Onderzoek Fileupload

Fileupload naar remote applicatie

Afstuderen

Bert Gritter
Rob Juurlink
2004

Laatste wijziging:

maandag 12 april 2004
23:22:11 uur.

Onderzoek Fileupload

Fileupload naar remote applicatie

Versiebeheer

Datum	*auteur*	*Versie*	*Status/Wijziging*
03-03-2004	Rob	0.1	Start document
08-03-2004	Rob	0.2	Beschrijving verder uitgewerkt
09-03-2004	Rob	0.3	Start beschrijven Axis bibliotheek en Axis TCP Monitor
15-03-2004	Rob	0.4	Details upload-common en axis beschreven
16-03-2004	Rob	0.5	Autorisatie en conclusie toegevoegd
16-03-2004	Bert	1.0	Review
29-03-2004	Bert	1.1	Ontwerp + Testen toegevoegd + conclusie
29-03-2004	Rob	2.0	Review
12-04-2004	Rob	2.1	Deel van de conclusie verduidelijkt

INHOUDSOPGAVE

1. INLEIDING

Er is een vooronderzoek gepleegd naar fileupload. Deze functie voor de webapplicatie moet aan een aantal specificaties voldoen. De punten worden genoemd in het hoofdstuk Probleemstelling.

Daarnaast worden in dit document ook de te gebruiken technieken in detail behandeld om een beter beeld te krijgen hoe de communicatie met de server verloopt. Deze gebruikte technieken zijn `HTTP`, `SOAP` en `XML`.

Als de lezer een beeld heeft van de technieken, worden de verschillende bestaande bibliotheken in theorie behandeld. In dit onderzoek is een eenvoudig voorbeeld verwerkt..

De eindconclusie van het onderzoek is dat `SOAP`, specifieker, de Apache Axis implementatie, meer voordelen biedt ten opzichte van `HTTP Post`. Wat betreft de werking is `HTTP Post` dan misschien eenvoudiger te doorgronden, het gebruik van de `SOAP` implementatie betekent dat er voor het verzenden en ontvangen van objecten veel minder code nodig is en het ontwerp van het geheel daardoor eenvoudiger blijft.

Nadat het ontwerp behandeld is, wordt deze uitgewerkt met Apache Axis' SOAP implementatie. De programmacode is niet toegevoegd als bijlage. De meest recente broncode is altijd te bekijken via CVS. Een rechtstreekse verwijzing naar respectievelijk de server en client broncode in CVS:

http://afstuderen.arsoftware.nl/cgi-bin/viewcvs.cgi/afstuderen/src/Fileupload/AxisServer

http://afstuderen.arsoftware.nl/cgi-bin/viewcvs.cgi/afstuderen/src/Fileupload/AxisClient

In het laatste hoofdstuk van dit document is het testrapport opgenomen. In de tabel die daar is afgedrukt, zijn de resultaten van de verschillende testen zichtbaar.

2. PROBLEEMSTELLING

Er moeten vanuit de lokale applicatie bestanden verzonden kunnen worden naar de webapplicatie. In dit geval gaat het om fotobestanden. Voordat deze bestanden verzonden worden, moet de gebruiker geautoriseerd worden en moet gecontroleerd kunnen worden of de bestanden al aanwezig zijn op de server en zo ja, of het ook dezelfde versies zijn.

Is het te verzenden bestand al op de server aanwezig en betreft het dezelfde versie, dan hoeft dit bestand niet nogmaals verzonden te worden. In de andere gevallen wordt het bestand wel verzonden naar de webserver.

Naast het uploaden van foto's, moeten ook de objectgegevens verstuurd kunnen worden naar de webapplicatie. Zie het hoofdstuk eisen en wensen op bladzijde 6 voor meer details.

Objecten of bestanden die wel op de server aanwezig zijn, maar waarvan geen versie (meer) aanwezig is aan de kant van de client, kunnen verwijderd worden.

3. ONDERZOEK

3.1. FILEUPLOAD

3.1.1. Open-source

Het is natuurlijk onzin om zelf iets te gaan implementeren als er goede open-source bibliotheken bestaan die de gewenste functionaliteit al implementeren.

Er wordt eerst gekeken welke bibliotheken er zijn en wat de mogelijkheden er van zijn.

3.1.2. Eisen en wensen

Vanaf de cliënt moeten op een server op afstand opdrachten uitgevoerd kunnen worden. Dit wordt Remote Procedure Call (RPC) genoemd.

Een opsomming van de eisen voor de communicatie met de server op afstand:

Eis	***Omschrijving***
Authenticatie	Niet elke gebruiker mag zomaar een bestand of gegevens kunnen uploaden naar de server (webapplicatie). Voordat er gegevens verzonden kunnen worden moet bekend zijn met welke gebruiker we te maken hebben en of deze gebruiker ook de rechten heeft om data te versturen naar de server (webapplicatie).
Bestanden versturen	Er moet een mogelijkheid zijn om binaire bestanden (op een efficiënte manier) te verzenden naar de webapplicatie.
Bestanden controleren	Voordat er daadwerkelijk bestanden verzonden gaan worden, moet gecontroleerd kunnen worden of deze bestanden zich al op de server bevinden en zo ja, of het ook dezelfde versies zijn.
Welkomst bericht	Er kan zich op de server een berichten voor de gebruiker bevinden. Er moet een mogelijkheid zijn om dit tekst bericht op te halen.
Vastgoed-object	De gegevens van een vastgoed-object moeten verzonden kunnen worden naar de webapplicatie. Het VastgoedObject voldoet aan de specificaties van een JavaBean.

3.1.3. Voorkennis

Voordat we dieper op de standaard oplossingen ingaan, wordt eerst de theorie van de gebruikte technieken behandeld.

HTTP

HTTP staat voor HyperText Transfer Protocol. In het HTTP protocol is vastgelegd welke vragen (requests) een cliënt aan de server kan stellen en welke antwoorden (responses) een server daarop kan teruggeven. Elke vraag bevat een URL[1] die naar een web component of een statisch object zoals een webpagina of plaatje verwijst.

HTTP Requests

Een HTTP request bestaat uit de request-soort, de URL, de header velden en eventueel een body. Een overzicht van de meest gebruikte HTTP requests:

- `GET` – Ontvang het document gespecificeerd door de URL.
- `HEAD` – Ontvang alleen de headers van het op te vragen document.
- `POST` – Zend data naar de server.

Een voorbeeld

Een HTTP GET request kan er als volgt uit zien:

```
1  GET /index.html HTTP/1.1
2  Accept: text/html
3  Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
4  Host: localhost
5  User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.6)
```

In bovenstaande voorbeeld op regel 1 wordt het document `index.html` opgevraagd. De cliënt geeft aan dat het gebruik maakt van HTTP versie 1.1.

In regel 2 maakt de cliënt duidelijk dat het HTML documenten kan ontvangen en in de regel eronder staan de karakter coderingen die de cliënt begrijpt.

In regel 4 staat de opgevraagde hostname. Op deze manier is het voor de server duidelijk om welke domein het gaat. Er kunnen meerdere domeinen per server geconfigureerd worden, vandaar.

In regel 5 stuurt de cliënt extra info mee over de gebruikte cliënt. In dit geval de browsernaam, versie en het gebruikte besturingssysteem.

1 Een URL, dat staat voor Uniform Resource Locator, is een label, een etiket, dat aan een specifieke website, een bestand of een andere informatiebron is toegewezen.

HTTP Responses

Een HTTP response bestaat uit een resultaat-code, header velden en een body. Een overzicht van de meest voorkomende resultaat codes:

- `404 File not Found` – Het opgevraagde document bestaat niet.
- `401 Unauthorized` – Om het document op te vragen is authenticatie vereist.
- `200 Ok` – Het gevraagde document is succesvol opgevraagd.

Een voorbeeld

Het resultaat van bovenstaand GET request is:

```
1  HTTP/1.1 200 OK
2  Date: Tue, 09 Mar 2004 09:13:27 GMT
3  Server: Apache/1.3.29 (Debian GNU/Linux)
4  Content-Type: text/html; charset=iso-8859-1
5
6  1039
7  (...)
```

In regel 1 geeft de server aan dat de aanvraag succesvol afgehandeld is. Het antwoord voldoet aan HTTP versie 1.1.

In regel 2 is de datum van het opgevraagde document zichtbaar. Deze datum is voor cache management.

In regel 3 staat het type en de naam van de server die het document serveert. Deze data wordt meestal genegeerd door de cliënt.

In regel 4 staat de karakter codering van het verstuurde document en het type. In dit geval HTML. Na deze regel volgt er 2 maal een enter. Op regel 6 staat de lengte van het document in bytes. Het document zelf is niet afgedrukt in dit voorbeeld.

Details van het HTTP protocol

Details van het Het HTTP protocol staan beschreven in verschillende RFC's. Het HTTP protocol versie 1.0 staat in het RFC met nummer 1945. Versie 1.1 van het protocol staat in RFC met nummer 2616. Deze zijn te downloaden op http://www.rfc-editor.org/rfc.html.

