Web Services
Semantic Web

Overview

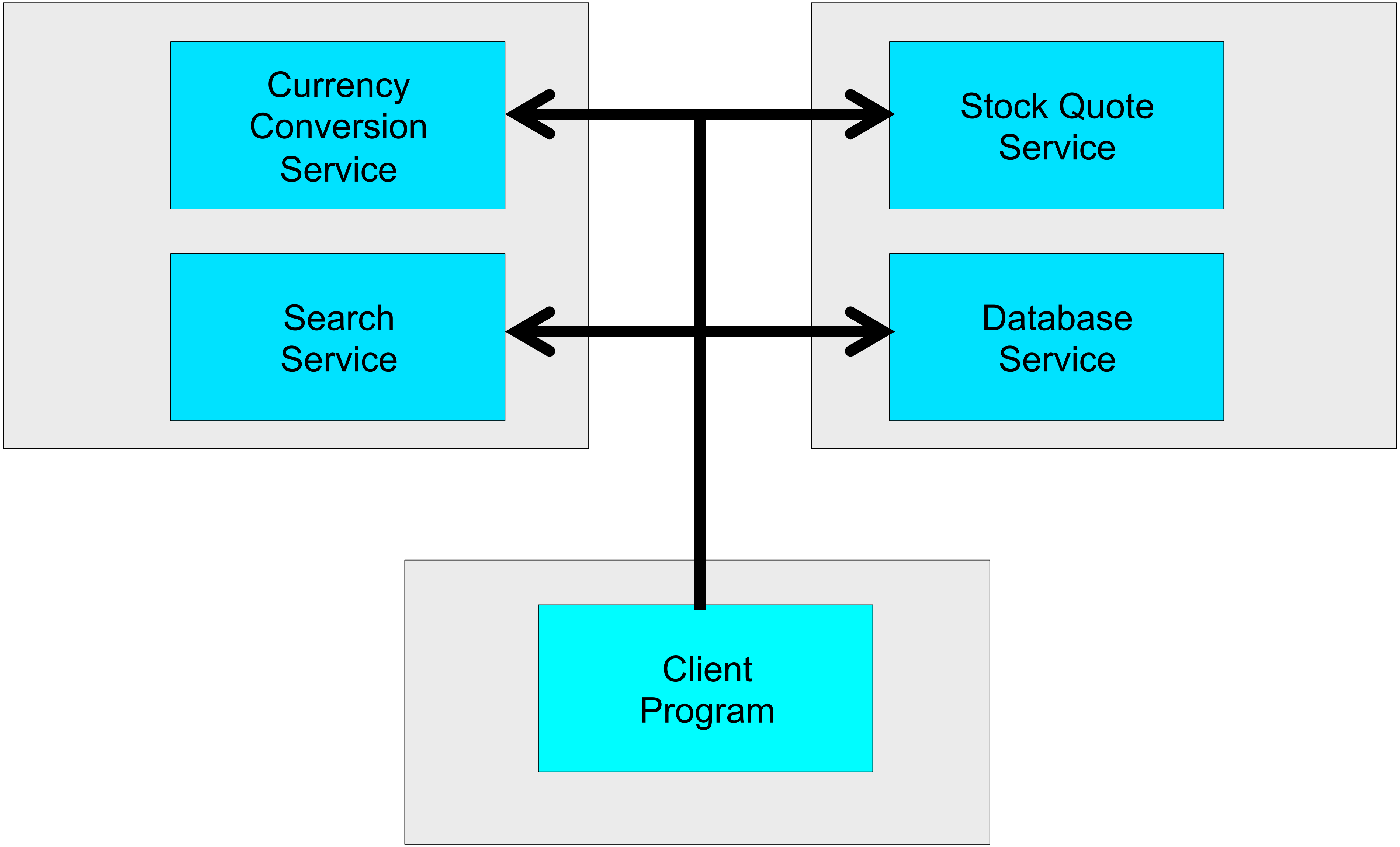
- Web services
  - SOAP
  - WSDL

- Semantic web
  - RDF
  - Ontologies
  - SPARQL
  - RDFa

Service-oriented computing

- Applies principles behind internet and web to software functionality:
  - Decentralised
  - Distributed administration
  - Protocols remain constant even if servers change
  - Allows companies to provide, maintain and update proprietary software on their own sites
- Multiple services can be combined to provide a greater product
- Extension of object / component principles

