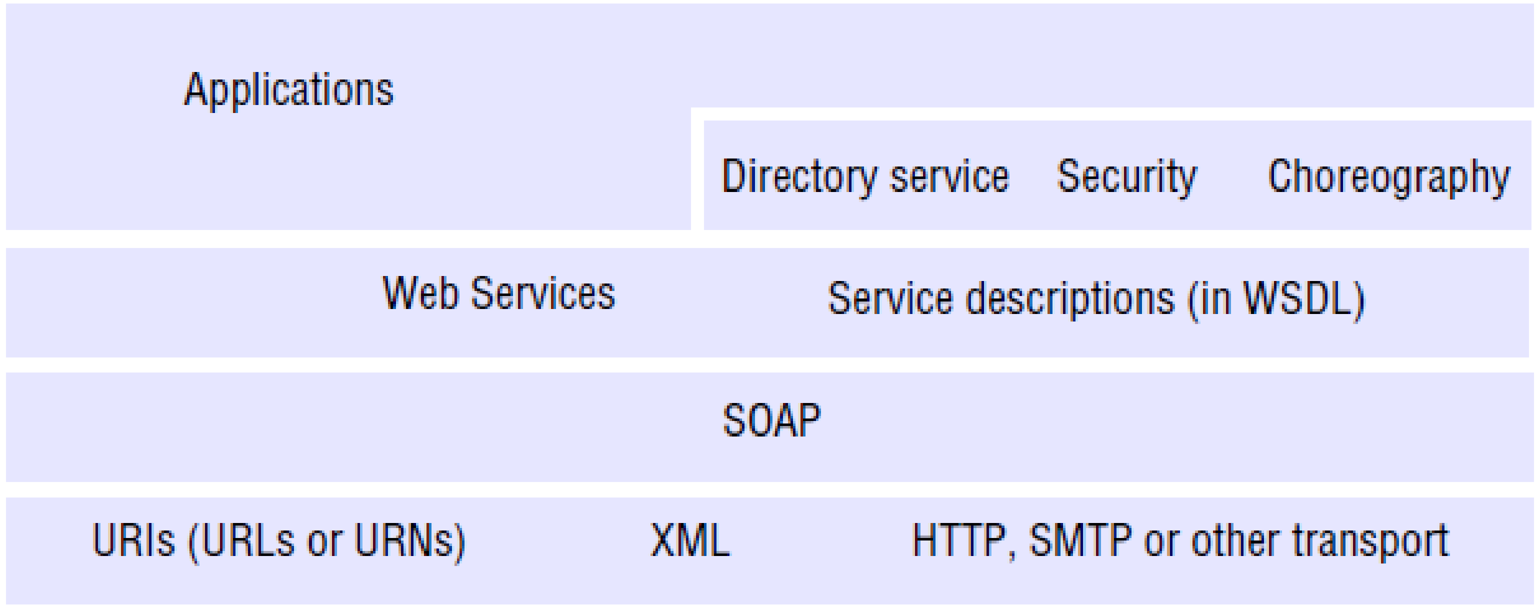
Usecase:

- **Banking:** card transaction, credit cards, authorization, and etc.
- **Manufacturing:** Inventory management (keep tracking of the number of materials)
- **Insurance**
- **E-commerce**
- **Online purchase**
- **And many more…**

Service-oriented architecture (SOA) Introduction

- **Web services** provide an infrastructure for maintaining a richer and more structured form of interoperability between clients and servers. They provide a basis whereby a client program in one organization may interact with a server in another organization without human supervision.
- **Web services** allow complex applications to be developed by providing services that integrate several other services. Due to the generality of their interactions, web services cannot be accessed directly by browsers.
- **Web server** provides a basic HTTP service.
- **Web service** provides a service based on the operations defined in its interface.
- Extensible Markup Language (XML), External data representation and marshalling of messages exchanged between clients and web services is done in XML.
- **The SOAP protocol** specifies the rules for using XML to package messages, for example to support a request-reply protocol.
- **SOAP** is used to encapsulate these messages and transmit them over HTTP or another protocol, for example, TCP or SMTP. A web service deploys service descriptions to specify the interface and other aspects of the service for the benefit of potential clients.

Service-oriented architecture (SOA) Introduction



Web services infrastructure and components

- **SOAP** is used to encapsulate these messages and transmit them over HTTP or another protocol, for example, TCP or SMTP. A web service deploys service descriptions to specify the interface and other aspects of the service for the benefit of potential clients.

Web Services

- **A web service interface** generally consists of a collection of operations that can be used by a client over the Internet. The operations in a web service may be provided by a variety of different resources, for example, programs, objects or databases.
- **A web service** may be managed by a web server along with web pages; or it may be a totally separate service.
- **The key characteristic** of most web services is that they can process XML formatted SOAP messages. An alternative is the REST approach Each web service uses its own service description to deal with the service-specific characteristics of the messages it receives.

