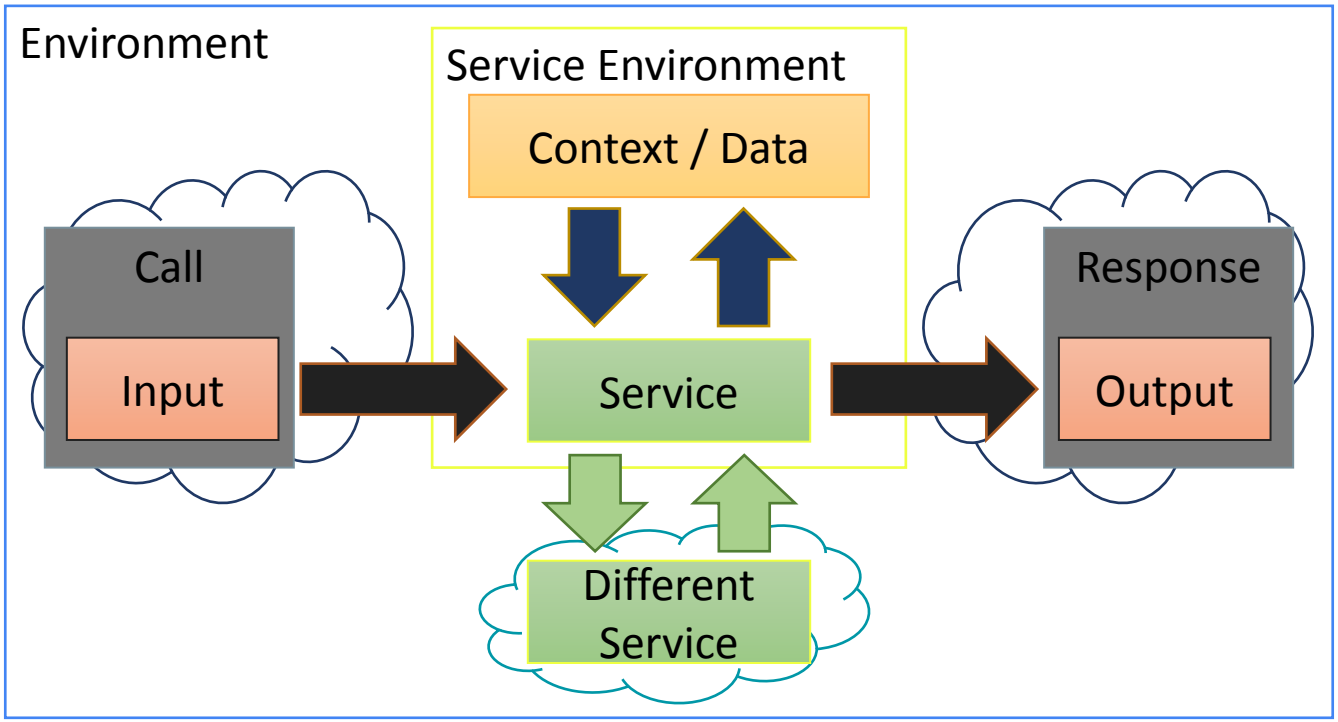
UNTERNEHMENS INFORMATIONSSYSTEME

WEBSERVICES
ARCHITEKTUR UND BEISPIELE

Dr.rer.soc.oec. Maxim Vidgof, MSc BSc, Workflow Systems and Technology, Fakultät für Informatik
Folien mit freundlicher Genehmigung von emer. o. Univ.-Prof. Dr.techn Dimitris Karagiannis, Mag. Dr. Wilfrid Utz und Alexander Völz, B.Sc. MSc

Web Service – Simple Definition

- A digital component that can be accessed remotely and provides some benefit.
  - Additionally there are different parts in the environment which "connect" to the service.