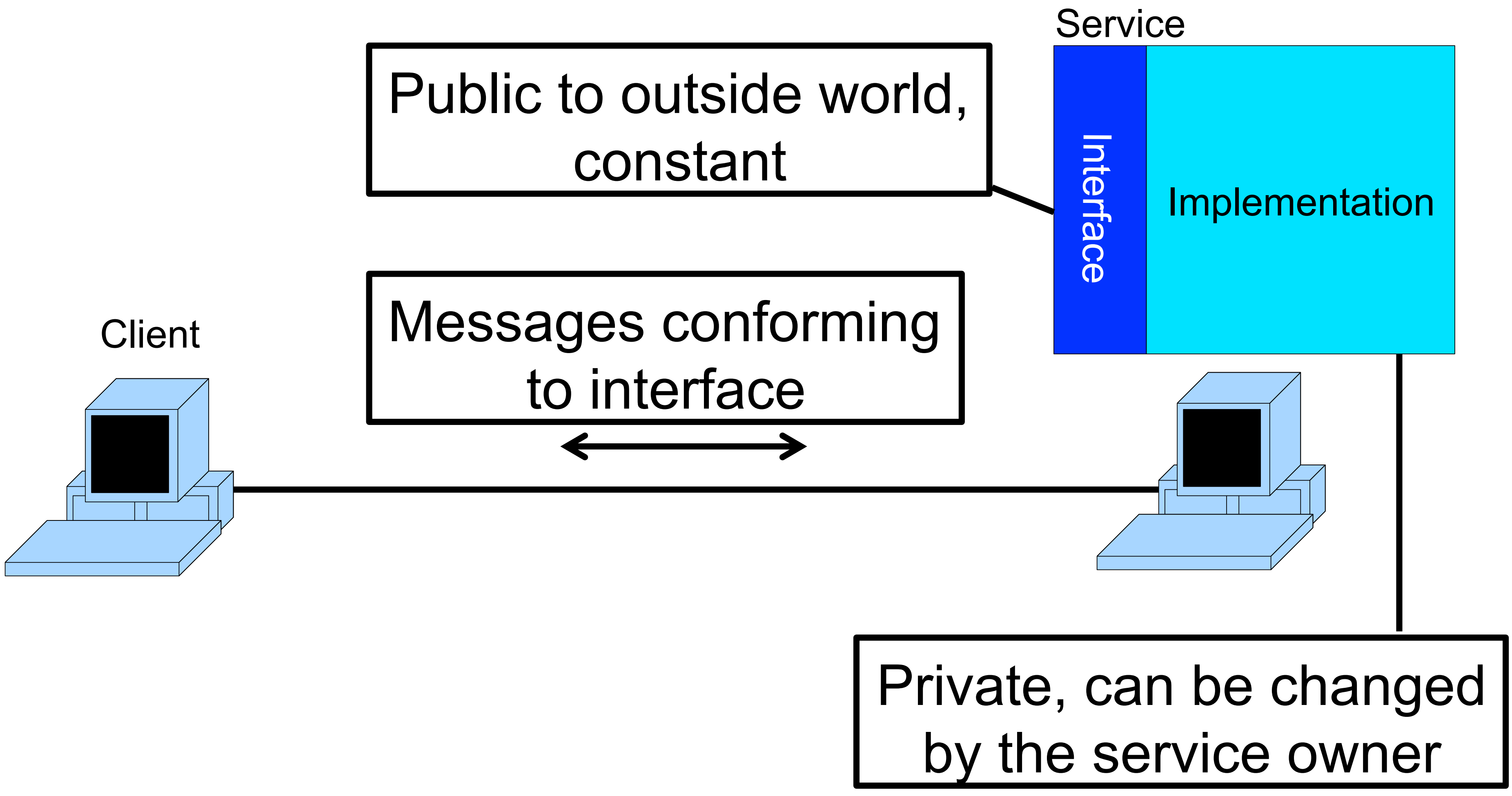
Services


Currency Conversion Service
Stock Quote Service
Search Service
Database Service
Client Program

Interfaces

- Each service states the protocols it supports
- The protocols can be specific to its business function
- The description of the protocols supported by a service is called its interface
- For example, a service providing stock quotes for named companies on demand supports a protocol for asking for stock quotes
- Multiple services providing interchangeable functionality can use same interface definition

Services

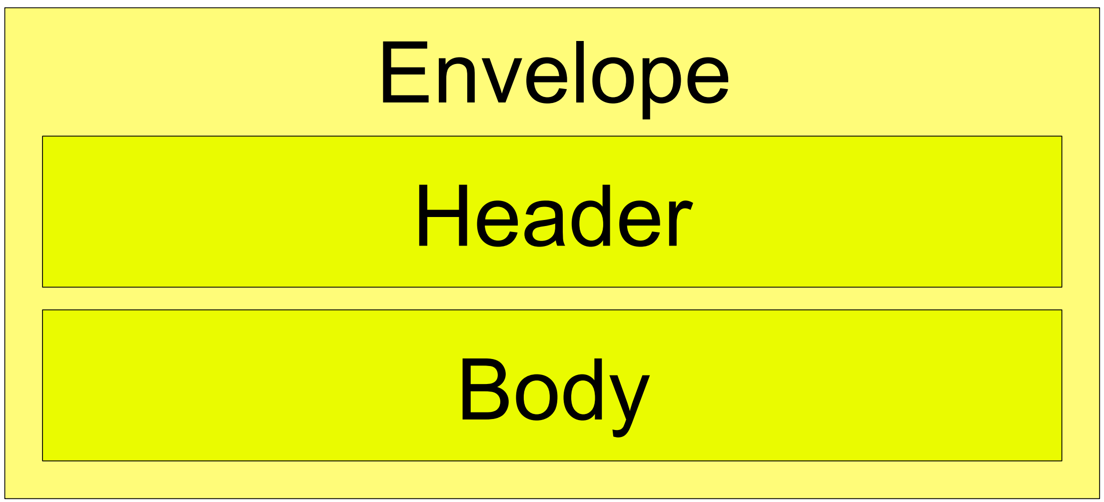

Service
Public to outside world, constant
Interface
Implementation
Client
Messages conforming to interface
Private, can be changed by the service owner