Web Services

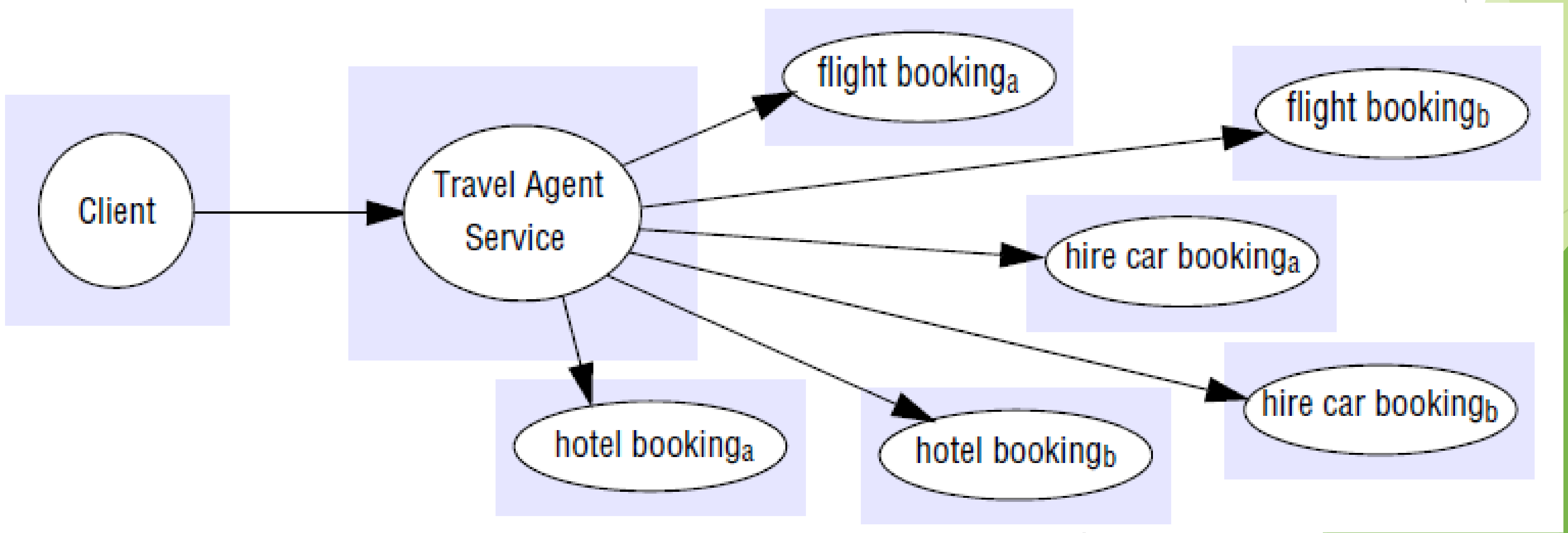
Example of web servers

- Many well-known commercial web servers including Amazon, Yahoo, Google and eBay, offer web service interfaces that allow clients to manipulate their web resources. As an example, the web service offered by Amazon.com provides operations to allow clients to get information about products, to add an item to a shopping cart or to check the status of a transaction.

Web Services

Combination of web services

- Provides an interface for a web service allows its operations to be combined with those of other services to provide new functionality.
- example of the benefits of combining several services, consider the fact that many people book flights, hotels and rental cars for trips online using a variety of different web sites. If each of these web sites were to provide a standard web service interface, then a 'travel agent service' could use their operations to provide a traveller with a combination of these services.



The 'travel agent service' combines other web services

Web Services

Communication pattern

- The processing of a booking takes a long time to complete and could well be supported by an asynchronous exchange of documents, starting with the details of the dates and destinations, followed by a return of status information from time to time and eventually the details of completion. Performance is not an issue here.
- The checking of credit card details and the interactions with the client should be supported by a request-reply protocol.

*Web services are designed to support distributed computing in the Internet, in which many different programming languages and paradigms coexist. Hence, they are designed to be independent of any particular programming paradigm.

Web Services

Loose coupling

- There is considerable interest in *loose coupling* in distributed systems, particularly in the web services community. The terminology is often ill-defined and imprecise, though. In the context of web services, loose coupling refers to minimizing the dependencies between services in order to have a flexible underlying architecture (reducing the risk that a change in one service will have a knock-on effect on other services). This is partially supported by the intended independence of web services with the subsequent intention to produce combinations of web services as discussed above.

Representation of messages

- Both SOAP and the data it carries are represented in XML.Textual representations take up more space than binary ones and the parsing that they require takes more time to process. In document-style interactions speed is not an issue, but it is important in request-reply interactions. However, it is argued that there is an advantage in a human-readable format that allows for the easy construction of simple messages and for debugging of more complex ones.

Web Services

Service references

- In general, each web service has a URI, which clients use to refer to it. The URL is the most frequently used form of URI. Because a URL contains the domain name of a computer, the service to which it refers will always be accessed at that computer. This service reference is known as an *endpoint* in web services.

Activation of services

- A web service will be accessed via the computer whose domain name is included in its current URL. That computer may run the web service itself or it may run it on another server computer. For example, a service with tens of thousands of clients may need to be deployed on hundreds of computers. A web service may run continuously, or it may be activated on demand. The URL is a persistent reference, meaning that it will continue to refer to the service for as long as the server the URL points to exists.

