3 Theory

3.0 Statement

The following chapters have been written by the team members only.
No paragraphs or figures have been copied from other sources, although some paragraphs and figures may have been influenced by web pages or presentations listed in chapter 3.3 (Further Reading and References) on page 11.

3.1 Web Services

A Web Service is many ways similar to a Web Application. The main difference is the targeted audience. A Web Application is designed to present its data to a human user via a specific category of client software – a web browser. A Web Services is designed to deliver its data to a non-human client of any kind. A Web Service thus does not “present” but merely serialise the delivered data.
Web Services can be used to allow different applications to communicate over a network. Web Services are specifically useful when communicating over the internet. They make use of standard network protocols like HTTP or SMTP to bypass even strictly configured firewalls.
Web Services are mostly independent of the platform they run on or the language they are implemented in. To achieve the required level of interoperability, various standards have been defined on how to design the Web Service interfaces. Two of these have grown to become widely accepted and will be further explained in this document: SOAP and REST.

3.1.1 SOAP

3.1.1.1 Introduction

SOAP was originally an acronym for *Simple Object Access Protocol*, but this meaning has since been lost.
It is usually served over HTTP, but can use other protocols as well (e.g. SMTP). Using predefined and well-known protocols has the already mentioned advantage that even strictly configured firewalls will highly likely let the packets pass into and out of the local network. By using a proprietary, unknown protocol you risk being blocked by a firewall configured not to deliver packets through certain ports.
However, its messages always have to be encoded in XML. Its specification requires the messages to be contained in a so-called envelope. This envelope may contain additional message headers besides the already existing protocol headers. But it has to contain the mandatory message body. This message body may contain arbitrary XML data or optionally a SOAP fault indicating an error of any type. It may optionally be wrapped in a MIME container and contain non-XML attachments.

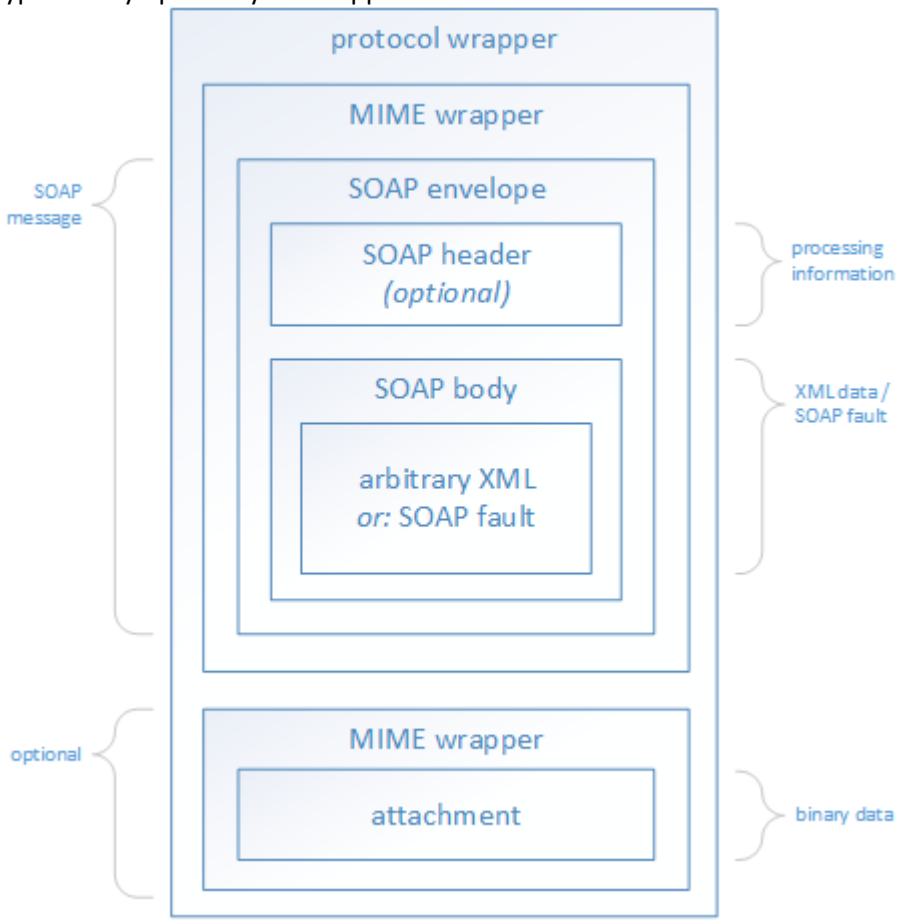

Figure 1: SOAP envelope

3.1.1.2 Example Request

```
POST http://localhost:5678/ HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: text/xml;charset=UTF-8
SOAPAction: "http://tempuri.org/TerminalService/GetStatistics"
Content-Length: 218
Host: localhost:5678
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)
```