Web services

- Web Services are services deployed using internet and web technology
  - Communication usually over HTTP (over TCP/IP)
  - Languages for interfaces and communication are in XML
- Many Web Service related concepts are defined in the Web Services Architecture
- However, key technologies are:
  - SOAP: XML-based communication protocol
  - WSDL: XML-based interface definition language

Simple Object Access Protocol (SOAP)

- SOAP is the Web Services communication protocol
- SOAP messages have a common structure
- Outer envelope containing header then body
- Expressed as XML element hierarchy



SOAP message structure

- The body contains a message conforming to the definition of the service's interface
- The header contains information about the communication, message body, sender etc.
  - Authentication, authorisation information
  - Transaction context
  - Resource to which message addressed
- Both are pieces of XML (SOAP XML schema allows any XML content in each)

SOAP example

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




  </soap:Header>
  <soap:Body>




  </soap:Body>
</soap:Envelope>
```

SOAP example

```
<soap:Envelope xmlns:soap =
     "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <t:Transaction xmlns:t="http://www.example.com"
                   soap:mustUnderstand = "1">
      5
    </t:Transaction>
  </soap:Header>
  <soap:Body>




  </soap:Body>
</soap:Envelope>
```

SOAP example

```
<soap:Envelope xmlns:soap =
     "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <t:Transaction xmlns:t="http://www.example.com"
                   soap:mustUnderstand = "1">
      5
    </t:Transaction>
  </soap:Header>
  <soap:Body>



  </soap:Body>
</soap:Envelope>
```

Parser should reject this message if it does not know how to interpret t:Transaction

SOAP example

```
<soap:Envelope xmlns:soap =
     "http://www.w3.org/2003/05/soap-envelope">
  <soap:Header>
    <t:Transaction xmlns:t="http://www.example.com"
                   soap:mustUnderstand = "1">
      5
    </t:Transaction>
  </soap:Header>
  <soap:Body>
    <m:GetLastTradePrice xmlns:m =
                "http://example.com/stockquote">
      <m:symbol>DEF</m:symbol>
    </m:GetLastTradePrice>
  </soap:Body>
</soap:Envelope>
```

SOAP over HTTP

- SOAP is commonly sent over HTTP
- The SOAP envelope is the entity body in the HTTP request/response

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset=utf-8

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

SOAP actions

- SOAP allows the "intent" of the SOAP message to be specified as a URI in the HTTP Content-Type field
- It is extra information about how to process the message

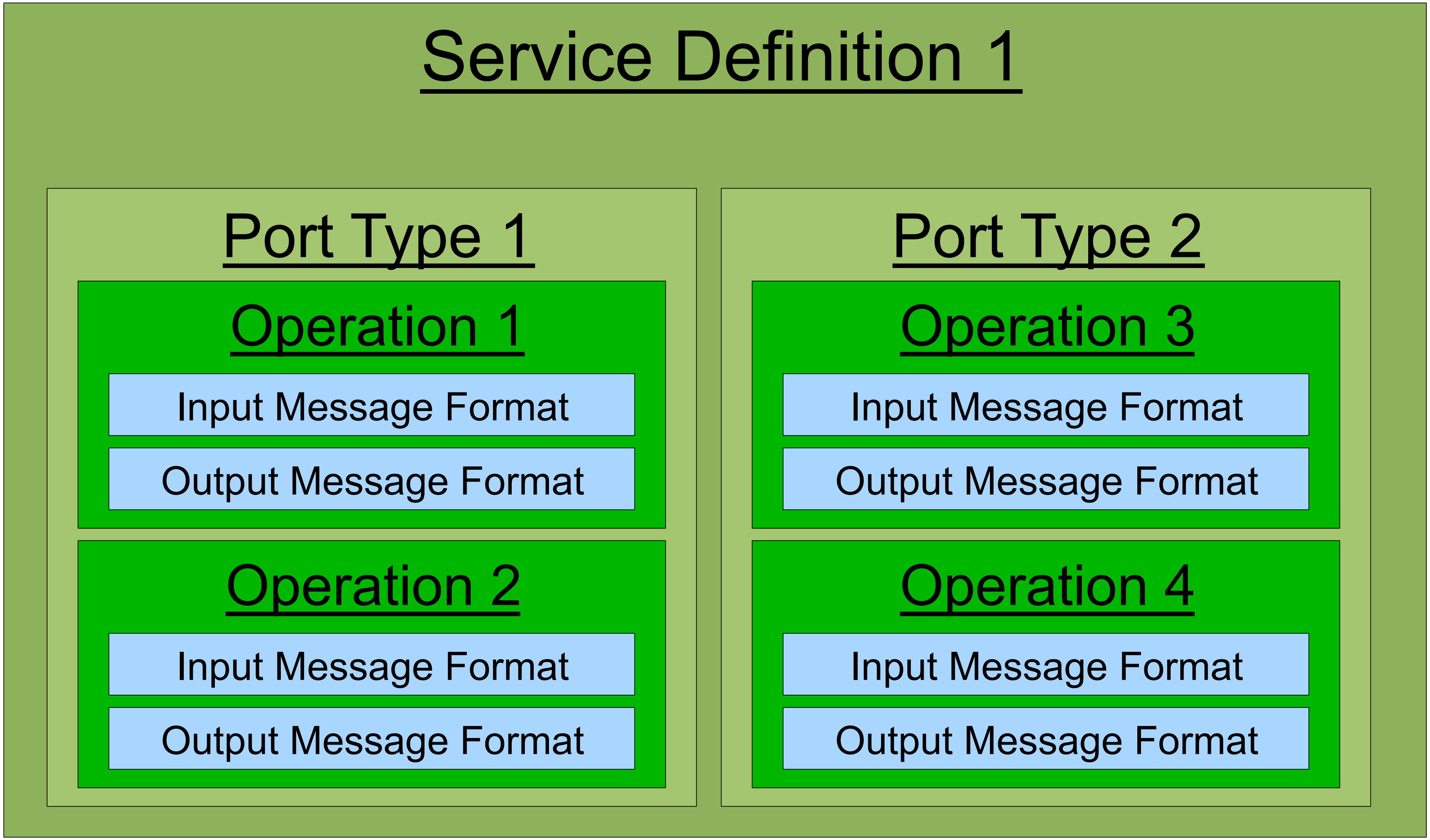
```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap; charset=utf-8;
  SOAPAction="http://www.example.com/
  GetLastTradePrice"