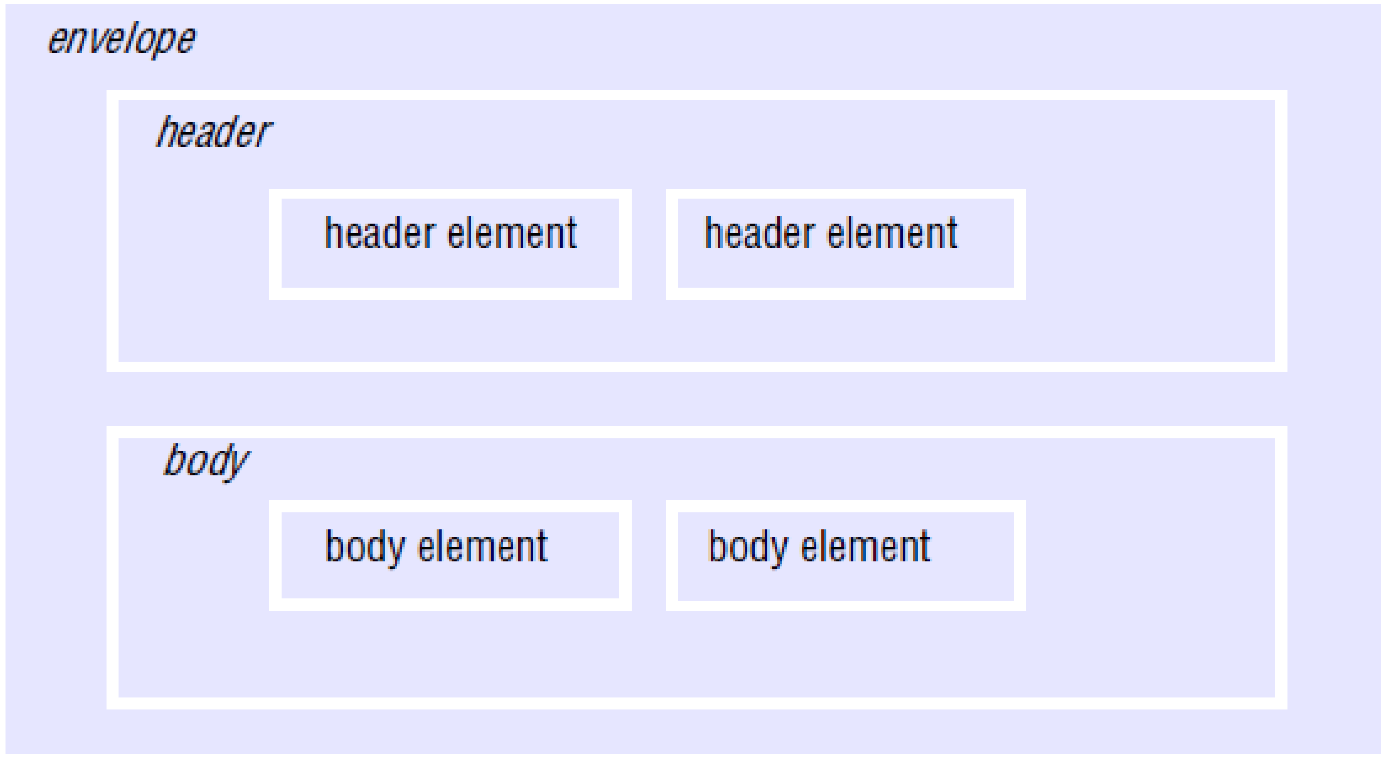
Transparency

- A major task of many middleware platforms is to protect the programmer from the details of data representation and marshalling; another is to make remote invocations look like local ones. None of these things are provided as a part of an infrastructure or middleware platform for web services. At the simplest level, clients and servers may read and write their messages directly in SOAP, using XML.

Service Oriented Architecture Protocol (SOAP)

SOAP

- SOAP is designed to enable both client-server and asynchronous interaction over the Internet. It defines a scheme for using XML to represent the contents of request and reply messages as well as a scheme for the communication of documents. Originally SOAP was based only on HTTP, but the current version is designed to use a variety of transport protocols including SMTP, TCP or UDP.



SOAP message in an envelope

Service Oriented Architecture Protocol (SOAP)

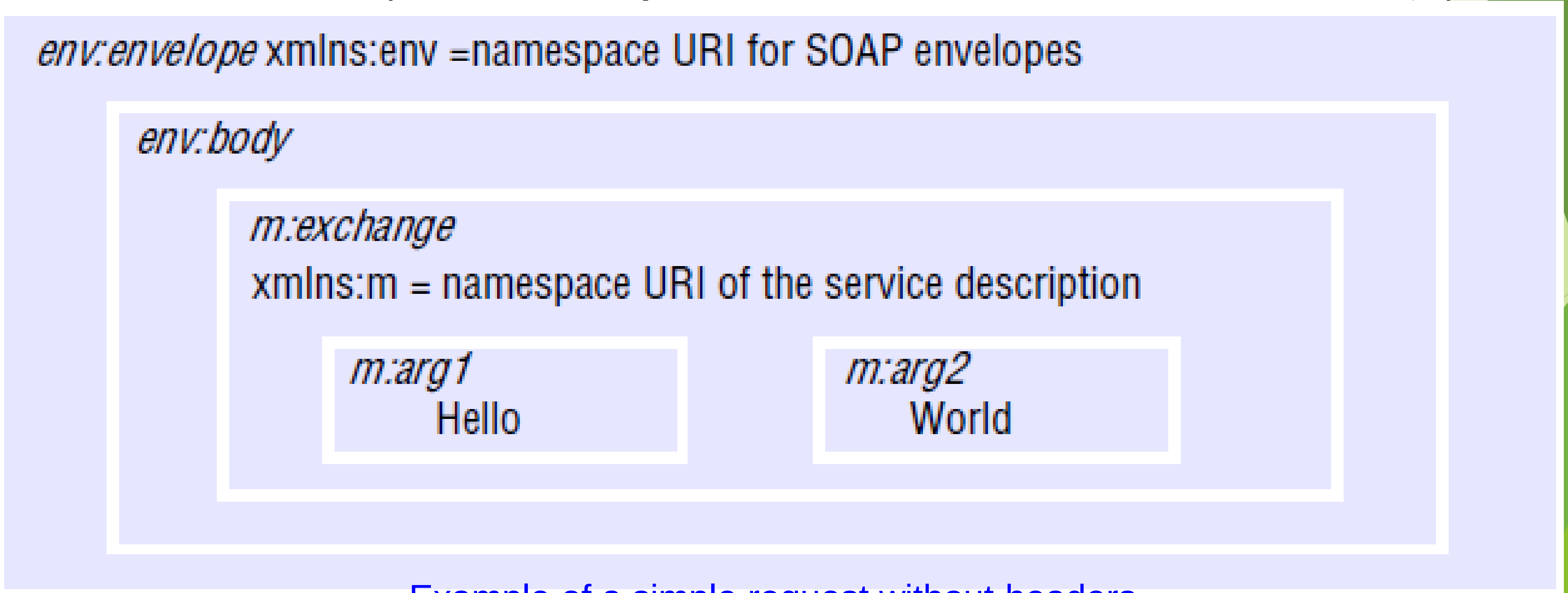
- SOAP APIs have been implemented in many programming languages, including Java, JavaScript, Perl, Python, .NET, C, C++, C# and Visual Basic.
- To support client-server communication, SOAP specifies how to use the HTTP *POST* method for the request message and its response for the reply message. The combined use of XML and HTTP provides a standard protocol for client-server communication over the Internet.

SOAP messages

- SOAP message is carried in an 'envelope'. Inside the envelope there is an optional header and a body. Message headers can be used for establishing the necessary context for a service or for keeping a log or audit of operations. An intermediary may interpret and act on the information in the message headers, for example by adding, altering or removing information. The message body carries an XML document for a particular web service.
- The XML elements *envelope*, *header* and *body*, together with other attributes and elements of SOAP messages, are defined as a schema in the SOAP XML namespace.

Service Oriented Architecture Protocol (SOAP)

- An example of a simple request message without a header. The *body* encloses an element containing the name of the procedure to be called and the URI of the namespace (the file containing the XML schema) for the relevant service description, which is denoted by *m*. The inner elements of a request message contain the arguments of the procedure. This request message provides two strings to be returned in the opposite order by the procedure at the server. The XML namespace denoted by *env* contains the SOAP definitions for an *envelope*.

