EDA095
Web Protocols and Architecture

Pierre Nugues

Lund University
http://www.cs.lth.se/home/Pierre_Nugues/

May 6, 2010

Covers: Chapter 15, pages 493–551, *Java Network Programming*, 3rd ed., Elliotte Rusty Harold

An Example of Protocol: TFTP

The Trivial File Protocol Transfer (TFTP) is a protocol to transfer files
TFTP is a simplified and unconnected FTP.
It is build on top of UDP although an implementation with TCP is possible
The description is available here `http://tools.ietf.org/html/rfc783`
It is an example of unsupported protocol that the URL class fails to locate
The TFTP datagram:

Ethernet	IP	UDP	TFTP

A didactical implementation is available in W. Richard Stevens, *Unix Network Programming*, Prentice-Hall, 1990,
`http://www.kohala.com/start/unp.html`,
`http://en.wikipedia.org/wiki/W._Richard_Stevens`

TFTP: The Packets

RRQ

x01	filename	0	mode	0
2 b.	N bytes	1 b.	N bytes	1 b.

WRQ

x02	filename	0	mode	0
2 b.	N bytes	1 b.	N bytes	1 b.

DATA

x03	block#	data
2 b.	2 b.	0..512 bytes

ACK

x04	block#
2 b.	2 b.

ERROR

x05	Errcode	errstring	0
2 b.	2 bytes	N bytes	1 b.

TFTP: Errors

The mode is one of: `netascii` (lines ending with `\r\n` or `\r\0`), `octet`, or `mail`.
The TFTP protocol defines a set of error values:

Value	Meaning
0	Not defined, see error message (if any).
1	File not found.
2	Access violation.
3	Disk full or allocation exceeded.
4	Illegal TFTP operation.
5	Unknown transfer ID.
6	File already exists.
7	No such user.

TFTP: Data Communication

Sending a file				Receiving a file			
Client			Server	Client			Server
WRQ	→			RRQ	→		
		←	ACK 0			←	DATA 1
DATA 1	→			ACK 1	→		
		←	ACK 1			←	DATA 2
DATA 2	→			ACK 2	→		
		←	ACK 2			←	DATA 3
DATA 3	→			ACK 3	→		
		←	ACK 3			←	DATA 4

...

Modeling the Communications: Finite-State Machines

We can model the behavior of the client and the server using finite–state machines.

Sent	Received				
	RRQ	WRQ	DATA	ACK	ERROR
RRQ			•		•
WRQ				•	•
DATA				•	
ACK			•		
ERROR					•

Table: Client. After Richard Stevens, *Unix Network Programming*, Prentice-Hall, 1990

Modeling the Communications: Finite-State Machines (II)

Sent	Received				
	RRQ	WRQ	DATA	ACK	ERROR
nothing	•	•			
RRQ					
WRQ					
DATA				•	
ACK			•		
ERROR					

Table: Server. After Richard Stevens, *Unix Network Programming*, Prentice-Hall, 1990

Programming TFTP

From Richard Stevens, *Unix Network Programming*, Prentice-Hall, 1990.

- Define functions for all possible transitions: `recv_DATA` for state `[sent = OP_ACK][recv = OP_DATA]`
- `recv_DATA` calls `send_ACK`

```
send_ACK(int blocknum)
{
  stshort(OP_ACK, sendbuff);
  stshort(blocknum, sendbuff + 2);
  sendlen = 4;
  net_send(sendbuff, sendlen);
  op_sent = OP_ACK;
}
```

where we have `extern char sendbuff[]` and

```
#define stshort(sval, addr)
 ( *((u_short *) (addr)) = htons(sval) )
```

HTTP

HTTP is a protocol consisting of pairs: a client request and a server response
It encapsulates data in an envelope, where labels are in plain text
It is based on TCP, which makes the design simpler
The client request consists of:

1. Request header: Method Request-URI HTTP-Version `\r\n`
2. (headers `\r\n`) *
3. `\r\n`
4. message-body

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

Server Response

The server response consists of:

1. Status line: HTTP-Version Status-Code Reason-Phrase `\r\n`
2. (headers `\r\n`) *
3. `\r\n`
4. message-body

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

Client Methods

The client uses eight possible "methods":

- `GET`: retrieves information identified by the Request-URI
- `POST`: sends data to the identified resource
- `HEAD`. Same as `GET` but returns a response consisting of headers (without a message body)
- `PUT`: stores the resource identified by the Request-URI
- `OPTIONS`: returns the methods supported by the server
- `DELETE`: deletes the resource identified by the Request-URI
- `TRACE`: sends back the header to the client.
- `CONNECT`: reserved name to connect to a TCP/IP tunnel

HTTP servers must implement at least `GET` and `HEAD`.