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

Web Services Definition Language (WSDL)

- WSDL is an XML language for specifying the form of messages a service understands or produces
- Specifies rules on form of SOAP body, i.e. a schema written in XML Schema
- A service interface is split into **port types**
- Each port type contains a set of **input** and **output message** definitions
- An **operation** combines an input message and output message to say:
  - This form of output is returned in response to this form of input

WSDL interface



Port types

- Port types are like interfaces in OOP languages
- A port type groups a set of operations of a particular kind
- For example, a registry service might have
  - A **publish** port type containing multiple registration-related operations
  - An **inquiry** port type containing multiple search operations taking different criteria
- A port type specifies which messages can be received and produced by one port (more on ports later)

Operations

- An operation is something that can be performed on a service, like a method in an OOP language
- An operation consists of an input request message and an output response message
- Web service operations are assumed to be **asynchronous**, so the response may be received any time after the request

Operation, port type example

```
<portType name="StockQuotePortType">
 <operation name="GetLastTradePrice">
  <input
      message="tns:GetLastTradePriceInput"/>
  <output
     message="tns:GetLastTradePriceOutput"/>
 </operation>
</portType>
```

Messages

- Messages are XML documents (the contents of SOAP bodies)
- They must conform to a schema, so that the service and client can know the expected form of the request and response, and interpret the messages correctly
- One message could be the response for multiple operations, e.g.
  - ListAllEntries request returns ServiceList response
  - SearchByName request also returns ServiceList response

Message example

```
<schema>
 <element name = "PriceRequest">
  <complexType>
   <all>
    <element name="symbol" type=″string"/>
   </all>
  </complexType>
 </element>
</schema>

<message name = "GetLastTradePriceInput">
 <part name="body" element="PriceRequest"/>
</message>
```

WSDL interface documents

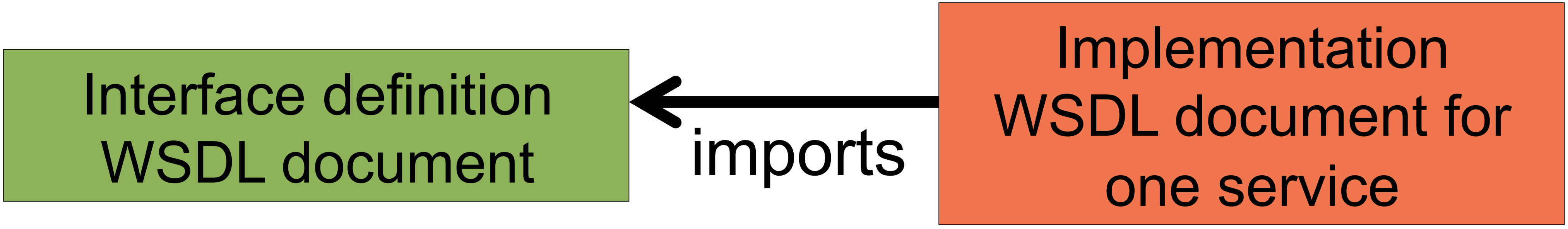
- As a whole, a WSDL interface document consists of multiple messages, port types and operations
- Port types, messages and operations define the interface of a service, separate from any deployed service
- The interface can be shared by many (interchangeable) Web Services, e.g.
  - Different companies providing the same kind of functionality with different qualities
  - Back-up services for fault tolerance

WSDL root node example

```
<definitions name = "StockQuote”
  targetNamespace = "http://example.com/stockquote"
  xmlns = http://schemas.xmlsoap.org/wsdl>
…
…
…
</definitions>
```