Web Service: Examples

- "Context" in and output: https://www.timeanddate.com/worldclock/ – a list of the local times of several cities, presented in HTML
- Output only: http://www.clocktab.com/ – the current time of the computer (sends the JavaScript which is executed locally)
- Input and "Context" out (and possible Output): (https://www.dropbox.com/developers-v1/core/start/java) Dropbox API uploadFile – uploads a file on Dropbox. The method itself does return a value, but it's not the thing we are after (i.e. the returned thing is not the benefit we want).
- Input and Output (and possible "Context"): https://www.google.at/maps/dir/Währinger+Str.+29,+1090+Wien/Universitätsring+1,+Wien/ – The route from one address to a different address. Note that in this case the inputs are part of the URL.

Service – Categorization

- Services can be categorized based on different aspects:
  - Who is meant to consume the service? Human? Computer? Agent?
  - What technologies are used to communicate with the service? REST? SOAP? Plain old HTTP? Command-line? Library?
  - Is the service state-full or state-less?
  - Where is the service located? Local? Intranet? Internet? The Web?
- Having a **Web** Service answers or at least restricts some of the possible answers.

Web Services

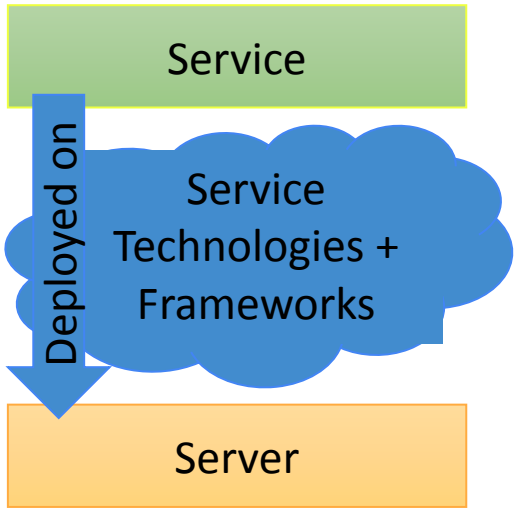
- Just like other applications, programs etc. there are many aspects that have to be dealt with in services, like:
  - Authentication
  - Security
  - Accessibility / Interfaces
  - Persistence
  - Resource usage
  - Logging
  - Deployment
  - …
- However, the aspects where services mostly differ from other applications is “Accessibility / Interfaces” (i.e. how to use the service) and “Deployment” (i.e. on a Server).

Web Service vs. Server

- There is also the question where does Server end and Web Service begin.
  - A Web Service needs to be deployed to work, so without a Server it won't work.
    - "If a tree falls in a forest and no one is around to hear it, does it make a sound?" → "If a Web Service exists and it isn't running on a Server, does it really provide a benefit?"
  - A Server can support the service by taking over certain tasks, like authentication, pre-processing of requests and responses (e.g. extract query parameters, message de-/encoding for SSL), logging of accesses etc. That's where the line can start to blur a bit.
    - Depending on the used server where the service is to be deployed, it might actually require the service to take care of those things.
    - Maybe it could be seen as the Server providing services to the Web Service.
- Maybe a metaphor: Think of it like a meal, where the Server is the plate, the Service is the food on the plate, and cutlery is the "additional support" provided by the server.

Web Service vs. Server

- Depending on the used technologies and frameworks certain tasks are/have to be taken care of either by the server or the service (e.g. reading the parameters passed in a request)
  - For example using the JRE built in HTTP server the service has to write the JSON to the response body.
  - With Spring framework a method returning an object is specified to be available and Spring takes care of the marshalling (Object to JSON) and writing the response.



Web Service vs. Server

Note that the Server is both the hardware and the “server software”

Programmed Service Code

Server

Service

Very general service tasks (e.g. accept HTTP requests)

JRE Http Server

Apache Tomcat

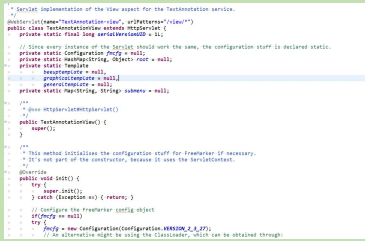
Spring + Tomcat

Very specific service tasks (e.g. calculate a Fibonacci number)

Web Service vs. Server

Note that the Server is both the hardware and the “server software”