HTTP Request

The request behind the URL `http://www.cs.lth.se/~pierre/index.html` consists of:

1. HTTP method, URL, version

   ```
   GET /~pierre/index.html HTTP/1.1
   ```

2. Sequence of parameter names (46 types) followed by ':' and values – pairs `Name: Value`

   ```
   Accept: text/plain
    ...
   Host: cs.lth.se
   User-Agent: Mozilla/4.0
   ```

3. Empty line: `\r\n`
4. Possibly a message body (data) whose size is given by the `Content-Length` attribute

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

HTTP Response

Servers send a response: header followed by data

1. Protocol, status code, textual phrase

   `HTTP/1.1 200 OK`

2. Sequence of parameter names followed by ':' and values

   ```
   Date: Wed, 05 May 2010 14:42:26 GMT
   Server: Apache/2.2.3 (Red Hat)
   ..
   Connection: close
   Content-Type: text/html; charset=iso-8859-1
   ```

3. Empty line: `\r\n`
4. Data

   ```
   <html>
   ...
   </html>
   ```


URLConnection

`URLConnection` represents a communications link between the application and a URL.
It is created from a URL object that calls `openConnection()`
It is more complex than `openStream()` but more flexible
In fact, `openStream()` is a shortcut that corresponds to `openConnection().getInputStream()`
It enables the programmer to have more control over a connection:

- Access the header fields
- Configure the client properties
- Use more elaborate commands (`POST`, `PUT` for http)
- Write a protocol handler

Reading a URL

```
try {
  URL myDoc = new URL("http://cs.lth.se/");
  URLConnection uc = myDoc.openConnection();
  InputStream is = uc.getInputStream();
  BufferedReader bReader =
    new BufferedReader(new InputStreamReader(is));
  String line;
  while ((line = bReader.readLine()) != null) {
    System.out.println(line);
  }
} catch (Exception e) { e.printStackTrace(); }
```

//ReadURL.java
Nearly the same as in ViewHTML.java except that we have a `URLConnection` object instead of an `InputStream` object.

Reading the Header

The header is part of the HTTP protocol and consists of a list of pairs: parameter/value. There are two main methods to read it:

- `String getHeaderFieldKey(int n)` // the parameter name of the nth header
- `String getHeaderField(int n)` // the parameter of the nth header

Headers have typical parameters: Date, Content type, etc.
There are shortcuts to access them:

- `String getContentEncoding()`
- `String getContentType()`
- `long getDate()`, etc.

Reading the Header (I)

Extracting the complete list:

```
try {
  URL myDoc = new URL("http://cs.lth.se/");
  URLConnection uc = myDoc.openConnection();
  for (int i = 0; ; i++) {
    String header = uc.getHeaderField(i);
    if (header == null) break;
    System.out.println(uc.getHeaderFieldKey(i) + ": "
      + header);
  }
} catch (Exception e) {
  e.printStackTrace();
}
```

// ReadHeader.java

Reading the Header (II)

Extracting selected parameters:

```
try {
  URL myDoc = new URL("http://cs.lth.se/");
  URLConnection uc = myDoc.openConnection();
  System.out.println("Date: " + new Date(uc.getDate()));
  System.out.println("Content type: " + uc.getContentType());
  System.out.println("Content encoding: " +
    uc.getContentEncoding());
  System.out.println("Last modified: " + uc.getLastModified()
} catch (Exception e) {
  e.printStackTrace();
}
```

// ReadHeader2.java

MIME

The MIME (Multipurpose Mail Internet Extensions) is a tag to identify the content type. RFC 2045 and 2046 (`http://tools.ietf.org/html/rfc2045`)
MIME defines a category and a format: a type and a subtype
Useful MIME types are `text/html`, `text/plain`, `image/gif`, `image/jpeg`, `application/pdf`, and so on.
In ReadHeader.java, let's replace

```
URL myDoc = new URL("http://cs.lth.se/");
```

with

```
URL myDoc = new URL("http://fileadmin.cs.lth.se/cs/Bilder/
  Grundplatta-2.jpg");
```

HTTP servers should send a content type together with data. It is not always present however.
Sometimes, the client has to guess using `guessContentTypeFromStream()`

Downloading Text (and Ignoring the Rest)

```
public ArrayList<URL> readURL(URL url) {
  LinkGetter callback = null;
  try {
    URLConnection uc = url.openConnection();
    String type = uc.getContentType().toLowerCase();
    // We read only text pages
    if ((type != null) && !type.startsWith("text/html")) {
      System.out.println(url + " ignored. Type " + type);
      return null;
    }
    ParserGetter kit = new ParserGetter();
    HTMLEditorKit.Parser parser = kit.getParser();
...
}
```

HTML Forms

An Elementary Form

A text box: My text

Radio buttons:

- FM
- LW
- SW

A drop-down list:

Low

Send! Cancel

HTML Code

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=UTF-8"/>
    <title>Testing HTML forms</title>
  </head>
  <body>
    <h1>An Elementary Form</h1>
    <form action="http://localhost:25001/program.sh"
      method="post">
      <p>A text box: <input type="text" name="name" value=""
        size="30"/></p>
      <hr/>
```


HTML Code

```
<p>Radio buttons:</p>
<ul>
  <li><input type="radio" name="buttons" value="FM"
    checked="checked"/>FM</li>
  <li><input type="radio" name="buttons" value="LW"/>
    LW
  </li>
  <li><input type="radio" name="buttons" value="SW"/>
    SW
  </li>
</ul>
```


HTML Code

```
      <p>A drop-down list:</p>
      <p>
        <select name="dropdown">
          <option selected="selected">Low</option>
          <option>Medium</option>
          <option>High</option>
        </select>
      </p>
      <hr/>
      <p>
        <input type="submit" value="Send!"/>
        <input type="reset" value="Cancel"/>
      </p>
    </form>
  </body>
</html>
```

HTTP Request with POST

To send data URL `http://www.cs.lth.se/~pierre/prog.sh`, the request consists of:

1. HTTP method, URL, version

   `POST /~pierre/prog.sh HTTP/1.0`

2. Sequence of parameter names (46 types) followed by ':' and values – pairs Name: Value

   ```
   Accept: text/plain
    ...
   Host: www.cs.lth.se
   User-Agent: Mozilla/4.0
   ```

3. Empty line: `\r\n`
4. Data length should match the `Content-Length` attribute

RFC 2616 (`http://www.ietf.org/rfc/rfc2616.txt`)

An Example of HTTP Request with POST

```
POST /program.sh HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6;
AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0
Safari/528.16
Content-Type: application/x-www-form-urlencoded
Accept: application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Origin: file://
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Content-Length: 36
Connection: keep-alive
Host: localhost:25001

name=My+text&buttons=FM&dropdown=Low
```


An Example of HTTP Request with GET

```
URL: http://localhost:25001/program.sh?name=My+text&buttons=FM
&dropdown=Low
GET /program.sh?name=My+text&buttons=FM&dropdown=Low HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_6;
 fr-fr) AppleWebKit/528.16 (KHTML, like Gecko) Version/4.0
 Safari/528.16
Accept: application/xml,application/xhtml+xml,text/html;q=0.9,
text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: fr-fr
Accept-Encoding: gzip, deflate
Connection: keep-alive
Host: localhost:25001
```


Configuring the Parameters

A set of `URLConnection` methods enables a program to read and modify the connection's request parameters:

- `protected boolean connected` //false
- `protected boolean doInput` // true
- `protected boolean doOutput` // false
- `protected URL url`, etc.

The methods to read and modify the connection are:

- `URL getURL()`
- `boolean getDoInput()`
- `void setDoInput(boolean)`
- `String getRequestProperty(String key)`
- `void setRequestProperty(String key, String value)`

etc.

Getting the Parameters

```
try {
  URL myDoc = new URL("http://cs.lth.se/");
  URLConnection uc = myDoc.openConnection();
  System.out.println("URL: " + uc.getURL());
  System.out.println("Do Input: " + uc.getDoInput());
  System.out.println("Do Output: " + uc.getDoOutput());
  uc.setDoOutput(true);
  System.out.println("Do Output: " + uc.getDoOutput());
} catch (Exception e) {
  e.printStackTrace();
}
```

// ReadParameters.java

Faking the Lizard

A naïve Google query from Java fails miserably:

```
Server returned HTTP response code: 403 for URL:
http://www.google.com/search?hl=fr&source=hp&q=nugues&
  lr=&aq=f&aqi=g4&aql=&oq=&gs_rfai=
```

Google sets constraints on the user agent.
It is possible to remedy this.
Just set a user agent corresponding to a known browser


```
uc.setRequestProperty("User-Agent", "Mozilla/5.0");
```


Using the POST Command

The two main commands of a http client are GET and POST
GET sends parameters as an extension of a URL address
They are visible to everybody
How to send data with POST?
Programs FormPoster.java and QueryString.java from Elliotte Rusty Harold, *Java Network Programming*, page 519 and page 212 are examples of it. (http://www.cafeaulait.org/books/jnp3/examples/15/)
They form a client that works in conjunction with a server and formats a query.
The query is sent back by the server

Formatting a Query

QueryString.java encodes and formats a query.
The pairs of parameter names (keys) and values are separated with &
In the example, we send:

- Name: Elliotte Rusty Harold
- Email: elharo@metalab.unc.edu

Using the POST Command

Switching from GET to POST is done implicitly through setDoOutput()