SOAP

SOAP is een computer protocol dat wordt gebruikt voor communicatie tussen verschillende componenten. De afkorting staat voor Simple Object Access Protocol. SOAP wordt gesteund door een groot aantal bedrijven waaronder Sun, IBM, Microsoft, BEA, Oracle, Apache enz.

SOAP is een protocol dat XML berichten stuurt, gebruikelijk over HTTP, maar ook over SMTP, HTTPS of FTP is ook mogelijk.

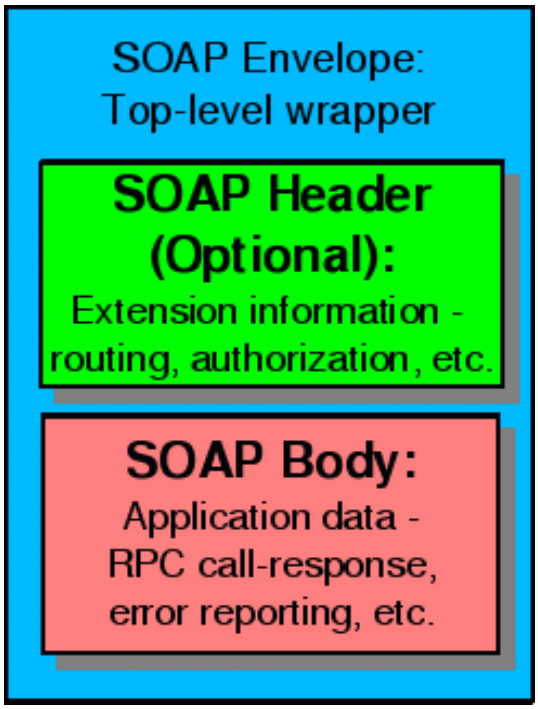
Door het gebruik van HTTP kunnen de berichten zonder speciale aanpassing te maken aan instellingen, eenvoudig een proxy of een firewall passeren, omdat deze meestal de standaard HTTP poorten open hebben staan.

Structuur

Een SOAP bericht bestaat uit een envelop. Dit is een omhulsel voor de inhoud. Het bevat verder geen informatie. In de envelop bevinden zich de SOAP header en de SOAP body. De header kan onder andere info bevatten voor het routeren van het bericht. De header is niet verplicht.

De body bevat de data van het bericht in XML gecodeerd, zie figuur 1.

Om bestanden of een andere vorm van binaire data mee te kunnen sturen, is er de mogelijkheid voor zogenaamde bijlagen aanwezig. Dit bij te voegen document bevindt zich dan buiten de SOAP envelop. De details van de XML van een SOAP aanroep staan beschreven bij de SOAP implementatie.

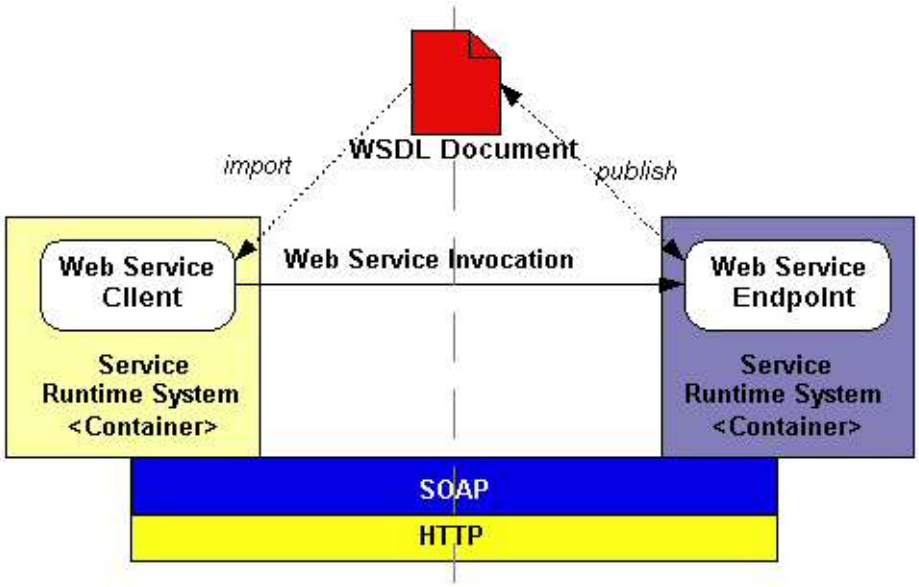


Figuur 1, de structuur van een SOAP bericht. In de envelop kan zich een header bevinden. Naast de header is er altijd een body aanwezig.

Met behulp van het SOAP protocol kunnen applicaties met elkaar communiceren; onafhankelijk van besturingssysteem, programmeertaal en objectmodel.

De geleverde SOAP service wordt beschreven in een WSDL document. Hierin staan de methoden met hun parameters die kunnen worden aangeroepen en de te verwachten antwoorden. Als van tevoren bekend is welke methoden op afstand aangeroepen kunnen worden, is een WSDL document niet verplicht.

Zie figuur 2 voor een schematische voorstelling van de communicatie tussen een Web Service eindpunt en een cliënt.



Figuur 2, communicatie tussen een Web Service en een cliënt schematisch weergegeven.

XML

De definitie van XML luidt:

> *Extensible Markup Language (XML) is een standaard voor het definiëren van formele markup-talen voor de representatie van gestructureerde gegevens in de vorm van platte tekst. Deze representatie is zowel machine leesbaar als leesbaar voor de mens.*

Met andere woorden, XML is een manier om gegevens gestructureerd vast te leggen. Dit vastleggen kan in een bestand of database zijn, maar ook voor het doorsturen van informatie over het internet. Dit laatste wordt gebruikt bij SOAP.

Het gaat in dit formaat dus meer om de structuur van informatie, dit in tegenstelling tot HTML, waarin het meer gaat om de presentatie van de informatie. In een HTML-bestand beschrijven de tags wel hoe informatie moet worden gepresenteerd maar niet wat deze informatie betekent, dit is precies het tegenovergestelde van XML.

Hoewel in principe de XML tags vrij te kiezen zijn, is het bij uitwisseling van gegevens wel zo handig als er een gemeenschappelijke standaard wordt afgesproken. Op deze manier ontstaan er allerlei XML-dialecten, elk met een eigen specifieke toepassing.

```
<sizes>
  <item>123456</item>
  <item>1646</item>
  <item>564456</item>
</sizes>
```

Een voorbeeld van een stukje gestructureerde XML. Elke tag moet afgesloten worden met een eindtag. De tags moeten afgesloten worden in dezelfde volgorde als dat ze geopend worden.

3.1.4. HTTP Post of SOAP

De communicatie met de server zou op verschillende manieren kunnen plaatsvinden. Elke methode heeft z'n eigen voor- en nadelen. De twee verschillende manieren die het meest voor de hand liggen en zich in de praktijk bewezen hebben zijn door rechtstreeks gebruik te maken van HTTP Post of door gebruik te maken van SOAP.

Deze twee methoden komen erg overeen. Bij het gebruik van rechtstreeks HTTP Post moet het resultaat van een aanroep zelf geparst moet worden, er ligt hiervoor niets vast in specificaties.

Bij SOAP daarentegen is in de specificaties vastgelegd hoe de data gecodeerd en gedecodeerd moet worden. De implementatie neemt het werk van coderen en decoderen uit handen van de ontwikkelaar. Ook SOAP gebruikt als onderliggende communicatieprotocol voor z'n communicatie HTTP.

3.1.5. HTTP Post

Commons FileUpload

Commons FileUpload is een Java package van Apache. Op de website staat de volgende omschrijving:

> ***The Commons FileUpload package makes it easy to add robust, high-performance, file upload capability to your servlets and web applications.***
>
> *FileUpload parses HTTP requests which conform to RFC 1867 , "Form-based File Upload in HTML". That is, if an HTTP request is submitted using the POST method, and with a content type of "multipart/form-data", then FileUpload can parse that request, and make the results available in a manner easily used by the caller.*

Volgens de website zou deze Java Package op HTML-formulieren gebaseerde fileupload mogelijk moeten maken. De data wordt via een HTTP-POST aanroep verzonden. Wat dit precies inhoud komen we in de volgende hoofdstukken op terug.

De projectwebsite is te bereiken via het adres:
http://jakarta.apache.org/commons/fileupload/

Commons HTTP Client

Commons HTTP Client is een Java package van Apache. Op de website staat de volgende omschrijving:

> *Standards based, pure Java, implementation of HTTP versions 1.0 and 1.1, full implementation of all HTTP methods (GET, POST, PUT, DELETE, HEAD, OPTIONS, and TRACE) in an extensible OO framework.*
>
> *Supports encryption with HTTPS (HTTP over SSL) protocol and Transparent connections through HTTP proxies.*

Commons HTTP client vereenvoudigt het werken met HTTP aan de cliënt kant en is een alternatief voor de Java packages in het standaard Java framewerk.