Implementation WSDL

- WSDL is also used to give details of how to use an abstract interface with a given service
- Implementation details include:
  - URL of the service's web server
  - Underlying protocol to use (HTTP)
- While both the abstract definition and specific implementation details can be in one file, they are often split into two, so abstract interface can be re-used / shared



Bindings

- A **binding** describes a concrete binding of a port type component and associated operations to a particular concrete message format and transmission protocol
- One binding may specify the use of SOAP, for example, while another may specify the use of Java RMI
- Within a binding, further transport and encoding information is provided for each message of each operation of the port type

Binding example

```
<binding name="StockQuoteSoapBinding”
         type="tns:StockQuotePortType“>
  <wsoap:binding style=“document”
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
      <wsoap:operation
       soapAction="http://example.com/GetLastTradePrice"/>
      <input>
            <wsoap:body use="literal"/>
      </input>
      <output>
            <wsoap:body use="literal"/>
      </output>
  </operation>
</binding>
```

Binding example

```
<binding name="StockQuoteSoapBinding”
         type="tns:StockQuotePortType“>
  <wsoap:binding style=“document”
        transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="GetLastTradePrice">
      <wsoap:operation
       soapAction="http://example.com/GetLastTradePrice"/>
      <input>
            <wsoap:body use="literal"/>
      </input>
      <output>
            <wsoap:body use="literal"/>
      </output>
  </operation>
</binding>
```

Ports

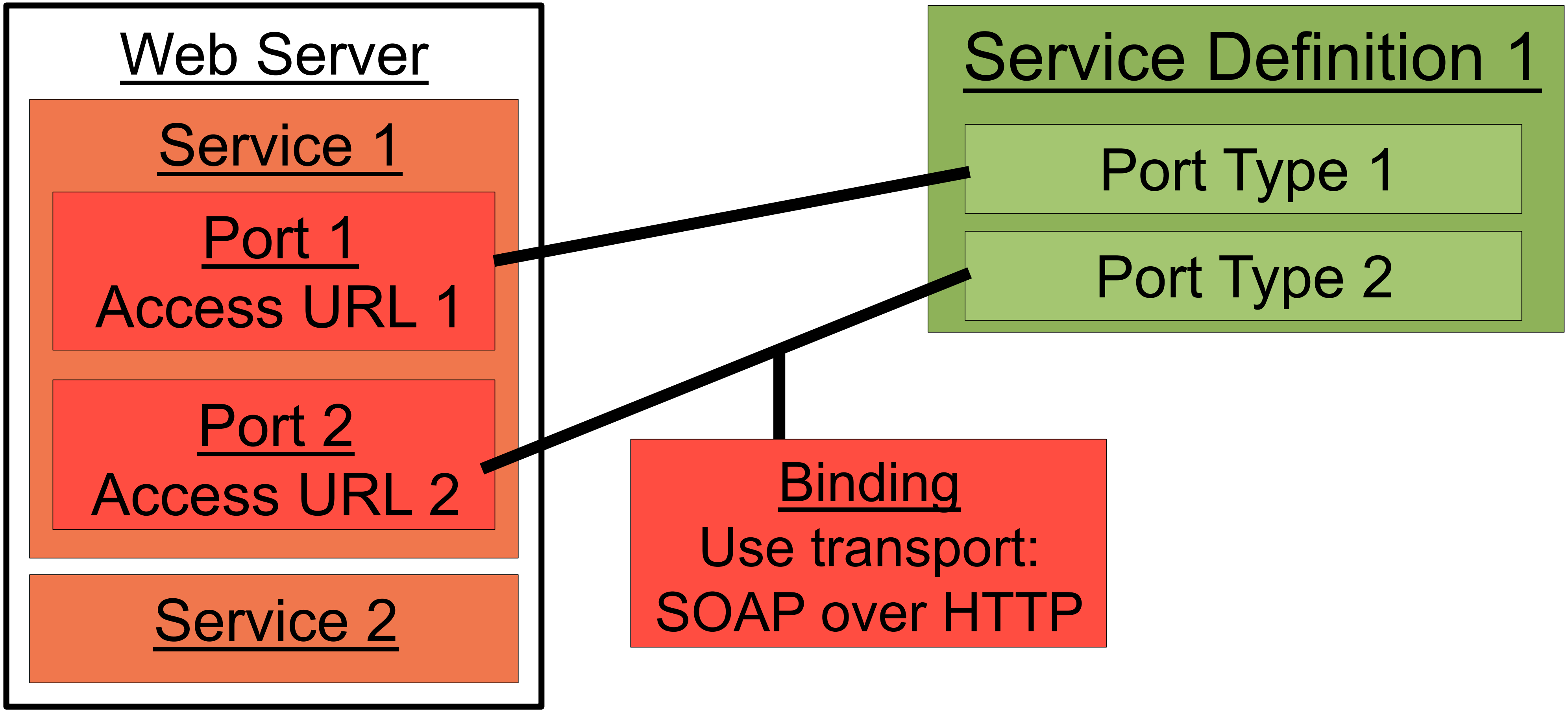
- A **port** of a web service is a similar idea to a TCP port of a host
- That is, it is one channel of communication to which messages of a particular purpose can be sent to and received from
- Each port has a different URL and a binding
- Clients send messages to the URL that conform to the message schemas of the binding's port type and to the binding's transport and encoding details