Programmed Service Code



Server

Service

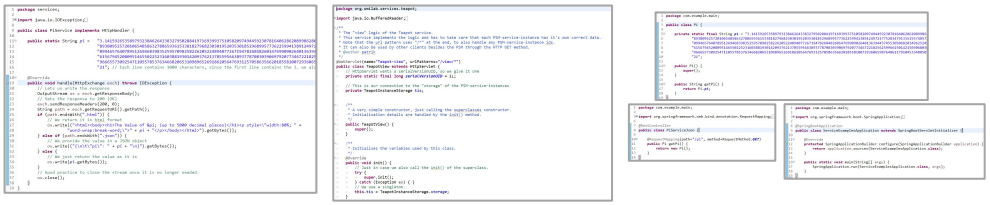
Very general service tasks (e.g. accept HTTP requests)

JRE Http Server

Apache Tomcat

Spring + Tomcat

Very specific service tasks (e.g. calculate a Fibonacci number)



Web Service vs. Server

- A simple distinction:
  - A Web Service is the logic, the algorithms and code that implements the provided benefit,
  - while the Server is the part of the infrastructure (real and/or virtual) where the Web Service is executed.
  - And the configuration takes care that the two things work together (wherever it may be located).
  - Something along the lines of: “The Server is the human, and the Web Service is the knowledge the human possesses”.
- This way there are also many “Web Service things”:
  - The “conceptual” Web Service
  - The Web Service implementation
  - The Web Service “object” that is actually deployed and running and performs the work
  - …

Web Service – By Example (1)

- A simple service that returns the number π.
- The benefit here is that it provides the number with 5000 decimal places, thus a very precise value for π.
- The Service is implemented in Java, aiming to be deployed on the HTTP Server provided by the JRE (Java Runtime Environment).

Web Service – By Example (1)

```
1  package services;
2
3  import java.io.IOException;
8
9  public class PiService implements HttpHandler {
10
11     public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862800
12                                 "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497 2
13                                 "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639 4
14                                 "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182 5
15                                 "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606 5
16                                 "21"; // Each line contains 1000 characters, since the first line contains the 3. we als
17
18     @Override
19     public void handle(HttpExchange exch) throws IOException {
20         // Lets us write the response
21         OutputStream os = exch.getResponseBody();
22         // Sets the response to 200 (OK)
23         exch.sendResponseHeaders(200, 0);
24         String path = exch.getRequestURI().getPath();
25         if (path.endsWith(".html")) {
26             // We return it in html format
27             os.write(("<html><body><h1>The Value of &pi; (up to 5000 decimal places)</h1><p style=\"width:80%; " +
28                     "word-wrap:break-word;\">" + pi + "</p></body></html>").getBytes());
29         } else if (path.endsWith(".json")) {
30             // We provide the value in a JSON object
31             os.write(("{\n\t\"pi\": " + pi + "\n}").getBytes());
32         } else {
33             // We just return the value as it is
34             os.write(pi.getBytes());
35         }
36         // Good practice to close the stream once it is no longer needed.
37         os.close();
38     }
39 }
```

Web Service – By Example (1)

This class implements the service

```
1  package services;
2
3  import java.io.IOException;
8
9  public class PiService implements HttpHandler {
10
11     public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862800
12                                 "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497 2
13                                 "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639 4
14                                 "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182 5
15                                 "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606 5
16                                 "21"; // Each line contains 1000 characters, since the first line contains the 3. we als
17
18     @Override
19     public void handle(HttpExchange exch) throws IOException {
20         // Lets us write the response
21         OutputStream os = exch.getResponseBody();
22         // Sets the response to 200 (OK)
23         exch.sendResponseHeaders(200, 0);
24         String path = exch.getRequestURI().getPath();
25         if (path.endsWith(".html")) {
26             // We return it in html format
27             os.write(("<html><body><h1>The Value of &pi; (up to 5000 deci
28                     "word-wrap:break-word;\">" + pi + "</p></body></html>
29         } else if (path.endsWith(".json")) {
30             // We provide the value in a JSON object
31             os.write(("{\n\t\"pi\": " + pi + "\n}").getBytes());
32         } else {
33             // We just return the value as it is
34             os.write(pi.getBytes());
35         }
36         // Good practice to close the stream once it is no longer needed.
37         os.close();
38     }
39 }
```