Projectwebsite is te bereiken via het adres:
http://jakarta.apache.org/commons/httpclient/

De HTTP Client heeft dependencies op commons-logging.

Een praktijkvoorbeeld

Installatie

Om de bibliotheek commons-http cliënt te kunnen gebruiken moet deze in het classpath gezet worden. Voor het loggen gebruikt de bibliotheek een eigen logging API. Om het voorbeeld aan de praat te kunnen krijgen, moet ook deze bibliotheek in het classpath van Java aanwezig zijn. Deze logging bibliotheek kan ook gedownload worden van de Apache commons webwerf.

De standaard bibliotheek commons-fileupload bevat alle functionaliteit om de uploads te kunnen verwerken. Het toevoegen van de commons-fileupload bibliotheek aan het path is voldoende om er mee te kunnen werken.

HTTP Cliënt, de code

De code om een bestand via HTTP Post te uploaden naar een server.

```
1    /**
2     * Een bestand uploaden via HTTP POST.
3     * @param targetUrl De locatie waarnaar het bestand
      *                  ge-HTTP-Post wordt.
4     * @param targetFileName Het bestand dat verstuurd gaat worden.
5     */
6    private static void doUploadFile(String targetUrl,
         String targetFileName) {
7
8        MultipartPostMethod filePost;
         filePost = new MultipartPostMethod(targetUrl);
9
10       try {
11           File targetFile = new File(targetFileName);
12
13           logger.info("Uploading " + targetFile.getName() + " to "
+
                 targetUrl);
14           filePost.addParameter(targetFile.getName(), targetFile);
15           HttpClient client = new HttpClient();
16           client.setConnectionTimeout(5000);
17
18           int status = client.executeMethod(filePost);
19
20           if (status == HttpStatus.SC_OK) {
21               logger.info("Upload complete");
22               System.out.println("response=" +
                     filePost.getResponseBodyAsString());
23           } else {
24               logger.severe("Upload failed, response=" +
                     HttpStatus.getStatusText(status));
25           }
26       } catch (Exception ex) {
27           logger.severe("Error: " + ex.getMessage());
28           ex.printStackTrace();
29       } finally {
30           filePost.releaseConnection();
31       }
32   }
```

Fileupload, de code

Om de werking te demonstreren is er een werkend voorbeeld geïmplementeerd. Commons-fileupload breidt de functionaliteit van een bestaande servlet uit. Hieronder volgt de programmacode die nodig is om een bestand dat verstuurt is via HTTP Post te ontvangen en te verwerken:

```
1  private void processRequest(HttpServletRequest request,
       HttpServletResponse response) throws ServletException {
2
3         // Is de request een fileupload? nee, dan geen actie.
4         boolean isMultipart = FileUpload.isMultipartContent
                                                    (request);
5         if (!isMultipart) return false;
6
7         String result = "";
8         List items = null;
10        // Creeer nieuwe file upload handler
11        DiskFileUpload upload = new DiskFileUpload();
12        upload.setSizeMax(2097152);
14        items = upload.parseRequest(request);
15
16        // De bestanden staan in de lijst items
17        if (items != null) {
18            for (Iterator iter = items.iterator; iter.hasNext();) {
19                FileItem item = (FileItem) iter.next();
20
21                // Het item is een formulier tekstveld
22                if (item.isFormField()) {
23                    String name = item.getFieldName();
24                    String value = item.getString();
25                    result +="formveld:"+name+"="+value+"<br/>\n";
26                }
27                // Het item is een bestand
28                else {
29                    String fieldName   = item.getFieldName();
30                    String fileName    = item.getName();
31                    String contentType = item.getContentType();
32                    boolean isInMemory = item.isInMemory();
33                    long sizeInBytes   = item.getSize();
34                    result +="Bestand: veldnaam='" +fieldName+"',"+
35                             "filename='" + fileName       + "'," +
36                             "contentType='" + contentType + "'," +
37                             "isInMemory:" + isInMemory    + ", " +
38                             "sizeInBytes:" + sizeInBytes  +
39                             "<br />\n";
40
41                    // De inhoud van het bestand kopieren.
42                    // De stream in stapjes van 1024 bytes uitlezen
43                    StringWriter stringWriter = new StringWriter();
44                    InputStream uploadedStream=item.getInputStream
();
45
46                    int count=0;
47                    byte[] buf = new byte[1024];
48                    int len;
```

```
49
50                      // Buffercopy
51                      while ((len = uploadedStream.read(buf)) >= 0) {
52                          stringWriter.write(new String(buf,0,len));
53                          count++;
54                      }
55                      stringWriter.close();
56                      uploadedStream.close();
57
58                      // Het bestand afdrukken naar scherm (echo)
59                      result += "Bestand:" + stringWriter.toString();
60                      logger.info(count + " Kbyte(s) copied");
61                  }
62              }
63          }
64          // HTML terug naar de browser
65          response.setContentType("text/html");
66          PrintWriter out = response.getWriter();
67          out.println("<html>");
68          out.println("<head>");
69          out.println("<title>Fileupload</title>");
70          out.println("</head>");
71          out.println("<body>");
72          out.println("<h1>Resultaat upload</h1>");
73          out.println("<p>" + result + "</p>");
74          out.println("</body>");
75          out.println("</html>");
76      }
77  }
```

De methode processRequest verwerkt de GET en de POST request.

In regel 4 wordt gecontroleerd of het te verwerken document een zogenaamd `multipart` document is. Een upload wordt `multipart` gecodeerd.

In regel 12 wordt de maximale lengte van een fileupload op 2 megabyte ingesteld.

In regel 14 wordt het document geparst door de `common-upload` code. De bestanden zijn daarna via een `List` te benaderen.

In regel 18 wordt de lijst doorgebladerd. De gegevens van het bestand, zoals de bestandsnaam worden in regel 29 uitgelezen.

De data wordt in regel 51 uitgelezen. Deze data kan naar schijf gekopieerd worden, maar in dit geval wordt de data naar een String gekopieerd om te demonstreren dat de inhoud ingelezen en verwerkt is.

vanaf regel 65 tot regel 75 wordt data teruggestuurd naar de cliënt.

3.1.6. Apache Axis (SOAP)

Apache Axis is de opvolger van de Apache SOAP implementatie v2.0. De thuis locatie van Apache Axis is http://ws.apache.org/axis.

Installatie

Naast het uitpakken van het Axis archief in de `webapp` map van Tomcat, moeten er nog een aantal bibliotheken aan het classpath toegevoegd worden. Deze bibliotheken hebben we globaal toegevoegd aan de webserver door ze in de map `common/lib` te plaatsen. Dit bibliotheken zijn de volgende:

- ✔ XML Parser, Bijvoorbeeld http://xml.apache.org/xerces-j. Deze is al standaard aanwezig in Apache Tomcat versie 5.
- ✔ Activation API, activation.jar: http://java.sun.com/products/javabeans/glasgow/jaf.html
- ✔ JavaMail API, mail.jar: http://java.sun.com/products/javamail/downloads/index.html

Onderstaande API's zijn niet globaal beschikbaar gemaakt via de webserver, maar bijgevoegd bij een webarchief:

- ✔ Axis API, axis.jar, jaxrpc.jar en saaj.jar. De Axis SOAP implementatie en de bijbehorende interfaces.

De volgende API's zijn optioneel:

- ✔ XMLSecurity API, http://xml.apache.org/security

Als de eenvoudige variant van de Web Service (JWS) gebruikt wordt, of als de configuratie ingesteld wordt via de admin Servlet, dan moet de Tomcat Webserver schrijf-rechten hebben in de map WEB-INF.

De eenvoudige variant van de Web Services zet na een eerste aanroep van de Web Service de gecompileerde bestanden in de WEB-INF map.

De eenvoudige variant van de Web Service en de admin servlet worden beide niet gebruikt in de uiteindelijke code.

JAX-RPC

Axis implementeert de JAX-RPC API, de standaard manier om Java Web Services te programmeren. De specificaties hiervoor zijn opgesteld door Sun. Door volgens deze specificaties te programmeren (het zijn interfaces) kan uiteindelijke elke willekeurige uitwisselbare implementatie gebruikt worden.

JWS Web Service

Nadat het standaard webarchief geïnstalleerd is (`axis.war`), is Axis te benaderen via de resource `/axis` op de webserver. Deze is dan bijvoorbeeld aan te roepen in een browser via de URL http://localhost:8080/axis.

Er verschijnt dan een standaard scherm waarin de installatie getest kan worden.

Het creëren van een Web Service in Apache Axis kan op meerdere manier. De eenvoudigste manier is het hernoemen van een *.java bestand naar *.jws. Nadat het verplaatst is naar de Axis map en na de eerste aanroep van dit bestand (de service) wordt het gecompileerd tot een Web Service en uitgevoerd.

Deze JWS Web Services zijn bedoeld voor de eenvoudige Web Service. Het is niet mogelijk om bibliotheken te gebruiken en omdat de code gecompileerd wordt tijdens de eerste aanroep, is niet van tevoren bekend of de compilatie fouten op zal leveren.