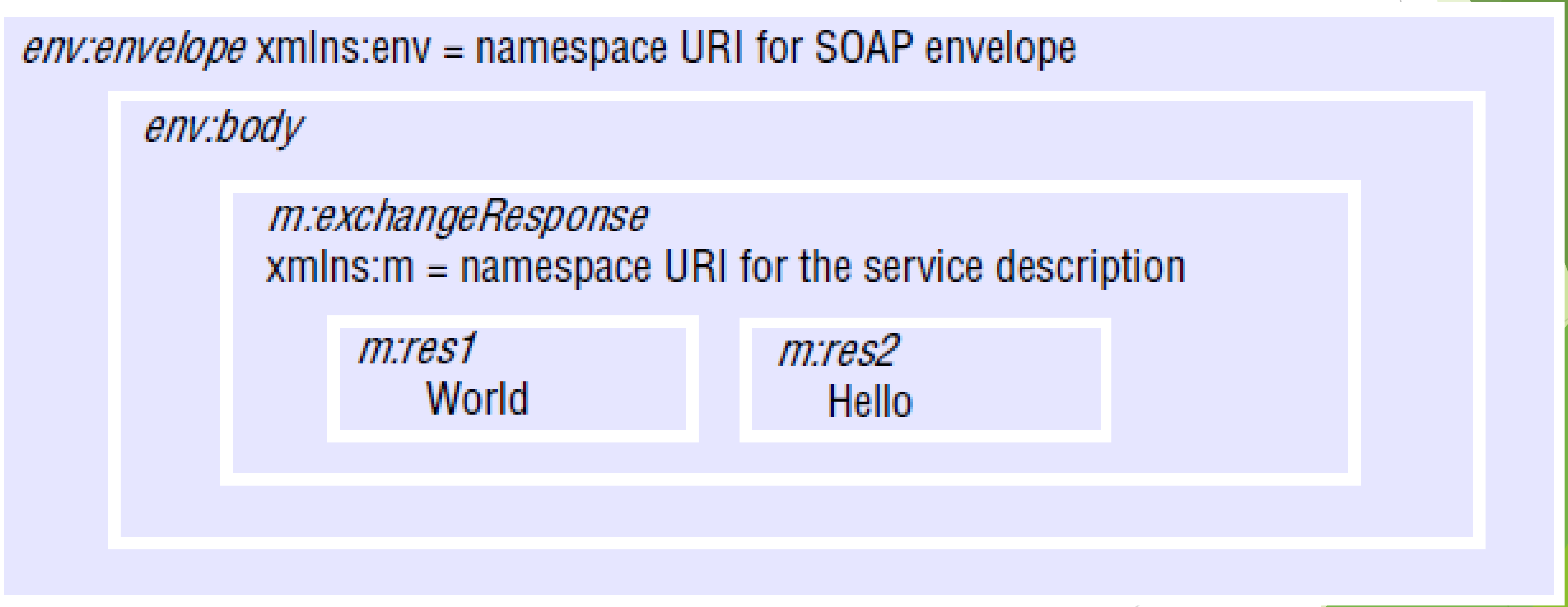


env:envelope xmlns:env =namespace URI for SOAP envelopes
env:body
m:exchange
xmlns:m = namespace URI of the service description
m:arg1
Hello
m:arg2
World

Example of a simple request without headers

Service Oriented Architecture Protocol (SOAP)

- It contains the two output arguments. Note that the name of the procedure has 'Response' added to it. If a procedure has a return value, then it may be denoted as an element called *rpc:result*. The reply message uses the same two XML schemas as the request message, the first defining the SOAP envelope and the second the application-specific procedure and argument names.


env:envelope xmlns:env = namespace URI for SOAP envelope
env:body
m:exchangeResponse
xmlns:m = namespace URI for the service description
m:res1
World
m:res2
Hello

Example of a reply corresponding to the request

Service Oriented Architecture Protocol (SOAP)

SOAP headers

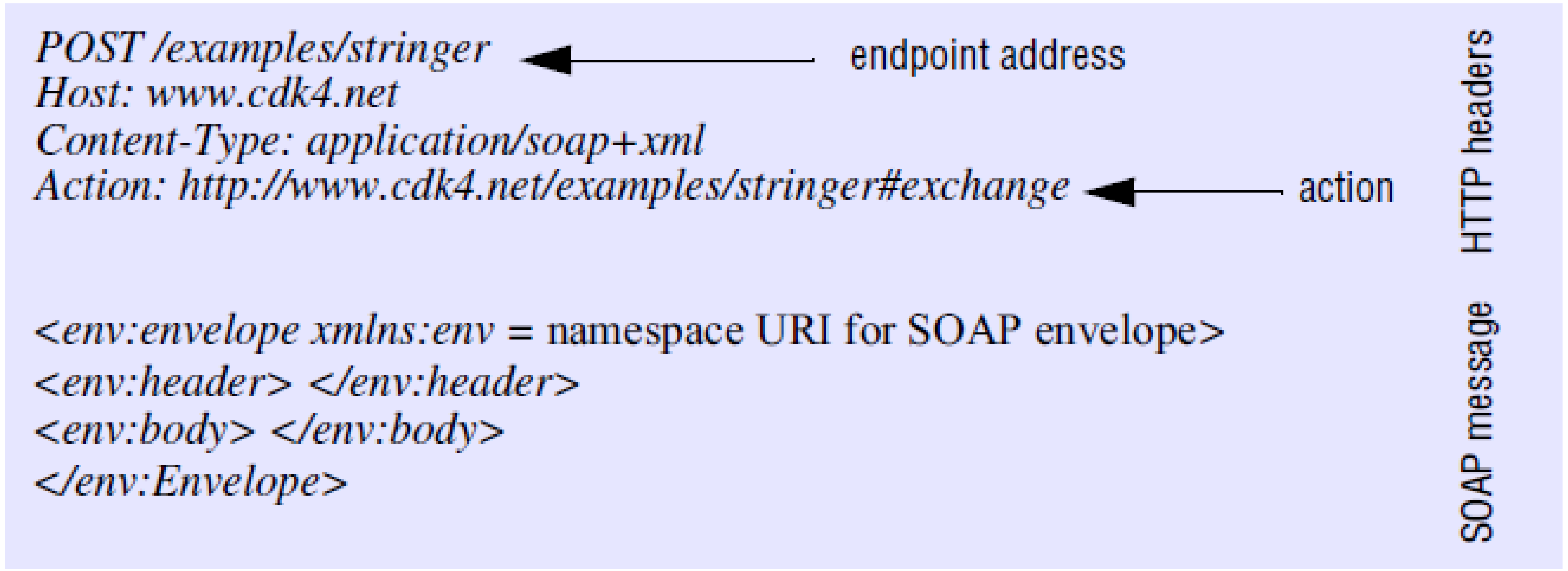
- Message headers are intended to be used by intermediaries to add to the service that deals with the message carried in the corresponding body.