Services

- A WSDL **service** is a collection of ports
- It ties together all the other parts of the interface into one, named, whole definition of a web service

```
<service name = "StockQuoteService">
 <port name = "StockQuotePort"
       binding = "tns:StockQuoteBinding">
  <soap:address location="http://x.com/sq"/>
 </port>
</service>
```

WSDL interfaces and implementations



Universal Description Discovery and Integration (UDDI)

- To use web services you first have to find them, and the aim has always been for services to be widely re-used
- UDDI is a directory service specification that was taken as de-facto standard for discovering web services
- UDDI is itself a web service: it has WSDL-defined port types for publishing descriptions of services and for discovering services
- A service description can contain information on owners of the service, the function it performs, its WSDL interface and more

The Semantic Web

- The web contains a wealth of information
- While it is written in a way that is helpful for people, it is not in a form that softwares can parse and use
- If softwares could search for and use the information on the web, it could potentially be a lot more useful, in the same way that a person can be more productive by using the web
- The idea of the **semantic web** is to include computer-readable information on the web alongside the current human-readable information

Resource Description Framework

- Several technologies are required to make the semantic web work
- The first thing required is a data structure in which to make computer-readable statements
- The structure used is the **Resource Description Framework (RDF)**

Statements

- An RDF document is a set of **statements**
- A statement asserts something about a resource, sometimes its relation to another resource
- Every statement consists of three parts:
  - The **subject** of the statement: what resource the statement is about, e.g. http://www.example.org/index.html
  - The **object** of the statement: what resource or value the subject is related to, e.g. Samhar Mahmoud
  - The **predicate** of the statement: how the subject and object are related, e.g. creator
- We can write an RDF statement in the form:

subject predicate object .

Resources

- The subjects of RDF statements, and sometimes the objects are **resources**
- A resource is something identifiable by a URI, e.g.
  - A webpage or other web-accessible document or service
  - A physical thing, such as a person, organisation or book
  - An abstract concept, e.g. being happy or the number pi
- URIs are also used to give unique identifiers for the predicates of RDF statements, e.g. creator
- We write a URI in a statement between <…> brackets:

```
<http://www.example.org/index.html>
        <http://purl.org/dc/terms/creator>
                <http://www.inf.kcl.ac.uk/staff/samharm> .
```

Vocabularies

- A **vocabulary** is a set of terms defined together to allow descriptions in some particular domain, similar to namespaces
- Each term is a URI, and all the URIs in a vocabulary start with the same string
- For example, the vocabulary **http://purl.org/dc/terms/** describes who created and published documents, at what times, and similar library data, including terms such as:
  - http://purl.org/dc/terms/**creator** – relates a document to its creator
  - http://purl.org/dc/terms/**publisher** – relates a document to its publisher
  - http://purl.org/dc/terms/**isReplacedBy** – relates an older edition of a document to a newer edition

Turtle and prefixes

- RDF statements can be encoded in different formats, including XML
- The format we are using here is called **Turtle**
- URIs are long to write, so we will often abbreviate them using prefixes, where the prefix replaces the vocabulary URI
  - **dc:creator** means the URI that dc: maps to combined with "creator"
- For example, we may say:
  - Prefix **ex:** is mapped to **http://www.example.org/**
  - Prefix **dc:** is mapped to **http://purl.org/dc/terms/**
  - Prefix **inf:** is mapped to **http://www.inf.kcl.ac.uk/staff/**
- The previous Turtle RDF statement then becomes:

```
ex:index.html   dc:creator   inf:samharm .
```

- Prefixes are declared at the start of the Turtle document, e,g.

```
#prefix ex: <http://www.example.org/> .
```

Values

- The objects of RDF statements do not have to be URIs, but can be data values (strings, integers, etc.) instead
- For example, the following says that the first name of the resource **inf:samharm** (a person) is "Samhar"

inf:samharm foaf:firstName "Samhar" .