Productiecode moet Java classes met een eigen deployment descriptor gebruiken om een Web Service te installeren. JWS is puur om te testen.

Web Service functionaliteit toevoegen aan eigen webapplicatie

Bij een uitgebreidere webapplicatie zou het een mooie oplossing zijn als de Web Service functionaliteit van Axis toegevoegd kan worden aan de eigen webapplicatie.

Dit is natuurlijk mogelijk en dat hebben we dan ook gedaan. Om dit te realiseren moeten de volgende handelingen uitgevoerd worden:

- Voeg de Axis bibliotheken toe aan de webapplicatie. (Of zorg dat ze globaal aanwezig zijn in de webserver). axis.jar, jaxrpc.jar, saaj.jar enz.
  - Alle instellingen uit de web.xml van Axis overzetten naar de web.xml van de eigen webapplicatie.
- Creëer het Axis configuratiebestand `server-config.wsdd` en voeg deze toe aan de `WEB-INF` map.

De deel van de `web.xml` waarin de Axis Web Service geactiveerd wordt, ziet er als volgt uit. De niet gebruikte onderdelen van Axis zijn uit de configuratie verwijderd:

```
<servlet>
    <servlet-name>AxisServlet</servlet-name>
    <display-name>Apache-Axis Servlet</display-name>
    <servlet-class>org.apache.axis.transport.http.AxisServlet
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>AxisServlet</servlet-name>
    <url-pattern>/services/*</url-pattern>
</servlet-mapping>

<mime-mapping>
    <extension>xsd</extension>
    <mime-type>text/xml</mime-type>
</mime-mapping>
```

Axis cliënt

Hieronder een eenvoudig werkend voorbeeld van een cliënt die een echo Web Service (een standaard Web Service uit de Axis distributie) aanroept. Onder de code staat de uitleg van de verschillende delen van de code.

```
10  import org.apache.axis.client.Call;
11  import org.apache.axis.client.Service;
12  import javax.xml.namespace.QName;
13
14  public class TestClient {
15
16      public static void main(String[] args) {
17          try {
18              String endpoint =
19               "http://juurlink.org:8080/axis/echo/CustomEcho.jws";
20
21              // creeer nieuwe service en Call objecten
22              Service service = new Service();
23              Call call = (Call) service.createCall();
24
25              // the destination for our SOAP message
26              call.setTargetEndpointAddress(
                    new java.net.URL(endpoint));
27
28              // the operation (method) name of the Web Service.
29              call.setOperationName("echo");
30
31              // De methode op afstand aanroepen.
                // De parameters staan in eem array.
32              String ret = (String) call.invoke(
                    new Object[]{"Hallo!"});
33
34              System.out.println("Verzonden='" + ret + "'");
35
36          } catch (Exception e) {
37              System.err.println(e.toString());
38          }
39      }
40  }
```

In regel 22 en 23 wordt een service en een “Call” object gecreëerd. Met deze objecten kan een methode van een Web Service aangeroepen worden.

In regel 26 wordt verteld op welke locatie de Web Service te vinden is.

In regel 29 staat de naam van de methode die we gaan aanroepen. In de regels 32 en 33 wordt de methode aangeroepen met één parameter “Hallo!”.

Details request en response

De cliënt stelt de opdracht samen volgens het SOAP protocol. Deze aanvraag inclusief de HTTP headers ziet er onder water als volgt uit:

```
POST /axis/echo/CustomEcho.jws HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, multipart/related, text/*
User-Agent: Axis/1.1
Host: juurlink.org
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 397

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <echo
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <arg0 xsi:type="xsd:string">Hallo!</arg0>
  </echo>
 </soapenv:Body>
</soapenv:Envelope>
```

Het resultaat wordt “geparst” naar een string. De bron van het SOAP bericht, dat een envelop en een inhoud bevat, ziet er als volgt uit:

```
HTTP/1.1 200 OK
Content-Type: text/xml;charset=utf-8
Date: Tue, 09 Mar 2004 11:41:06 GMT
Server: Apache-Coyote/1.1
Connection: close

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
  <echoResponse
soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <echoReturn xsi:type="xsd:string">Echo: Hello!</echoReturn>
  </echoResponse>
 </soapenv:Body>
</soapenv:Envelope>
```

Axis TCP monitor

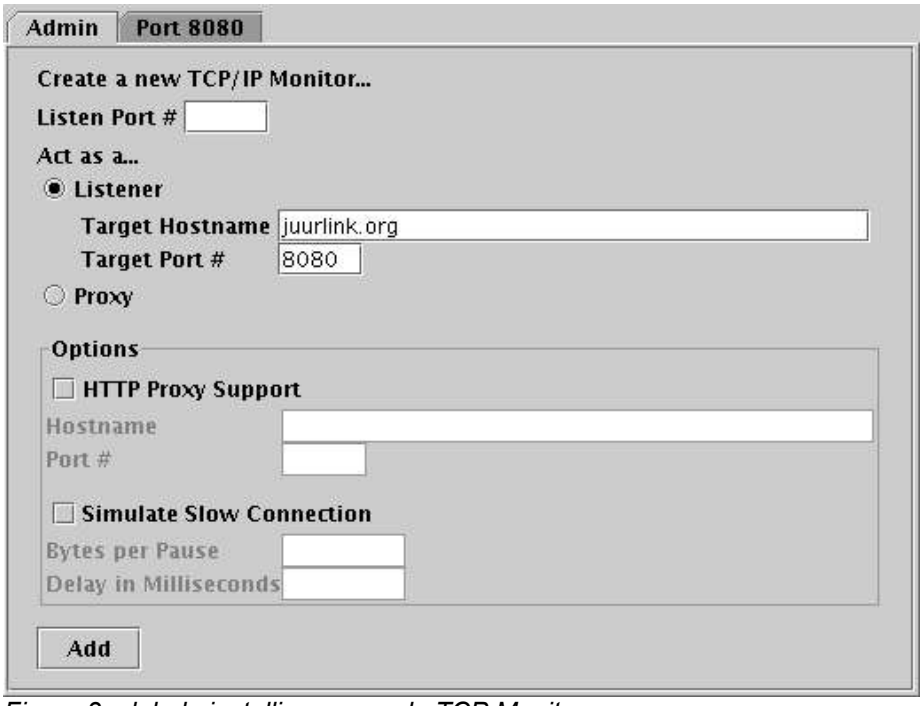
Bij de distributie van Apache Axis zit een TCP Monitor. Deze Monitor wordt als een proxy tussen de cliënt en Web Service geplaatst en kan daardoor alle SOAP berichten inclusief de HTTP Headers afluisteren en afbeelden.

Start de monitor op de volgende manier:

```
$java org.apache.axis.utils.tcpmon [listenPort targetHost
targetPort]
```

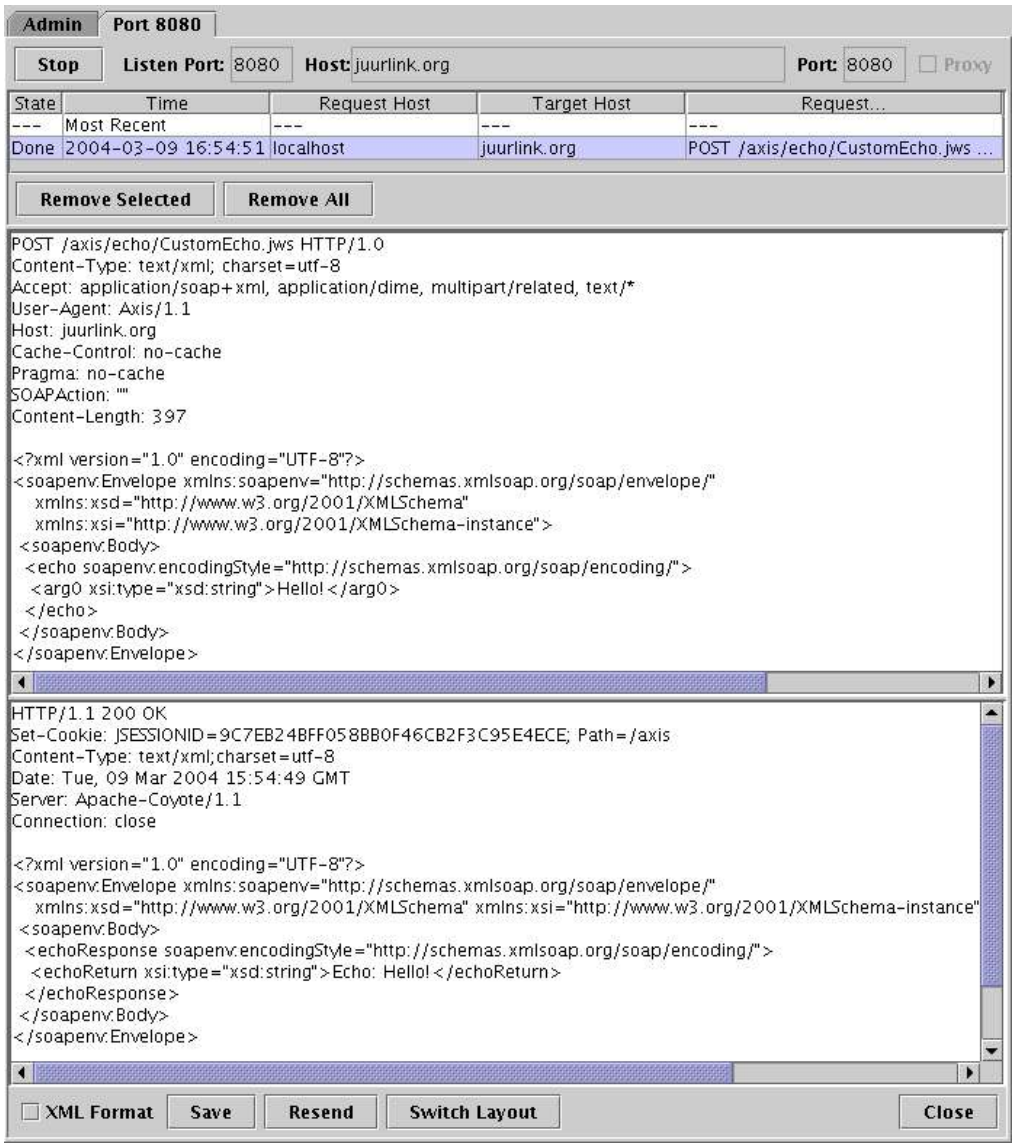
Daarnaast kan een verzonden XML bericht in de TCP Monitor gewijzigd worden en opnieuw verzonden worden. In dat laatste geval moet er wel even rekening mee gehouden worden dat de content-lengte handmatig aangepast moet worden.