Transport of SOAP messages

- A transport protocol is required to send a SOAP message to its destination. SOAP messages are independent of the type of transport used their envelopes contain no reference to the destination address. HTTP (or whatever protocol is used to transport a SOAP message) is left to specify the destination address.
  - The HTTP headers specify the endpoint address (the URI of the ultimate receiver) and the action to be carried out. The *Action* header is intended to optimize dispatching by revealing the name of the operation without the need to analyze the SOAP message in the body of the HTTP message.
  - The HTTP body carries the SOAP message.

Service Oriented Architecture Protocol (SOAP)

SOAP Request



```
POST /examples/stringer
Host: www.cdk4.net
Content-Type: application/soap+xml
Action: http://www.cdk4.net/examples/stringer#exchange


<env:envelope xmlns:env = namespace URI for SOAP envelope>
<env:header> </env:header>
<env:body> </env:body>
</env:Envelope>
```

REST and Systems of Systems

- **REST:** introduced and explained by Roy Thomas Fielding, one of the principal authors of the HTTP specification, in his doctoral dissertation in 2000.
- is a software architecture style for distributed systems, particularly distributed hypermedia systems, such as the World Wide Web.
- The RESTful web service exposes a set of resources which identify targets of interaction with its clients. The key abstraction of information in REST is a resource.
- Offers simplicity, lightweight nature, and integration with HTTP. With the help of URIs and hyperlinks,
- is not a standard. It is a design and architectural style for largescale distributed systems.

REST and Systems of Systems

Four principles of REST:

- **Resource Identification through URIs:** Any information that can be named can be a resource, such as a document or image or a temporal service. A resource is a conceptual mapping to a set of entities. Each particular resource is identified by a unique name, or more precisely, a Uniform Resource Identifier (URI) which is of type URL, providing a global addressing space for resources involved in an interaction between components as well as facilitating service discovery. The URIs can be bookmarked or exchanged via hyperlinks, providing more readability and the potential for advertisement.

REST and Systems of Systems

- **Uniform, Constrained Interface:** Interaction with RESTful web services is done via the HTTP standard, client/server cacheable protocol. Resources are manipulated using a fixed set of four CRUD (create, read, update, delete) verbs or operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can then be destroyed by using DELETE. GET retrieves the current state of a resource. POST transfers a new state onto a resource.
- **Self-Descriptive Message**: A REST message includes enough information to describe how to process the message. This enables intermediaries to do more with the message without parsing the message contents. In REST, resources are decoupled from their representation so that their content can be accessed in a variety of standard formats (e.g., HTML, XML, MIME, plain text, PDF, JPEG, JSON, etc.). REST provides multiple/alternate representations of each resource. Metadata about the resource is available and can be used for various purposes, such as cache control, transmission error detection, authentication or authorization, and access control.

REST and Systems of Systems

- **Stateless Interactions:** The REST interactions are "stateless" in the sense that the meaning of a message does not depend on the state of the conversation. Stateless communications improve visibility, since a monitoring system does not have to look beyond a single request data field in order to determine the full nature of the request reliability as it facilitates the task of recovering from partial failures, and increases scalability as discarding state between requests allows the server component to quickly free resources. However, stateless interactions may decrease network performance by increasing the repetitive data (per-interaction overhead). Stateful interactions are based on the concept of explicit state transfer.

Restlet

- a lightweight framework, implements REST architectural elements such as resources, representation, connector, and media type for any kind of RESTful system, including web services.
- In the Restlet framework, both the client and the server are components. Components communicate with each other via connectors.

Services and Web Services

web services Technology:

- Web Services Description Language (WSDL):
  - describes the interface, a set of operations supported by a web service in a standard format. It standardizes the representation of input and output parameters of its operations as well as the service's protocol binding, the way in which the messages will be transferred on the wire. Using WSDL enables disparate clients to automatically understand how to interact with a web service.
- Universal Description, Discovery, and Integration (UDDI):
  - UDDI provides a global registry for advertising and discovery of web services, by searching for names, identifiers, categories, or the specification implemented by the web service.

Services and Web Services

Grid Services and OGSA:

- The OGSA, developed within the OGSA Working Group of the Global Grid Forum (recently renamed to Open Grid Forum or OGF and being merged with the Enterprise Grid Alliance or EGA in June 2006), is a service-oriented architecture that aims to define a common, standard, and open architecture for grid-based applications. "Open" refers to both the process to develop standards and the standards themselves. In OGSA, everything from registries, to computational tasks, to data resources is considered a service. These extensible set of services are the building blocks of an OGSA-based grid. OGSA is intended to:
  - Facilitate use and management of resources across distributed, heterogeneous environments.
  - Deliver seamless QoS.
  - Define open, published interfaces in order to provide interoperability of diverse resources.
  - Exploit industry-standard integration technologies.
  - Develop standards that achieve interoperability.
  - Integrate, virtualize, and manage services and resources in a distributed, heterogeneous environment
  - Deliver functionality as loosely coupled, interacting services aligned with industry-accepted web service standards

Services and Web Services

Grid Services and OGSA:

- A grid is built from a small number of standards-based components, called grid services.
- OGSA defines a grid service as "a web service that provides a set of well-defined interfaces, following specific conventions (expressed using WSDL).

GSA services fall into seven broad areas, defined in terms of capabilities frequently required in a grid scenario.