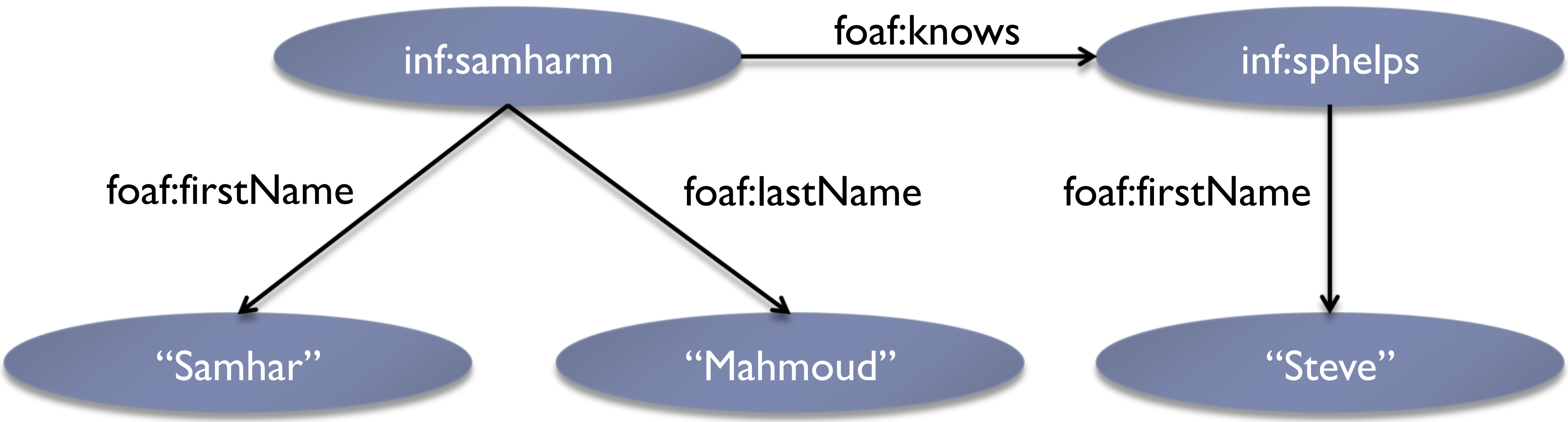
- An RDF document then consists of many statements, some of which may be about the same resource

#prefix foaf: <http://xmlns.com/foaf/0.1/> .
#prefix inf: <http://www.inf.kcl.ac.uk/staff/> .

inf:samharm foaf:firstName "Samhar" .
inf:samharm foaf:lastName "Mahmoud" .
inf:samharm foaf:knows inf:sphelps .

RDF graphs

- A set of RDF statements is often called an **RDF graph**, because the information forms a graph with the resources and values as nodes and the predicates as edges



Ontologies

- RDF allows us to make statements about resources that can be read by software
- It does not, by itself, allow the software to "reason" about the statements to determine how best to apply the information
- To allow this, we need to encode something about the meaning of the resources, such as what kind of thing a particular resource is and what is known about resources of that general type
- Data which encodes such meaning is called an **ontology**

Web Ontology Language

- The Web Ontology Language (OWL) is a language for encoding ontologies in RDF
- An OWL ontology defines a vocabulary of terms, but also says how those terms relate to each other, so as to give extra meaning for software to reason over using components called **reasoners**
- OWL is itself a vocabulary for defining the meaning of terms

Classes and individuals

- The first kind of statement OWL allows us to make is to say what **class** a resource belongs to, i.e. what kind of thing it is, using the predicate rdf:type
- For example, the statement below says that inf:simonm is a kind of person, i.e. an instance of the class ex:Person

inf:samharm **rdf:type** ex:Person .

- The URI ex:Person is a term in our ontology, representing the class of all people
- In the statement above, inf:samharm is said to be an **individual**, because it is a specific thing in the world
- rdf:type is so common it is abbreviated as **a**

inf:samharm **a** ex:Person .

Multiple classes

- A resource can be an instance of multiple classes
- inf:samharm is not only a person, but a lecturer and a man

```
inf:samharm  a  ex:Person ;
             a  ex:Lecturer ;
             a  ex:Man .
```

- However, we would do not want to have to state that every resource that is a man (or woman) is also a person
- Instead, our ontology can say that ex:Man is a subclass of ex:Person, so reasoners can automatically determine that any resource that is a man is also a person

```
ex:Man  rdfs:subclassOf  ex:Person .
```

Class hierarchies

- Ontologies generally include hierarchies of subclass relationships between many class terms

```
ex:Man      rdfs:subclassOf   ex:Person .
ex:Woman    rdfs:subclassOf   ex:Person .
ex:Mother   rdfs:subclassOf   ex:Woman .
```

- From the above subclass relations, a reasoner could determine that any instance of the class ex:Mother is also an instance of the class ex:Person, e.g.

```
inf:mary   a   ex:Mother .
```

implies

```
inf:mary   a   ex:Woman, ex:Person .
```

Properties

- Consider an RDF statement about OWL individuals

```
inf:samharm  ex:worksIn  ex:London .
```

- In OWL, the predicate above, ex:worksIn, is called a **property**
- We can say more about a property's meaning using OWL
- We can say that ex:worksIn only makes sense if it is relating a person (the **domain** of the property) to a city (the **range** of the property)

```
ex:worksIn  rdfs:domain  ex:Person ;
            rdfs:range   ex:City .
```

Datatypes

- We saw that, in RDF, the objects of statements can be values that are not resources: strings, integers etc.

```
ex:Mary  ex:hasName  “Mary” ;
         ex:hasAge    51 .
```

- To define the range of one of these properties, we cannot refer to a class like ex:Person or ex:City
- Instead the range of these properties are **datatypes**, such as string or integer
- OWL uses XML Schema data types to state the range in these cases, i.e. xs:string, xs:integer, xs:dateTime etc.

```
ex:hasName   rdfs:range    xs:string .
ex:hasAge    rdfs:range    xs:integer .
```

Social methodology

- Getting people to agree on even a small ontology is difficult
- The more people who need to agree, the harder it is
- If ontologies are imposed, people will not agree and so not use them
- The social approach of the semantic web:
  - Small groups agree on small ontologies
  - Mappings are created between ontologies to combine them into a larger ontology

Mapping ontologies

- OWL provides vocabulary to help map between two different ontologies
- We can say that one class is equivalent to another class, so any instance of one is an instance of the other

  my:Person **owl:equivalentClass** your:Human .