In figuur 3 wordt de TCP Monitor ingesteld als proxy die z'n aanvragen (requests) doorstuurt naar de server met hostnaam juurlink.org. De proxy luistert naar poort 8080 (niet zichtbaar in dit scherm) en stuurt de aanvraag ook door naar poort 8080 (wel zichtbaar).



Figuur 3, globale instellingen van de TCP Monitor.

In figuur 4 is een aanvraag (bovenste deel van venster) en de reactie daarop (onderste deel van venster) afgebeeld van de echo Web Service.



Figuur 4, de cliënt stuurt een aanvraag naar de Web Service. In deze aanvraag voert het de methode "echo" uit met als parameter "Hallo!" (bovenste scherm). In het onderste gedeelte van het scherm komt de reactie terug "Echo: Hallo!". het type van de reactie is een String.

Een Web Service

Een van afstand aan te roepen methoden van een object, oftewel een Web Service eindpunt, is een gewone methode die publiekelijk toegankelijk moet zijn. Er zij geen speciale bibliotheken vereist en ook hoeft er niet een bepaalde interface geïmplementeerd te worden. Onderstaande fragment toont een methode die dienst kan doen als Web Service eindpunt:

```
public int getPostcode (String city) {

    // Controleer de plaats en geen postcode terug
    if (city.equals("Nijverdal")) return 7440;
    if (city.equals("Vroomshoop")) return 7681;
    return -1;
}
```

Bovenstaande Web Service wordt aangeroepen via Axis. Om deze Web Service te activeren moet het bestaan van deze methode aan de configuratie `server-config.wsdd` bekend gemaakt worden. Het geheel van de configuratie voor bovenstaande Web Service ziet er dan als volgt uit:

```
<service name="urn:vastgoedonline" provider="java:RPC">

    <parameter name="scope" value="request"/>
    <parameter name="allowedMethods" value="getPostcode"/>
    <parameter name="className"
               value="services.PostcodeService"/>
</service>
```

Een compleet configuratiebestand `server-config.wsdd` van de geïmplementeerde test Web Service is als bijlage bijgevoegd.

Een minpunt van SOAP is dat er standaard geen complexe objecten verstuurd kunnen worden, zoals bijvoorbeeld een HashMap of een Vector object. In de SOAP specificatie zijn een aantal typen mogelijk. In onderstaande tabel zijn de typen afgebeeld waarvan er ook een Java equivalent bestaat.

WSDL	***Java***
xsd:boolean	boolean
xsd:byte	byte
xsd:double	double
xsd:float	float
xsd:int	int
xsd:long	long
xsd:string	String

Bestanden verzenden

Er zijn meerdere manieren mogelijk om binaire data te verzenden in een SOAP bericht. De eenvoudigste manier zou zijn om de data te coderen volgens de base64 codering en de bericht daarna te versturen als tekst.

Het grote nadeel van deze methode is dat base64 codering erg inefficiënt is, omdat het maar 6 bits voor de data kan gebruiken. Dat betekent dat het te versturen bestand een derde deel groter wordt dan het origineel.

Bij het versturen van grotere bestanden is niet alleen deze toename van grootte nadelig, maar ook de extra processorcapaciteit die het kost om het bestand te decoderen. De conclusie is dat voor het versturen van bestanden van een paar kilobyte base64 codering gebruikt kan worden, maar voor het versturen van grotere bestanden zal een andere manier gezocht moeten worden.

SOAP bijlagen

Het is mogelijk om binaire bijlagen mee te sturen met een SOAP bericht. Deze methode wordt SOAP with bijlagen (SwA) genoemd. Bij deze methode wordt het SOAP bericht als 1e bijlage van een set verstuurd naar de Web Service. De bijlagen worden er daarna bij aan geplakt. Deze set van bijlagen wordt gecodeerd volgens de MIME methode. MIME is een manier van coderen die bijvoorbeeld ook bijlagen in e-mail mogelijk maken, zie ook het document over Mime mail.

Om de bijlagen volgens het MIME formaat te coderen, wordt de programmacode van de mail API bibliotheek gebruikt. Om MIME te activeren zijn de bibliotheken `mail.jar` en `activation.jar` vereist.

Een andere manier om binaire bijlagen te versturen is volgens het DIME formaat. Om het volgens het DIME formaat te coderen, moet er een extra eigenschap ingesteld worden. Apache Axis ondersteunt beide formaten. Aan de ontvangende kant hoeft het formaat niet ingesteld te worden. Axis detecteert zelf in welk formaat het verstuurde bericht gecodeerd is. DIME is een protocol dat bedacht is door Microsoft.

De cliënt

In de onderstaande code wordt een bestand voorbereid om verzonden te worden als MIME SOAP bijlage. Beschrijving van de code is als commentaar in de onderstaande code aanwezig.

```
public boolean uploadFile(File pFile) {

    // Controleer of het bestand bestaat en een echt bestand is
    if (pFile==null || !pFile.isFile()) return false;

    try {
        // Creer de data voor de bijlage
        DataHandler fileData;
        fileData = new DataHandler(new FileDataSource(pFile));
        // creeer nieuw Service en Call object (standaard JAX-RPC)
        Service service = new Service();
        Call call = (Call) service.createCall();
        // De locatie van de Web Service
        call.setTargetEndpointAddress(new java.net.URL(ENDPOINT));
        // De methodenaam van de Web Service.
        call.setOperationName(
```

```
        new QName("urn:vastgoedonline","uploadFile"));

        QName qnamebijlage;
        qnamebijlage=new Qname("urn:vastgoedonline","DataHandler");

        // Serializer voor bijlage
        call.registerTypeMapping(fileData.getClass(),
                        qnamebijlage,
                        JAFDataHandlerSerializerFactory.class,
                        JAFDataHandlerDeserializerFactory.class);

        call.addParameter("binary", qnamebijlage,
                          ParameterMode.IN);
        call.addParameter("filename", XMLType.XSD_STRING,
                          ParameterMode.IN);
        call.setReturnType( XMLType.XSD_BOOLEAN );
        //DIME codering?
        // call.setProperty(call.bijlage_ENCAPSULATION_FORMAT,
        // call.bijlage_ENCAPSULATION_FORMAT_DIME);
        Boolean success = (Boolean)
            call.invoke(new Object[]{fileData,pFile.getName()});
        return success.booleanValue();

        } catch (AxisFault fault) {
            logger.severe(fault.toString());
            return false;
        } catch (Exception e) {
            logger.severe(e.toString());
            return false;
        }
    }
```

De server

De methode die op de server aangeroepen wordt, ziet er als volgt uit. Axis handelt het ontvangen af en plaatst het bestand in een map die in het configuratiebestand ingesteld is. Het bijbehorende configuratiebestand is afgedrukt op bladzijde 26.

```
public boolean uploadFile(DataHandler dh, String filename)
        throws Exception {

    if (dh == null ) {
        logger.warning("Bestand is null!");
        throw new FileNotFoundException("Bestand niet gevonden!");
    }
    String receivedfileName = dh.getName();//Get the filename.
    logger.info("Bestand: '" +receivedfileName+ "'(" +filename+ ")");

    File destinationFile = new File("/tmp/" + filename);
    File sourceFile = new File(receivedfileName);
    // Lokale methode om bestand te kopieren
    copyFile(sourceFile, destinationFile);
    sourceFile.delete();
    return true;
}
```

Configuratie Axis voor versturen bijlagen

Om bijlagen te activeren, moet een zogenaamde `deserializer` ingesteld en geconfigureerd worden in Axis' eigen configuratiebestand. Deze configuratie die is opgenomen in het bestand `server-config.wsdd` in het deel waar de Web Service geconfigureerd wordt, ziet er als volgt uit:

(De classnaam en de locatie is voor de leesbaarheid uit elkaar getrokken)

```
<typeMapping
    deserializer="org.apache.axis.encoding.ser.
                  JAFDataHandlerDeserializerFactory"
    languageSpecificType="java:javax.activation.DataHandler"
                   qname="ns1:DataHandler"
              serializer="org.apache.axis.encoding.ser.
                          JAFDataHandlerSerializerFactory"
           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
/>
```

Standaard JavaBeans versturen

Standaard is het in SOAP niet mogelijk om complexe objecten te versturen. Om het verzenden van niet standaard typen toch mogelijk te maken, is er in Axis een zogenaamde `serializer` en `deserializer` aanwezig.