```
URLConnection uc = url.openConnection();
uc.setDoOutput(true);
OutputStreamWriter out =
  new OutputStreamWriter(uc.getOutputStream(), "ASCII");
out.write(query.toString());
out.write("\r\n");
out.flush();
out.close();
```

The client header is sent automatically

Using the POST Command (II)

HttpURLConnection is a subclass designed to carry out HTTP interaction
The POST method is more explicit with it and the setRequestMethod()

```
HttpURLConnection uc =
  (HttpURLConnection) url.openConnection();
uc.setRequestMethod("POST");
uc.setDoOutput(true);
```

(FormPoster2.java)

HttpURLConnection

Possible requests with HttpURLConnection are:

- GET // default download
- POST
- HEAD //same as GET but return the header only
- OPTIONS //lists the possible commands
- PUT // Upload a file
- DELETE // delete a file
- TRACE // send back the header

It makes provision to manage HTTP protocol codes

FTP

```
//URL url = new URL("ftp://username:password@ftp.whatever.com/

URL url = new URL("ftp://ftp:Pierre.Nugues%40cs.lth.se@ftp.si
URLConnection uc = url.openConnection();
InputStream is = uc.getInputStream();
BufferedReader bReader = new BufferedReader(new InputStreamRe
String line;
while ((line = bReader.readLine()) != null) {
  System.out.println(line);
}
```

//ReadURLftp.java

Extending URL Classes

We have used the URL class with supported protocols (HTTP, FTP) and to transfer text.
It is possible to extend it to new or unsupported protocols and to other media.
Most implementations divide it into two tasks:

- Handling protocols
- Handling content

Both tasks are described in Chapter 16 and 17 of Eliotte Rusty Harold, *Java Network Programming*.

Protocol and Content Handling

The URL methods fetch data streams without structure
This is insufficient for most applications that have to

- Handle the connection between the parties (protocol handler)
- Recover the structure from the data and create the respective objects (content handler)

```
URL url = new URL("tftp://www.cs.lth.se/first/house.jpg");
ImageProducer img = (ImageProducer) url.getContent();
```

Solve two problems:

- Handle the protocol (tftp)
- Handle the image (gif suffix or MIME type)

Protocol Handlers

The protocol handler uses the following classes:

- `URLStreamHandler` parses the URL and creates an appropriate `URLConnection` corresponding to the protocol
- `URLConnection` handles the connection physically. It creates an `InputStream` able to read data.
- `URLStreamHandlerFactory` loads the protocol handler. Otherwise, Java will use the default protocol handlers

From the Java API: Protocol handlers for the following protocols are guaranteed to exist on the search path: http, https, ftp, file, and jar

REST Architecture

REST – representation state transfer – An a posteriori model of the web: clients, servers, and HTTP
RESTful architecture implicitly means: the client-server transactions based on three standards
HTTP:

- Transfer protocol of the web
- On top of TCP/IP
- Pairs of requests from clients and responses from servers

URI/URLs:

- A way to name and address objects on the net

HTML/XML

HTTP Methods and REST

In the context of REST, HTTP methods are used with a different meaning. This defines the interaction protocol or API
Amazon S3 is an example (from RESTful web services, Chap. 3, O'Reilly)
It uses two types of objects: buckets (a folder or a collection) and objects and four methods: GET, HEAD, PUT, and DELETE

	GET	HEAD	PUT	DELETE
Bucket	List content		Create bucket	Delete bucket
Object	Get data and metadata	Get metadata	Set values and metadata	Delete object

One example among others...

REST and Sesame

Sesame extends the REST protocol to manage graphs:

- GET: Fetches statements from the repository.
- PUT: Updates data in the repository, replacing any existing data with the supplied data. The data supplied with this request is expected to contain an RDF document in one of the supported RDF formats.
- DELETE: Deletes statements from the repository.
- POST: Performs updates on the data in the repository. The data supplied with this request is expected to contain either an RDF document or a special purpose transaction document. In case of the former, the statements found in the RDF document will be added to the repository. In case of the latter, the updates specified in the transaction document will be executed.

REST Examples

Get all repositories (tuple query):

```
curl -X GET -H "Accept: application/sparql-results+xml"
  http://asimov.ludat.lth.se/openrdf-sesame/repositories
```

Delete all statements in the repository:

```
curl -X DELETE
  http://asimov.ludat.lth.se/openrdf-sesame/repositories/sandb
```

SPARQL queries is also straightforward. Some examples
A SELECT query 'SELECT ?s ?p ?o WHERE ?s ?p ?o' (tuple query):

```
curl -X GET -H "Accept: application/sparql-results+json"
http://asimov.ludat.lth.se/openrdf-sesame/repositories/sandbo
?query=SELECT+%3fs+%3fp+%3fo+WHERE+%7b%3fs+%3fp+%3fo%7d
```