Code Fragment 1: SOAP request

3.1.1.3 Example Response

```
HTTP/1.1 200 OK
Content-Length: 439
Content-Type: text/xml; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Sun, 24 May 2015 11:54:06 GMT

<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
 <s:Body>
  <GetStatisticsResponse xmlns="http://tempuri.org/">
   <GetStatisticsResult
      xmlns:a="http://schemas.datacontract.org/.../BFH.NetDS.WebServices.Terminal"
      xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
    <a:numberOfTimeStamps>3</a:numberOfTimeStamps>
    <a:numberOfUniqueUsers>1</a:numberOfUniqueUsers>
   </GetStatisticsResult>
  </GetStatisticsResponse>
 </s:Body>
</s:Envelope>
```

Code Fragment 2: SOAP response

3.1.1.4 WSDL

3.1.1.4.1 Introduction

The WSDL or *Web Service Description Language* is a way to describe Web Services (most notably SOAP Web Services).
The WSDL is particularly useful for easily publishing information about the Web Service in a well-known and reliable way. WSDL files can be used in a contract to specify how two parties can invoke each other's Web Services.
A WSDL v1.1 file contains the following information:

- **XML Schemas** define the transferred arbitrary (user-defined) XML data.
- **Messages** define the transferred SOAP messages. *Messages have been removed in v2.0.*
- **Port types** ***(interfaces** in v2.0)* define the Web Service interface(s).
  - **Operations** define the methods the interface provides.
- **Bindings** define the protocol(s) to use to invoke the operations.
- **Services** define the actual Web Services available.
  - **Ports** ***(endpoints** in v2.0)* define the actually available Web Service interfaces.

3.1.1.4.2 Example Document (Version 1.1)

```
<wsdl:definitions name="TerminalService" targetNamespace="http://tempuri.org/"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://tempuri.org/"
   xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl">
 <wsdl:types>
  <xsd:schema targetNamespace="http://tempuri.org/Imports">
   <xsd:import schemaLocation="http://localhost:5678/?xsd=xsd0"
      namespace="http://tempuri.org/"/>
   <xsd:import schemaLocation="http://localhost:5678/?xsd=xsd1"
      namespace="http://schemas.microsoft.com/2003/10/Serialization/"/>
   <xsd:import schemaLocation="http://localhost:5678/?xsd=xsd2"
      namespace="http://schemas.datacontract.org/2004/07/BFH...Terminal"/>
  </xsd:schema>
 </wsdl:types>
```

Code Fragment 3: WSDL v1.1 file – part 1

Please continue reading on the next page.

```
 <wsdl:message name="TerminalService_SetNews_InputMessage">
  <wsdl:part name="parameters" element="tns:SetNews"/>
 </wsdl:message>
 <wsdl:message name="TerminalService_SetNews_OutputMessage">
  <wsdl:part name="parameters" element="tns:SetNewsResponse"/>
 </wsdl:message>
 <wsdl:message name="TerminalService_GetStatistics_InputMessage">
  <wsdl:part name="parameters" element="tns:GetStatistics"/>
 </wsdl:message>
 <wsdl:message name="TerminalService_GetStatistics_OutputMessage">
  <wsdl:part name="parameters" element="tns:GetStatisticsResponse"/>
 </wsdl:message>
 <wsdl:portType name="TerminalService">
  <wsdl:operation name="SetNews">
   <wsdl:input message="tns:TerminalService_SetNews_InputMessage"
     wsaw:Action="http://tempuri.org/TerminalService/SetNews"/>
   <wsdl:output message="tns:TerminalService_SetNews_OutputMessage"
     wsaw:Action="http://tempuri.org/TerminalService/SetNewsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="GetStatistics">
   <wsdl:input message="tns:TerminalService_GetStatistics_InputMessage"
     wsaw:Action="http://tempuri.org/TerminalService/GetStatistics"/>
   <wsdl:output message="tns:TerminalService_GetStatistics_OutputMessage"
     wsaw:Action="http://tempuri.org/TerminalService/GetStatisticsResponse"/>
  </wsdl:operation>
 </wsdl:portType>
 <wsdl:binding name="BasicHttpBinding_TerminalService" type="tns:TerminalService">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="SetNews">
   <soap:operation soapAction="http://tempuri.org/TerminalService/SetNews"
     style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="GetStatistics">
   <soap:operation soapAction="http://tempuri.org/TerminalService/GetStatistics"
     style="document"/>
   <wsdl:input>
    <soap:body use="literal"/>
   </wsdl:input>
   <wsdl:output>
    <soap:body use="literal"/>
   </wsdl:output>
  </wsdl:operation>
 </wsdl:binding>
 <wsdl:service name="TerminalService">
  <wsdl:port name="BasicHttpBinding_TerminalService"
    binding="tns:BasicHttpBinding_TerminalService">
   <soap:address location="http://localhost:5678/"/>
  </wsdl:port>
 </wsdl:service>
</wsdl:definitions>
```