Deze uitbreiding in Axis maakt het mogelijk om zonder extra code toch complexe objecten over te sturen. De enige vereiste is dat de objecten voldoen aan de specificatie voor een JavaBean. Met andere woorden, het moet een object zijn dat `get-` en `set-` methoden bevat via welke de instance variabelen te benaderen zijn.

Hieronder is afgebeeld hoe de uitbreiding voor JavaBeans te activeren is. Deze instelling wordt toegevoegd aan het `server-config.wsdd` bestand:

```
<beanMapping qname="vastgoedonline:VastgoedObject"
             xmlns:vastgoedonline="urn:BeanService"
             languageSpecificType="java:nl.arsoftware.vastgoedonline.
                                              domain.VastgoedObject"/>
```

De extra code die toegevoegd moet worden om een JavaBean aan de kant van de cliënt te kunnen versturen, ziet er als volgt uit:

```
QName qn  = new QName( "urn:BeanService", "VastgoedObject" );
call.registerTypeMapping(VastgoedObject.class, qn,
    new org.apache.axis.encoding.ser.BeanSerializerFactory
        (VastgoedObject.class, qn),
    new org.apache.axis.encoding.ser.BeanDeserializerFactory
        (VastgoedObject.class, qn));

call.addParameter( "vastgoedObject", qn, ParameterMode.IN );
```

Authenticatie

Last, but not least, de authenticatie. Het is natuurlijk belangrijk om te kunnen vaststellen wie de methode van de Web Service uitvoert. Dit is mogelijk door gebruik te maken van authenticatie. De functionaliteit hiervoor is standaard in Axis aanwezig.

In de cliënt worden twee extra methoden aangeroepen waarmee de gebruikersnaam en wachtwoord opgegeven wordt. De extra code voor de cliënt ziet er als volgt uit:

```
call.setUsername( username );
call.setPassword( password );
```

Door de extra code wordt bij elke aanroep de gebruikersnaam en het wachtwoord meegestuurd. het wachtwoord wordt volgens de `HTTP basic authentication` methode verwerkt in de HTTP header, zie onderstaande fragment van een aanvraag:

```
POST /vastgoedonline/services HTTP/1.0
SOAPAction: ""
Content-Length: 391
Authorization: Basic cm9iOnJvYjAx
```

HTTP Basic authentication

In bovenstaande `Authorization` header geeft het woordje basic aan dat `HTTP basic authentication` gebruikt word. De gebruikersnaam en het wachtwoord staan daar in gecodeerde vorm achter. Ze zijn aan elkaar geplakt met een dubbele punt er tussen en vervolgens gecodeerd volgens de base64 codering.

Authenticatie implementeren en activeren

Om de functionaliteit in Axis te activeren, moet er een zogenaamde `AuthenticationHandler` geïmplementeerd worden. Dit is een class die erft van `BasicHandler` class uit Axis en aangeroepen wordt voordat een SOAP aanvraag de Web Service bereikt.

De code in `AuthenticationHandler` haalt de gebruikersnaam en het wachtwoord uit een `SessionContext` en laat via een `SecurityProvider` controleren of deze gebruiker mag inloggen. Als de authenticatie geslaagd is, wordt er een `AuthenticatedUser` object gecreëerd waarin de naam van de gebruiker bewaard wordt. De gebruik is nu succesvol geauthentiseerd.

Het niet slagen van een authenticatie daarentegen levert een AxisFault op, wat op z'n beurt weer een HTTP code 401 oplevert, dat “Not Authorized” betekent.

De code voor de `AuthenticationHandler` ziet er als volgt uit:

```
public void invoke(MessageContext messageContext) throws AxisFault {

    // Creeer de security provider
    SecurityProvider provider;
    provider = new SecurityProviderImpl();

    String username = messageContext.getUsername();
    String password = messageContext.getPassword();

    if ( username == null || username.equals(""))
        throw new AxisFault("Server.Unauthenticated", username,
                                                    null, null );

    AuthenticatedUser authUser = provider.authenticate
                                    (messageContext);

    // Als de gebruiker ge-authenticeerd is, is er een
    // authUser object gecreeerd
    if ( authUser == null)
        throw new AxisFault( "Server.Unauthenticated", username,
                                                    null, null );

    logger.info("Gebruiker succesvol geauthenticeerd!");
}
```

In de Apache Axis distributie bevindt zich voorbeeldcode hoe functionaliteit voor authenticatie geschreven kan worden. Deze voorbeeldcode is geïmplementeerd in de classes `SimpleAuthenticationHandler`, `SimpleAuthenticatedUser` en `SimpleSecurityProvider`.

Om er voor te zorgen dat er voor elke SOAP aanroep geautoriseerd wordt, moet de volgende code nog even toegevoegd worden aan het configuratiebestand van Axis (`server-config.wsdd`). In onderstaande configuratie wordt aangegeven welke `Handler` de autorisatie afhandelt:

```
<requestFlow name="checks">
    <handler type="java:nl.arsoftware.vastgoedonline.services.
                                            AuthenticationHandler"/>
</requestFlow>
```

3.2. CONCLUSIE

De applicatie wordt op afstand via HTTP benaderd, dit is een vaststaand feit, echter het op afstand aanroepen van methoden kan op verschillen de manieren. In dit document zijn twee verschillende manieren van communiceren met de server behandeld, te weten rechtstreeks HTTP POST en GET commando's, of met behulp van het SOAP protocol (dat onder water ook HTTP POST gebruikt).

Beide manieren hebben hun eigen voor- en nadelen.

3.2.1. HTTP POST en GET

Voordelen

Het HTTP protocol is een eenvoudig protocol dat goed te begrijpen is. Door gebruik te maken van de bestaande `commons-fileupload` en `commons-http-client` bibliotheken, kan de functionaliteit met slechts enkele methoden geïmplementeerd worden.

Er zijn behalve bovenstaande twee bibliotheken geen extra bibliotheken vereist.

De functionaliteit voor het uploaden en het ontvangen van een binair bestand is met een paar methoden geïmplementeerd.

Nadelen

Als een methode, of in dit geval een aanroep, data oplevert, zal deze handmatig “geparst” moeten worden. Het resultaat wordt altijd ontvangen als één groot tekstdocument.

Complexe objecten zoals een HashMap of een Vector object zijn niet rechtstreeks te versturen en en er is ook geen standaard functionaliteit aanwezig om JavaBeans te verzenden.

Er kunnen via een HTTP Post alleen naam-waarde combinaties als tekst verzonden worden. Een `integers`, `long` of `double` versturen is standaard niet mogelijk.

Om te kunnen communiceren met de server, of om deze opdrachten te geven, moet een eigen protocol bedacht worden.

3.2.2. SOAP

Voordelen

SOAP is een standaard.

De mogelijkheid voor het versturen van een object is aanwezig als dit object voldoet aan de JavaBean specificaties.

Het verzenden van waarden die geen tekst zijn, zoals een `integer`, een `double` of een `long` is standaard mogelijk m.b.v. het SOAP protocol.

Er kan eenvoudig een nieuwe service toegevoegd worden door een nieuwe methode toe te voegen aan de service class. Axis zorgt er voor dat deze service beschikbaar is voor de cliënt.

Nadelen

Het implementeren van de een SOAP Web Service kost meer tijd. Om een simpele Web Service op te zetten kost moet Axis geconfigureerd en geïnstalleerd worden.

Het ontwerp is complexer. De programmeur moet het concept van verschillende technieken kennen, zoals SOAP, HTTP en XML.

Er is meer dan één externe bibliotheken vereist. Zoals een aparte bibliotheek voor de interfaces, de implementatie, het “parsen” van de XML, de autorisatie en MIME coderen van de bijlagen enz.

3.2.3. Eindconclusie

Wat functionaliteit betreft zitten er aan het gebruik van SOAP alleen maar voordelen. Het enige nadeel dat er aan zit is de extra complexiteit die het oplevert ten opzichte van het gebruik van alleen HTTP.