- **Infrastructure Services** Refer to a set of common functionalities, such as naming, typically required by higher level services.
- **Execution Management Services** Concerned with issues such as starting and managing tasks, including placement, provisioning, and life-cycle management. Tasks may range from simple jobs to complex workflows or composite services.
- **Data Management Services** Provide functionality to move data to where it is needed, maintain replicated copies, run queries and updates, and transform data into new formats. These services must handle issues such as data consistency, persistency, and integrity. An OGSA data service is a web service that implements one or more of the base data interfaces to enable access to, and management of, data resources in a distributed environment. The three base interfaces, Data Access, Data Factory, and Data Management, define basic operations for representing, accessing, creating, and managing data.

Services and Web Services

- **Resource Management Services** Provide management capabilities for grid resources: management of the resources themselves, management of the resources as grid components, and management of the OGSA infrastructure. For example, resources can be monitored, reserved, deployed, and configured as needed to meet application QoS requirements. It also requires an information model (semantics) and data model (representation) of the grid resources and services.
- **Security Services** Facilitate the enforcement of security-related policies within a (virtual) organization, and supports safe resource sharing. Authentication, authorization, and integrity assurance are essential functionalities provided by these services.
- **Information Services** Provide efficient production of, and access to, information about the grid and its constituent resources. The term "information" refers to dynamic data or events used for status monitoring; relatively static data used for discovery; and any data that is logged. Troubleshooting is just one of the possible uses for information provided by these services.
- **Self-Management Services** Support service-level attainment for a set of services (or resources), with as much automation as possible, to reduce the costs and complexity of managing the system. These services are essential in addressing the increasing complexity of owning and operating an IT infrastructure.

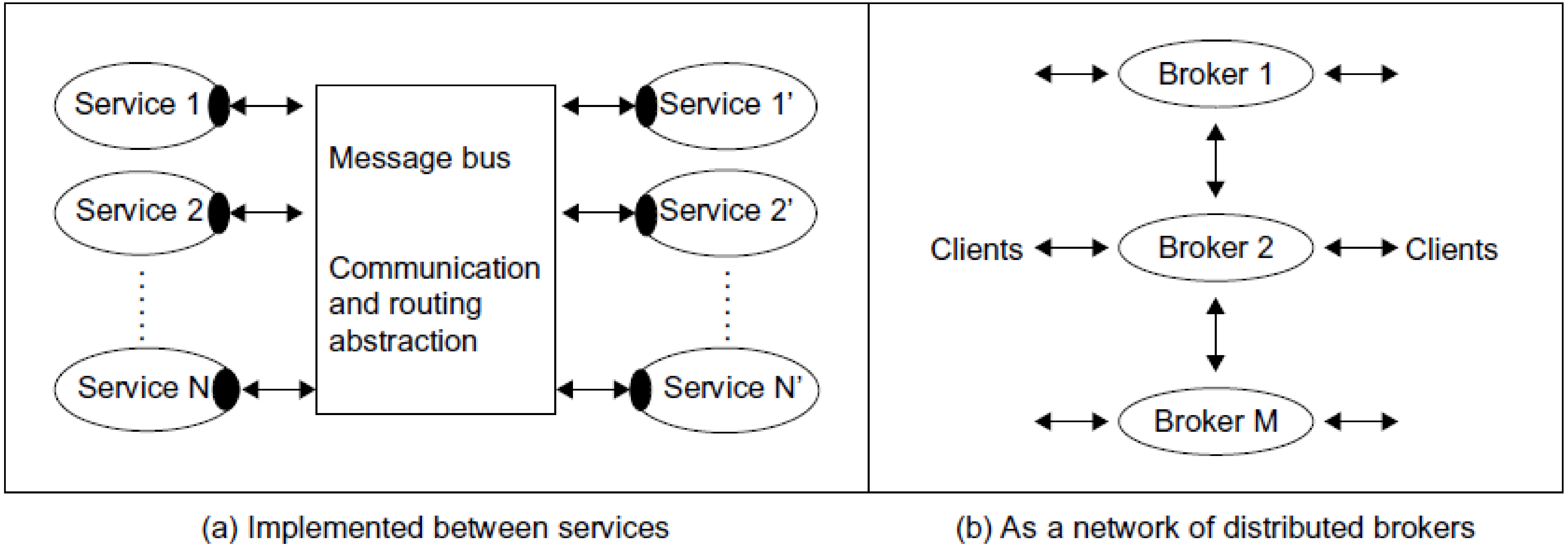
MESSAGE-ORIENTED MIDDLEWARE
Enterprise Bus

Services interact with messages with a variety of different formats (APIs), wire protocols, and transport mechanisms. It is attractive to abstract the communication mechanism so that services can be defined that communicate independent of details of the implementation.

The term “enterprise service bus” or ESB refers to the case where the bus supports the convenient integration of many components, often in different styles.