Code Fragment 4: WSDL v1.1 file – part 2

3.1.2 REST

3.1.2.1 Introduction

REST stands for *Representational State Transfer*.
Unlike SOAP, REST is not a protocol but merely an architecture style defining six constraints a RESTful (Web) Service has to fulfil.
Typically RESTful Web Services are invoke over HTTP. The data transferred with REST can be in any format (machine-readable or not) but most RESTful Web Services use either XML, JSON(-LD), (X)HTML or plain text. To transfer binary data, you do not need attachments or similar workarounds, you can simply return the binary data together with the correct Content-Type (in case of HTTP).
The six constraints are:

1) **Client-Server**: Client and server have to be properly separated by a uniform interface. Clients and servers have different responsibilities: e.g. servers have to deal with data storage but not with the user interface. The servers and clients are independent of each other and may be updated independently as long as the interface is not altered.
2) **Stateless**: The communication between the client and the server has to be stateless. There must not be maintained any sessions, cookies or other storage of sorts. All information necessary (the *State*) for an operation has to be included in the request or response (to be *Transfer*red).
3) **Cacheable**: Just like in Web Applications, responses can be cached by any involved part (the server, the client or any intermediary).
4) **Layered System**: The Web Service must be able to support intermediary layers e.g. to improve scalability or to enforce security policies.
5) **Code on Demand**: *optional and not discussed here*
6) **Uniform Interface**
   a. **Identification of resources**: Every resource must have a unique identification (in Web Services, resources are identified by URIs). The identification of the resource does not depend on the representation they are returned in. If used with HTTP, the client may specify the desired representation using the Accept header.
   b. **Manipulation of resources through these representations**: Once the client obtained a representation of a resource, he holds all the information necessary to update or delete a resource. The action to perform on a resource is defined by the HTTP method (if used in conjunction with HTTP): GET returns the resource, PUT replaces the resource, POST creates a new resource and DELETE deletes a resource.
   c. **Self-descriptive messages**: A response must include all the information necessary to process it. For example, it has to indicate the status (success, error), the format of the representation.
   d. **Hypermedia as the engine of application state (HATEOAS)**: The only way to maintain an application state is by using hypermedia and the references within it. A client only knows where to enter the application, from there on it can use the hypermedia to dynamically identify the actions available to it.

This last sub-constraint (HATEOAS) is a bit of a catch. Many developers (including us) do not strictly follow it. By specifying *hypermedia* as the engine of application state it requires the requests and responses to use hypermedia. But: JSON is not hypermedia (although JSON-LD is). This means, if you use JSON, you strictly cannot call your Web Service RESTful!

3.1.2.2 Example Request

```
POST http://localhost:6789/employee HTTP/1.1
Accept-Encoding: gzip,deflate
Content-Type: application/json
Content-Length: 37
Host: localhost:6789
Connection: Keep-Alive
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

{"login": "marc","name": "Marc Touw"}
```

Code Fragment 5: REST request

3.1.2.3 Example Response

```
HTTP/1.1 200 OK
Content-Length: 35
Content-Type: application/json; charset=utf-8
Server: Microsoft-HTTPAPI/2.0
Date: Sun, 24 May 2015 13:31:10 GMT

{"login": "marc","name": "Marc Touw"}
```

Code Fragment 6: REST response

3.2 Comparison of SOAP and REST

It is not really fair to compare SOAP to REST because they are two different things. However, as a rough guide: use SOAP for big, complex applications and REST for small, dynamic ones.

3.3 Further Reading and References

- Web Services in general
  - Wikipedia page on Web Services
    http://en.wikipedia.org/wiki/Web_service
  - W3Schools tutorial on Web Services
    http://www.w3schools.com/webservices/default.asp
  - Information on both SOAP and RESTful Web Services on SoapUI's Testing Dojo
    http://www.soapui.org/testing-dojo/world-of-api-testing/soap-vs--rest-challenges.html
  - Presentation on Web Services (especially in Java EE) by Stephan Fischli (2015)
    http://www.sws.bfh.ch/~fischli/courses/info/javaee/scripts/JavaWebServices.pdf
- SOAP
  - Wikipedia on SOAP Web Services
    http://en.wikipedia.org/wiki/SOAP
  - W3C SOAP recommendation
    http://www.w3.org/TR/soap/
  - Presentation on SOAP Web Services by Marine Yegoryan (2008)
    http://www.slideshare.net/guest0df6b0/web-service-presentation
- REST
  - Wikipedia page on REST
    http://en.wikipedia.org/wiki/Representational_state_transfer
  - Presentation on RESTful Web Services by David Zülke at the Dutch PHP Conference (2008)
    http://www.slideshare.net/Wombert/designing-http-interfaces-and-restful-web-services-dpc2012-20120608
  - Presentation on HTTP by Ross Tuck at ZendCon (2014)
    http://www.slideshare.net/rosstuck/http-and-your-angry-dog
  - Tutorial on RESTful Web Services by M. Vaqqas on Dr. Dobb's (2014)
    http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069