- where my: and your: are two different ontologies
- We can similarly say that one individual is the same as another individual, just identified with a different URI

  inf:samharm **owl:sameAs** ex:samharMahmoud .

SPARQL

- To extract knowledge from stores of RDF data (**triple stores**), we need to query them, as with any other database
- The standard query language for RDF is SPARQL
- SPARQL is an SQL-like query language for RDF
- As SPARQL is intended for use on the web, there is an accompanying protocol, the SPARQL Protocol, for sending queries to online triple stores and returning the query results

SPARQL queries

- A basic SPARQL query finds all the statements, or combinations of statements, following a particular pattern, and returns some subjects and/or objects of those statements
- For example, we may want to
  - retrieve the email address… (the object of statements with foaf:mbox as predicate)
  - …of everyone that inf:samharm knows (statements following the pattern: inf:samharm foaf:knows ???)

Pattern variables

- We require variables to represent the parts of the data we are looking to retrieve
- If we are querying for statements matching the pattern "any resource for which we have a name and an email address", we need three variables for the resource, the name and the email address
- We use the form ?var or $var for a variable name
- The pattern above could be expressed in SPARQL as:

```
{ ?x foaf:name ?name .
  ?x foaf:mbox ?mbox }
```

Example SPARQL query

- We then need to say which variables we are interested in returning as query results using a SELECT statement
- The query below returns the name and email address for any resource (person) for which we have that information

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
      { ?x  foaf:name  ?name .
        ?x  foaf:mbox  ?mbox  }
```

Query results

- The result of a query can be seen as a table, with the column headings as variable names and cells as values
- Each row is one set of bindings for the variables that when placed into the query pattern gives some RDF found in the triple store

?name	?mbox
Samhar Mahmoud	samhar.mahmoud@kcl.ac.uk
Steve Phelps	Steve.phelps@kcl.ac.uk

Semantic web pages

- While RDF can be stored in triple stores, the original intention of the semantic web was to provide the machine-readable knowledge **alongside** the human-readable web
- This means, to realise the semantic web, we need to specify how RDF can be embedded inside HTML pages using mark-up
- The RDF data is not presented to the user, but can be extracted from the webpage by software, e.g.
  - A party announced on a blog could be copied to the user's calendar, with the author's contact information added to the user's address book
  - Users could automatically recall previously browsed articles according to categorisation labels (tags)

RDFa

- RDFa allows RDF to be embedded in HTML files
- If we add a **property** attribute to an element marking up text, the attribute value is a predicate relating the webpage to the text
- If http://www.example.org/index.html contains:

```
<h2 property="http://purl.org/dc/terms/title">My first
story</h2>
```

- Then the following RDF statement is embedded in the page:

```
<http://www.example.org/index.html>
        <http://purl.org/dc/terms/title>   "My first story" .
```

Embedded statements example

```
<html>
  <head> ... </head>
  <body>
    <h2 property="http://purl.org/dc/terms/title">
      My first story
    </h2>
    <p>Date:
      <span property="http://purl.org/dc/terms/created">
        2012-11-29
      </span>
    </p>
  </body>
</html>
```

Embeds two RDF statements, giving the webpage's title and creation date

Multiple subjects

- In the examples above, the subject of all the embedded RDF statements is the webpage itself (e.g. http://www.example.org/index.html)
- We can also embed arbitrary RDF, with any subject
- To do this, we use the **resource** attribute to say which subject resource we are making statements about within a given HTML element

<p resource = "http://www.inf.kcl.ac.uk/staff/simonm">

- means that the statements embedded in this <p> element are not about the webpage, but about http://www.inf.kcl.ac.uk/staff/simonm

Multiple subjects example

```
<html>
 <head> ... </head>
 <body>
  <div resource="http://www.inf.kcl.ac.uk/staff/simonm">
   <h2 property="http://xmlns.com/foaf/0.1/name">Simon</h2>
   <p property="http://xmlns.com/foaf/0.1/title">Title: Dr</p>
  </div>
  <div resource="http://www.inf.kcl.ac.uk/staff/mluck">
   <h2 property="http://xmlns.com/foaf/0.1/name">Michael</h2>
   <p property="http://xmlns.com/foaf/0.1/title">Title: Prof</p>
  </div>
 </body>
</html>
```

Embeds four RDF statements, giving the name and title of each of two people

DBpedia

- One of the largest semantic web projects is DBpedia, an open collaboration to create machine-readable translations of Wikipedia information
- Using DBpedia RDF statements, software should have access to all the same Wikipedia information that humans have
- The RDF statements currently describe over 20 million things (statement subjects), and use an ontology with over 350 classes, though these repeat subjects for different human languages

Overview

- Web services
  - SOAP
  - WSDL

- Semantic web
  - RDF
  - Ontologies
  - SPARQL
  - RDFa