MESSAGE-ORIENTED MIDDLEWARE
Enterprise Bus

Messaging black box abstraction

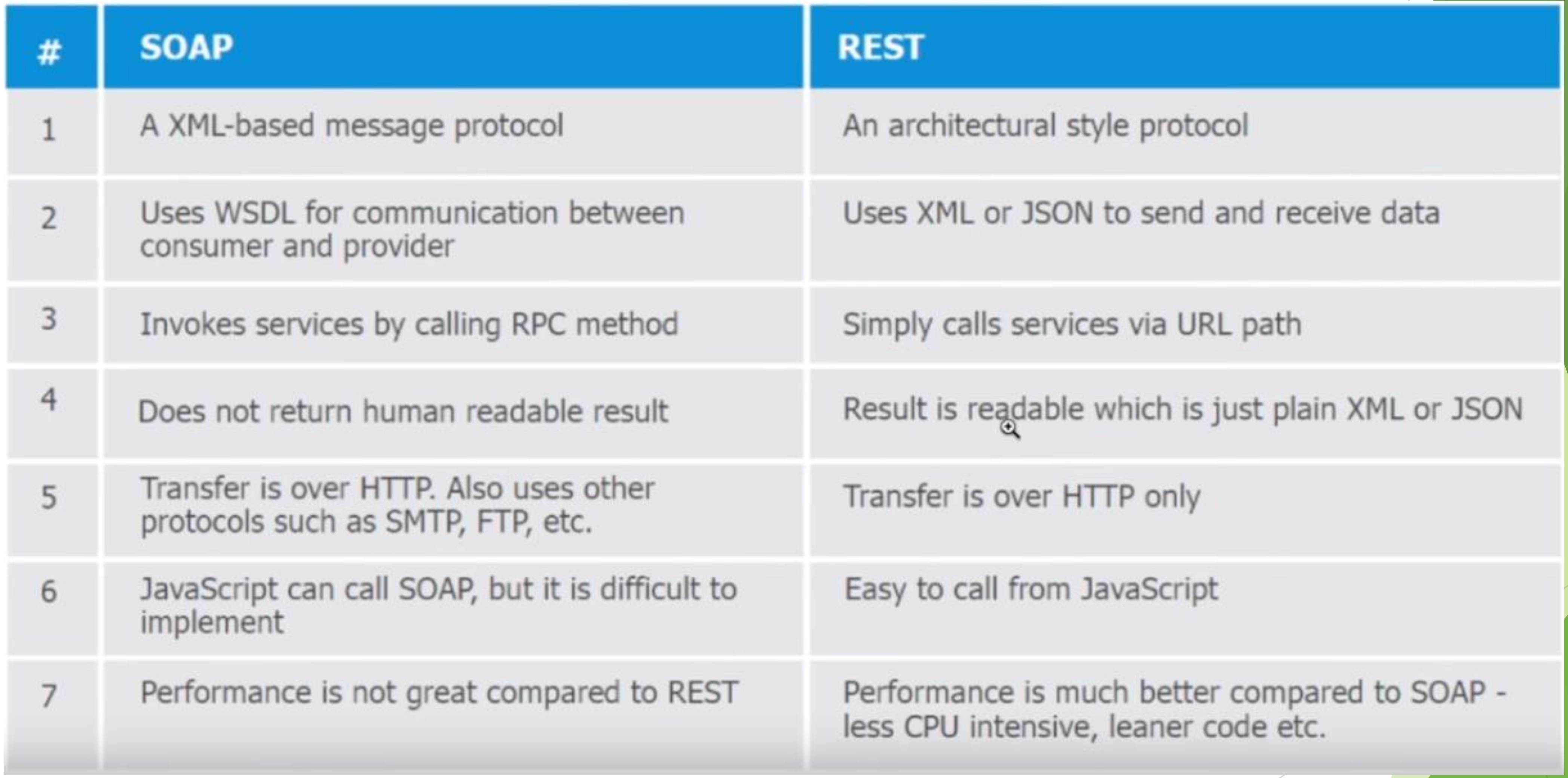

Service 1
Service 2
Service N
Message bus
Communication and routing abstraction
Service 1'
Service 2'
Service N'
(a) Implemented between services
Broker 1
Broker 2
Broker M
Clients
Clients
(b) As a network of distributed brokers

Two message bus implementations between services or using a broker network.

MESSAGE-ORIENTED MIDDLEWARE
Enterprise Bus

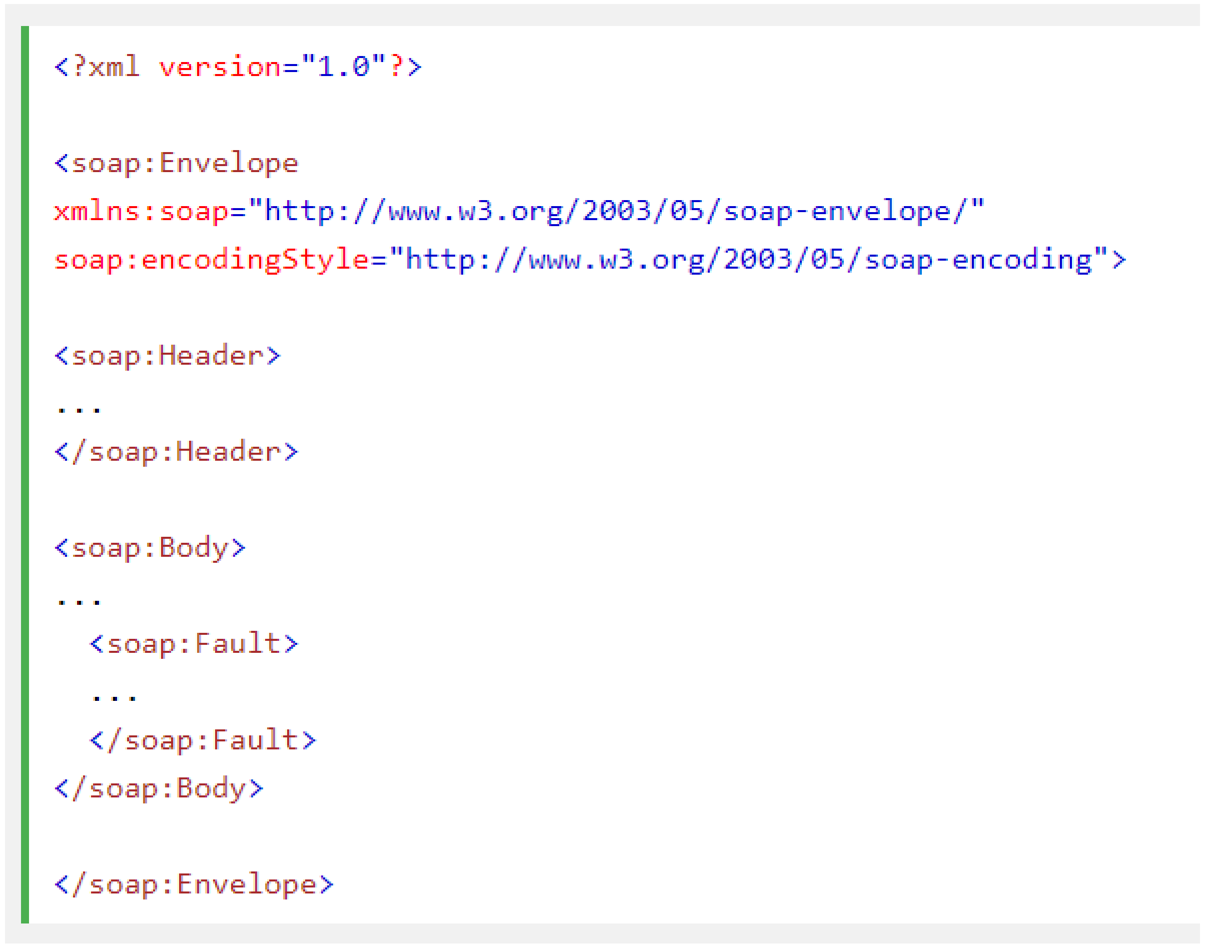
- The brokers is implemented as managers of queues, and software in this area often has MQ or “Message Queue” in its description.
- An early important example of MQ is MQSeries from IBM which is now marketed as the more recent WebSphereMQ. Both Azure and Amazon offer basic queuing software.
- A typical use of a message queue is to relate the master and workers in the “farm” model of parallel computing where the “master” defines separate work items that are placed in a queue which is accessed by multiple workers that select the next available item. This provides a simple dynamically load-balanced parallel execution model. If necessary, the multiple brokers can be used to achieve scalability.