Omdat op dit moment actief aan een nieuwe versie van Apache Axis gewerkt wordt en omdat SOAP een protocol lijkt dat ook in de toekomst gebruikt zal blijven worden, is het geen weggegooide tijd om de werking van SOAP te doorgronden.

De eindconclusie is dat voor de communicatie tussen de cliënt en de server SOAP gebruikt gaat worden.

Naast de in dit document behandelde theorie, zou in de toekomst nog gekeken kunnen worden naar het gebruik van sessies en het comprimeren van de SOAP XML data. Beide zouden de performance van de verbinding kunnen verbeteren.

4. ONTWERP

4.1. ANALYSE

Het ontwerp voor deze deelproduct is opgesplitst in twee delen. Er is een ontwerp van de client en een ontwerp van de server. De zes punten die hieronder beschreven worden zijn van belang voor het ontwerp.

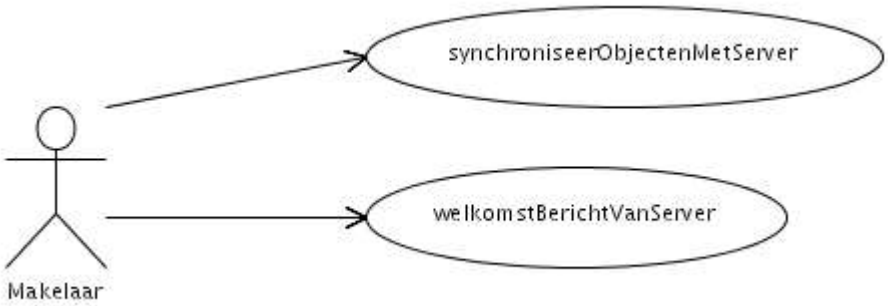
- Het kijken welke objecten zijn veranderd ten opzichte van de server.
- Dan het versturen van objecten die nieuw of veranderd zijn.
- Het verwijderen van objecten van de server die niet meer aanwezig zijn bij de client.
- Het kijken welke foto's vernieuwd of veranderd zijn.
- Het versturen van foto's naar de server.
- Het verwijderen van foto's op de server die niet meer aanwezig zijn bij de client.

4.1.1. Client

Use Cases Diagram

Nadat de makelaar klaar is met het invoeren, veranderen of verwijderen van gegevens, kan de makelaar de gegevens gaan synchroniseren met de server. Zo als in figuur 5 is te zien zijn er twee acties nodig om de gegevens naar de server te sturen.

Maar op de achtergrond gebeuren veel meer dingen. Zoals controle op welke object aanwezig zijn op de server of niet.



Figuur 5, De use cases van FileUpload (client)

Class diagram

Bij de synchronisatie is het belangrijk dat de communicatie wordt verdeeld in zo groot als mogelijke brokken. Daarnaast moet onnodige communicatie vermeden worden omdat communicatie op afstand vertragend werkt.

Objecten die al op de server staan, en actueel zijn, mogen niet nog een keer verzonden worden. Dit zelfde geldt voor de aan de objecten gekoppelde foto's.

Nadat de methode `synchronizeWithServer()` is aangeroepen, gebeuren er op de achtergrond meerdere dingen. Als parameter wordt met deze methode een Set van objecten meegegeven. In deze Set staan alle objecten die gepresenteerd moeten worden op de site van de makelaar.

Voordat de objecten en/of foto's geupload worden, vraagt de client aan de server welke objecten en foto's niet aanwezig zijn of veranderd zijn. Om hier niet te veel communicatie te veroorzaken, worden de vergelijkingen uitgevoerd door gebruik te maken van hashcodes van objecten.

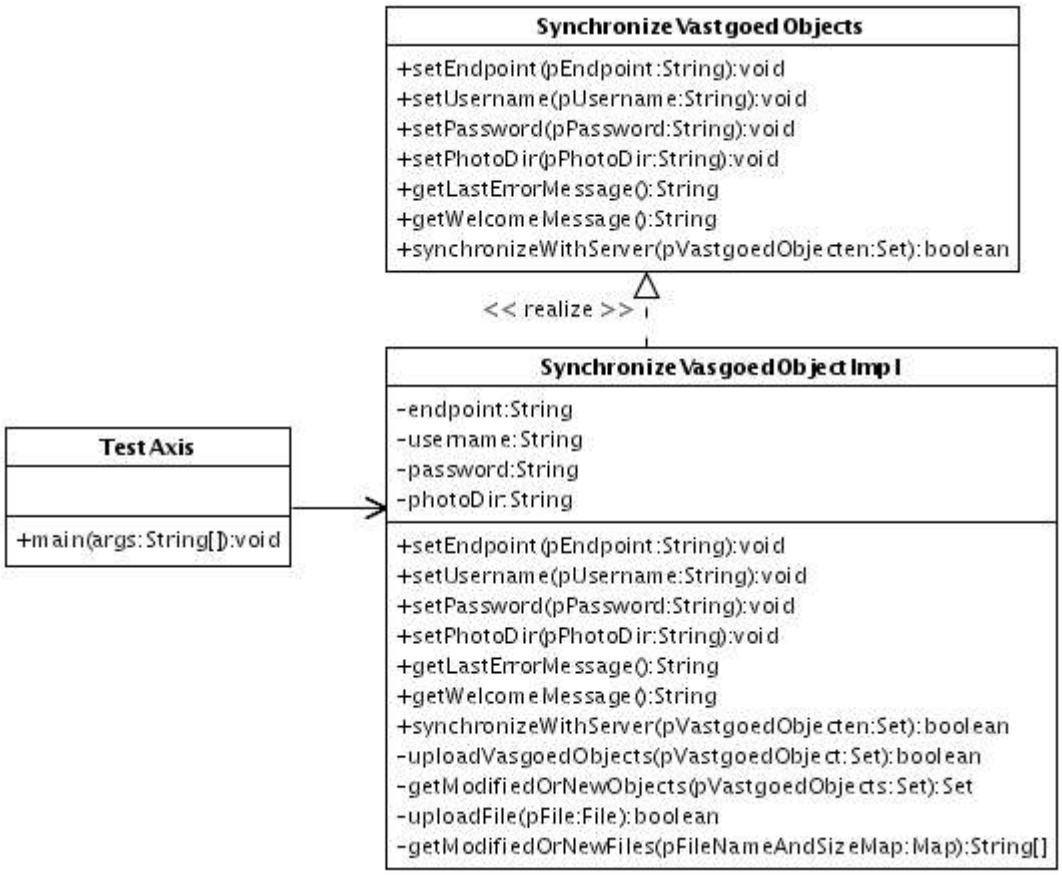
De foto's worden vergeleken aan de hand van de naam en de grootte. De client verwerkt deze informatie en bepaalt welke objecten en foto's naar de server gestuurd moeten worden.

De client heeft vier private methoden die er voor zorgen dat alleen de gewijzigde of nieuwe objecten en/of foto's naar de server verstuurd worden.

- `getModifiedOrNewObjects()`
- `uploadVastgoedObjects()`
- `getModifiedOrNewFiles()`
- `uploadFile()`

Setter methoden worden gebruikt voor het instellen van `endpoint`, `gebruikersnaam` en `wachtwoord`. De methode `getWelcomeMessage()` wordt gebruikt om de client een bericht te sturen.

De methode `getLastErrorMessage()` wordt gebruikt om na een eventueel opgetreden fout de foutmelding uit te lezen.