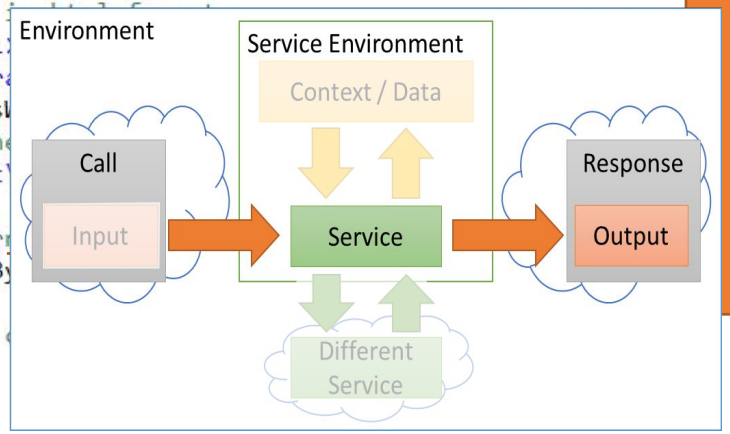
Communication - Output

Web Service – By Example (1)

This class implements the service

```
1  package services;
2
3  import java.io.IOException;
8
9  public class PiService implements HttpHandler {
10
11     public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862800
12                                 "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497 2
13                                 "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639 4
14                                 "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182 5
15                                 "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606 5
16                                 "21"; // Each lir                                    ne con ains the 3. we als
17
18     @Override
19     public void handle(HttpExchange exch) throws
20         // Lets us write the response
21         OutputStream os = exch.getResponseBody();
22         // Sets the response to 200 (OK)
23         exch.sendResponseHeaders(200, 0);
24         String path = exch.getRequestURI().getPath();
25         if (path.endsWith(".html")) {
26             // We return it
27             os.write(("<html
28                     "word-wr
29         } else if (path.ends
30             // We provide the
31             os.write(("{\n\t
32         } else {
33             // We just retur
34             os.write(pi.getBy
35         }
36         // Good practice to
37         os.close();
38     }
39 }
```

This service only provides an output. Everything needed is contained in the service.



Communication - Output

Web Service – By Example (1)

This class implements the service

```
1  package services;
2
3  import java.io.IOException;
8
9  public class PiService implements HttpHandler {
10
11     public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862800
12                                 "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497 2
13                                 "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639 4
14                                 "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182 5
15                                 "96665573092547110557853763466820653109896526918620564769312570586356 20185581007293606 5
16                                 "21"; // Each line contains 1000 characters, since the first line con ins the 3. we als
17
18     @Override
19     public void handle(HttpExchange exch) throws IOException {
20         // Lets us write the response
21         OutputStream os = exch.getResponseBody();
22         // Sets the response to 200 (OK)
23         exch.sendResponseHeaders(200, 0);
24         String path = exch.getRequestURI().getPath();
25         if (path.endsWith(".html")) {
26             // We return it in html format
27             os.write(("<html><body><h1>The Value of &pi; (u
28                     "word-wrap:break-word;\">" + pi + "</p></body></html>
29         } else if (path.endsWith(".json")) {
30             // We provide the value in a JSON object
31             os.write(("{\n\t\"pi\": " + pi + "\n}").getBytes());
32         } else {
33             // We just return the value as it is
34             os.write(pi.getBytes());
35         }
36         // Good practice to close the stream once it is no longer needed.
37         os.close();
38     }
39 }
```