SOAP VS REST



#	SOAP	REST
1	A XML-based message protocol	An architectural style protocol
2	Uses WSDL for communication between consumer and provider	Uses XML or JSON to send and receive data
3	Invokes services by calling RPC method	Simply calls services via URL path
4	Does not return human readable result	Result is readable which is just plain XML or JSON
5	Transfer is over HTTP. Also uses other protocols such as SMTP, FTP, etc.	Transfer is over HTTP only
6	JavaScript can call SOAP, but it is difficult to implement	Easy to call from JavaScript
7	Performance is not great compared to REST	Performance is much better compared to SOAP - less CPU intensive, leaner code etc.

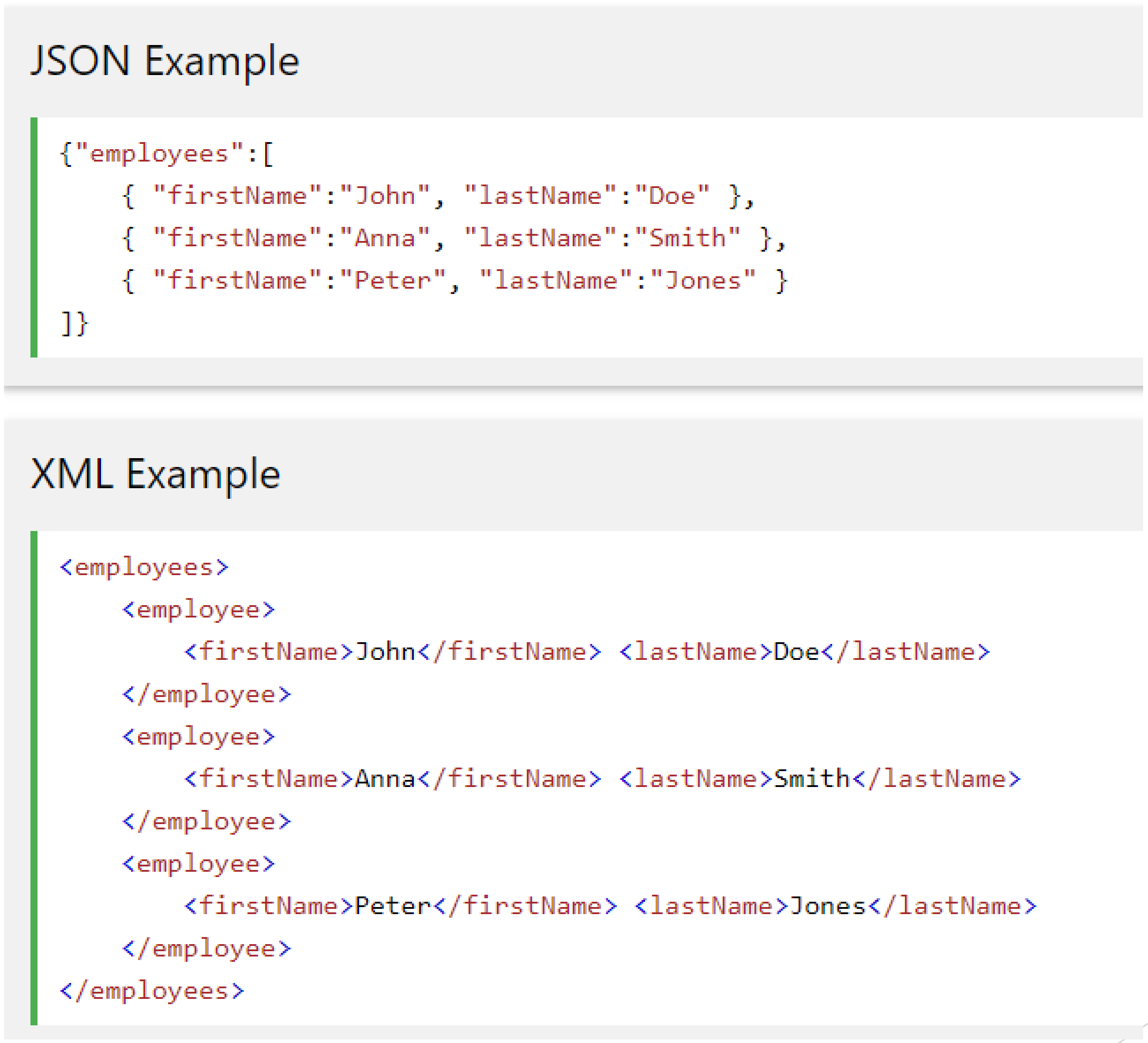
SOAP VS REST



```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope/"
soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

Skeleton SOAP Message

SOAP VS REST



JSON Example

```
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

XML Example

```
<employees>
    <employee>
        <firstName>John</firstName> <lastName>Doe</lastName>
    </employee>
    <employee>
        <firstName>Anna</firstName> <lastName>Smith</lastName>
    </employee>
    <employee>
        <firstName>Peter</firstName> <lastName>Jones</lastName>
    </employee>
</employees>
```

JSON VS XML Message

SOAP VS REST

Advantage SOAP

- Language, Platform & Transparent Independent.
- Distributed enterprise environment.
- Standardized.
- Pre-built extensibility.
- Build in error handling.
- Automation.

Disadvantage

- Heavy

SOAP VS REST

Advantage of Rest

- No expensive tools.
- Smaller learning Curve.
- Efficient.
- Fast.
- Closer to other web technologies.