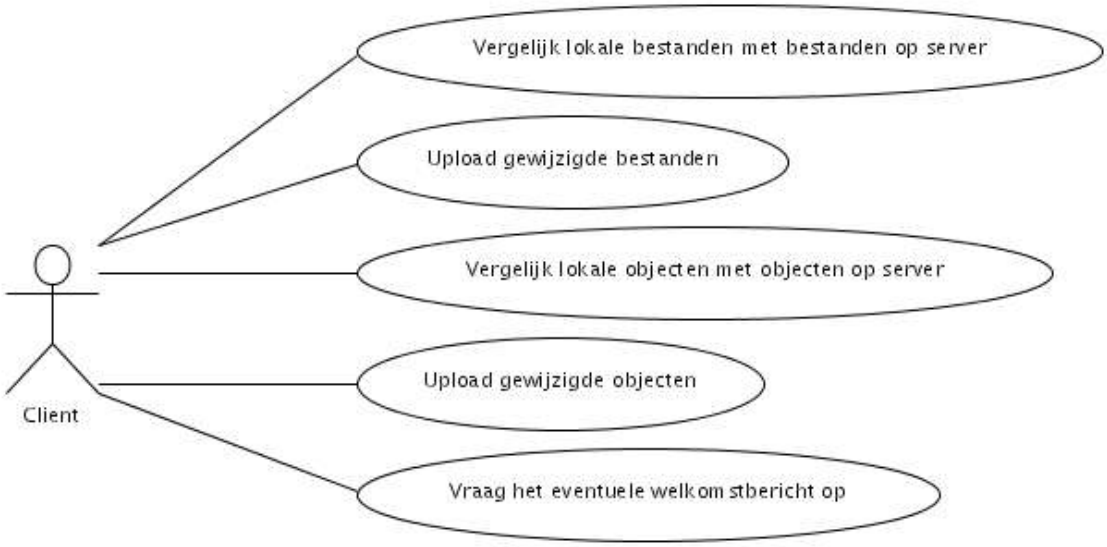


Figuur 6, Het classe diagram van FileUpload

4.1.2. Server

Use Case diagram

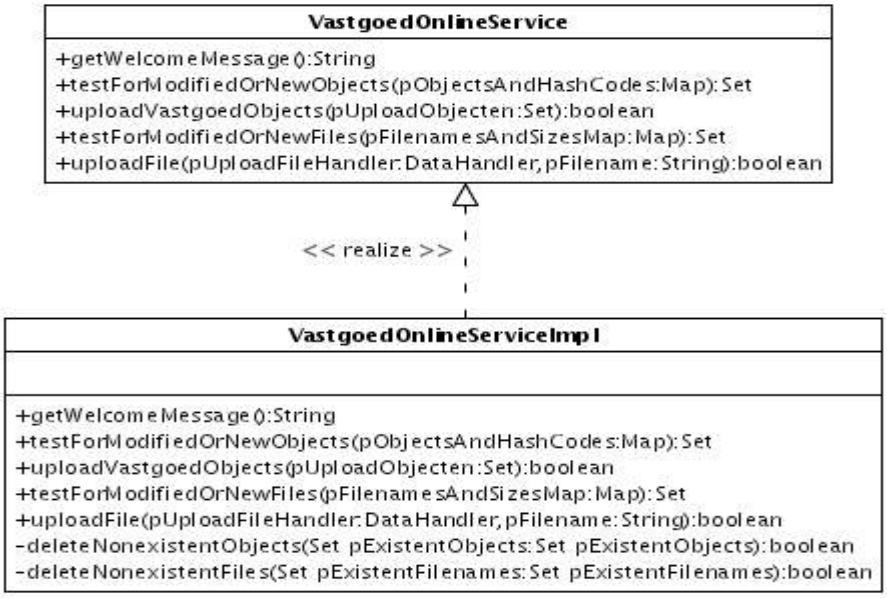
De server krijgt van de client steeds een opdracht die de server moet uitvoeren. Vandaar dat de actor hier de client is. Verder geeft de server de benodigde informatie terug aan de client en zal het de objecten opslaan die binnen komen vanaf de client.



Figuur 7, De use cases van de server

Class diagram

De methode die genoemd zijn in de interface in onderstaande figuur kunnen op afstand aangeroepen worden door de client.



Figuur 8, Het classe diagram van de server

5. TESTEN

5.1. TESTCLASS (TESTAXIS)

Om het opgeleverde deelproduct te kunnen testen, is er een testclass gemaakt. Deze class maakt een aantal objecten aan en probeert vervolgens de objecten naar de server te sturen. Op de server worden ook een aantal objecten gemaakt. Sommige objecten zijn hetzelfde, anderen zijn iets gewijzigd ten opzichte van de client. Alleen de objecten die exact hetzelfde zijn, mogen niet verzonden worden.

Ook worden er een aantal bestanden gemaakt voor foto's. Deze bestanden staan in het VastgoedObject, maar voor het testen zijn er aantal losse bestanden gemaakt. Ook hier geldt dat als het bestand hetzelfde is als op de server, het niet verstuurd mag worden.

5.2. TESTPLAN EN TESTRAPPORT

Alle test cases hebben betrekking op de eis met nummer 3; Fileupload.

Beschrijving	***Instructies***	***Verwachte uitvoer***	***Check***
De authenticatie	Voer de TestAxis uit. Waarbij er een verkeerde username is opgegeven.	Als terugwaarde “null” en een uitleesbare foutmelding.	√
Het welkomstbericht	Voer de TestAxis uit.	Welkomstbericht =”rob”	√
Uploaden van objecten naar webapplicatie	Voer de TestAxis nogmaals uit.	De objecten die geupload zijn, bevinden zich op de server (zichtbaar in log)	√
Geen extra communicatie	Voer de TestAxis nogmaals uit.	Er wordt gedetecteerd dat de objecten al op de server aanwezig zijn. (zichtbaar in log)	√
Verwijderen van objecten	Maak een object aan die niet bestaat bij de client. Voer dan de TestAxis uit.	Uitvoer van de server. Object(en) verwijderd. (zichtbaar in log)	√
Verwijderen van foto's	Maak een foto aan die niet bestaat bij de client. Voer dan de TestAxis uit.	Uitvoer van de server. Foto ('s) verwijderd. (zichtbaar in log)	√

6. SAMENVATTING

De belangrijkste doel van dit deelproduct is om de gegevens tussen een client en een server op een zo efficient mogelijke manier en te verzenden. Verder is er authenticatie aanwezig zijn om te bepalen met welke client de server aan het communiceren is. Deze authenticatie verloopt via de standaard HTTP authenticatie.

Bij het vergelijken van de objecten wordt er gebruik gemaakt van een hashcode. De inhoud van een object wordt als een serie bytes achterelkaar geplakt en vervolgens wordt hieruit de hashcode bepaald. Hierdoor is de hashcode van een object aan de client kant hetzelfde als een object bij de server, tenzij de inhoud van het object niet hetzelfde is.

Foto's worden vergeleken aan de hand van hun bestandsnaam en bestandgrootte.

Objecten of bestanden die niet meer lokaal aanwezig zijn, worden ook verwijderd op de server na een synchronisatie.

Tijdens de communicatie wordt de data verstuurd volgens de SOAP specificatie. De data gaat als een serie XML berichten over de lijn.

Als er tijdens de communicatie een fout optreedt, bijvoorbeeld een server die even niet bereikbaar is, wordt deze netjes afgevangen. Er wordt aan de gebruiker gemeld wat de mogelijke oorzaak is en wat de gebruiker er aan kan doen. (meestal opnieuw proberen)

7. REFERENTIES

[1] **Sun Microsystems**, *The J2EE 1.4 Tutorial*, maart 2004,
http://java.sun.com/j2ee/1.4/docs/tutorial/doc/index.html

[2] **Apache Axis**, *Web Services homepage*, maart 2004,
http://ws.apache.org/axis/

[3] **Javaworld**, *Axis: The next generation of Apache SOAP*, maart 2004,
http://www.javaworld.com/javaworld/jw-01-2002/jw-0125-axis.html

[4] **Java Butique**, *Digging deeper into Apache Axis*, maart 2004,
http://javaboutique.internet.com/tutorials/Axis2-2/

[5] **Java Butique**, *Web Services with Axis*, maart 2004,
http://javaboutique.internet.com/tutorials/Axis/

8. BIJLAGEN

8.1. SERVER-CONFIG.WSDD

VastgoedOnline test Web Service configuratiebestand.

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
     xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
     xmlns:ns1="urn:vastgoedonline">

    <globalConfiguration>
        <parameter name="bijlagen.Directory" value="/tmp"/>
        <parameter name="sendMultiRefs" value="true"/>
        <parameter name="sendXsiTypes" value="true"/>
        <parameter name="bijlagen.implementation"
             value="org.apache.axis.bijlagen.bijlagenImpl"/>
        <parameter name="sendXMLDeclaration" value="true"/>
        <parameter name="axis.sendMinimizedElements" value="true"/>
    </globalConfiguration>

    <handler name="LocalResponder"
         type="java:org.apache.axis.transport.local.LocalResponder"/>
    <handler name="URLMapper"
         type="java:org.apache.axis.handlers.http.URLMapper"/>

    <!-- Eigen implementatie Web Service -->
    <service name="urn:vastgoedonline" provider="java:RPC">
        <parameter name="scope" value="request"/>
        <parameter name="allowedMethods" value="*"/>
        <parameter name="className"
                   value="nl.arsoftware.vastgoedonline.services.
                          VastgoedOnlineService"/>

        <beanMapping qname="vastgoedonline:VastgoedObject"
            xmlns:vastgoedonline="urn:BeanService"
            languageSpecificType="java:nl.arsoftware.vastgoedonline.
                                  domain.VastgoedObject"/> -->
        <typeMapping
             deserializer="org.apache.axis.encoding.ser.
                           JAFDataHandlerDeserializerFactory"
             languageSpecificType="java:javax.activation.DataHandler"
             qname="ns1:DataHandler"
             serializer="org.apache.axis.encoding.ser.
                         JAFDataHandlerSerializerFactory"
            encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/
>

        <requestFlow name="checks">
            <handler type="java:nl.arsoftware.vastgoedonline.services.
                           AuthenticationHandler"/>
        </requestFlow>
```

```
    </service>

    <transport name="http">
        <requestFlow>
            <handler type="URLMapper"/>
            <handler type="java:org.apache.axis.handlers.http.
                          HTTPAuthHandler"/>
        </requestFlow>
    </transport>

    <transport name="local">
        <responseFlow>
            <handler type="LocalResponder"/>
        </responseFlow>
    </transport>
</deployment>
```