Output for Human

Output for Machine

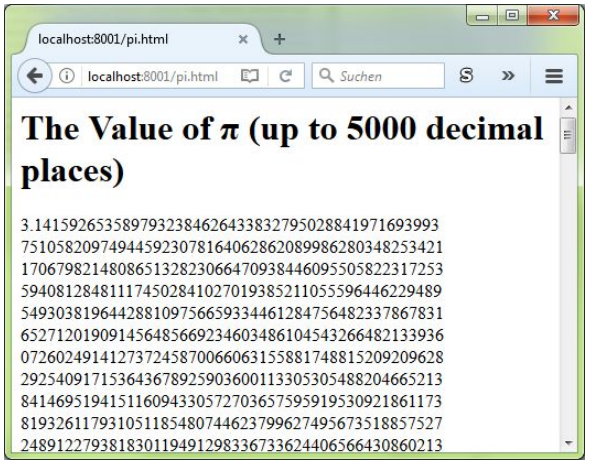
Default Output

Web Service – By Example (1)

Output for Human

Output for Machine

Default Output


localhost:8001/pi.html
localhost:8001/pi.html
Suchen
The Value of π (up to 5000 decimal places)
3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384460955058223172535940812848111745028410270193852110555964462294895493038196442881097566593344612847564823378678315
65271201909145648566923460348610454326648213393607260249141273724587006606315588174881520920962829254091715364367892590360011330530548820466521384146951941511609433057270365759591953092186117381932611793105118548074462379962749567351885752724891227938183011949129833673362440656643086021394946395224737190702179860943702770539217176293176752384674818467669405132000568127145263560827785771342757789609173637178721468440901224953430146549585371050792279689258923542019956112129021960864034418159813629774771309960518707211349999998372978049951059731732816096318595024459455346908302642522308253344685035261931188171010003137838752886587533208381420617177669147303598253490428755468731159562863882353787593751957781857780532171226806613001927876611195909216420198938095257201065485863278865936153381827968230301952035301852968995773622599413891249721775283479131515574857242454150695950829533116861727848912279381830119491298336733624406566430860213949463952247371907021798609437027705392171762931767523846748184676694051320005681271452635608277857713427577896091736371787214684409012249534301465495853710507922796892589235420199561121290219608640344181598136297747713099605187072113499999983729780499510597317328160963185950244594553469083026425223082533446850352619311881710100031378387528865875332083814206171776691473035982534904287554687311595628638823537875937519577818577805321712268066130019278766111959092164201989380952572010654858632788659361533818279682303019520353018529689957736225994138912497217752834791315155748572424541506959508295331168617278558907509838175463746493931925506040092770167113900984882401285836160356370766010471018194295559619894676783749494825531" />

HTML

JSON

Text

Since it is implemented as a REST Web Service, a Web Browser is used in this example to access it.

Web Service – By Example (1) – Deployment

- Following is the code that actually “creates” the server, deploys the services on it (deploys also the other “By Example” services that are described later) and then starts the server.
  - It creates an HTTP Server object (provided by the JRE),
  - adds objects of the implemented services to specific sub-paths of the server and
  - starts the server.

Web Service – By Example (1) – Deployment

```java
package main;

import java.net.InetSocketAddress;

/**
 * This class creates a Java Server (the one built-in the JRE) and "deploys" the services on it.
 * @author patrik
 */
public class ServiceExampleServer {

    public static final int DEFAULT_SOCKET = 8001;

    public static void main(String[] args) {
        try {
            // If no argument has been passed, then the default socket will be used
            int usedsocket = DEFAULT_SOCKET;
            if(args.length>0)
                usedsocket = Integer.valueOf(args[0]);
            // Creates the server represented by an object
            HttpServer server = HttpServer.create(new InetSocketAddress(usedsocket), 0);
            // "Deploys" the services on the server at specific URLs
            server.createContext("/theanswer", new TheAnswerService());
            server.createContext("/pi", new PiService());
            server.createContext("/time", new CurrentTimeService());
            server.createContext("/fibonacci", new FibonacciNumberService());
            server.createContext("/stopwatch", new StopwatchService("/stopwatch"));
            // Starts the server. It will run until the application is stopped (e.g. CTRL+C)
            server.start();
            System.out.println("Server started");
            System.out.println("Try the available contexts with .html and .json");
            System.out.println("\nAvailable contexts (relative paths):");
            System.out.println("/theanswer");
            System.out.println("/pi");
            System.out.println("/time");
            System.out.println("/fibonacci");
            System.out.println("/stopwatch");
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Web Service – By Example (1) – Deployment

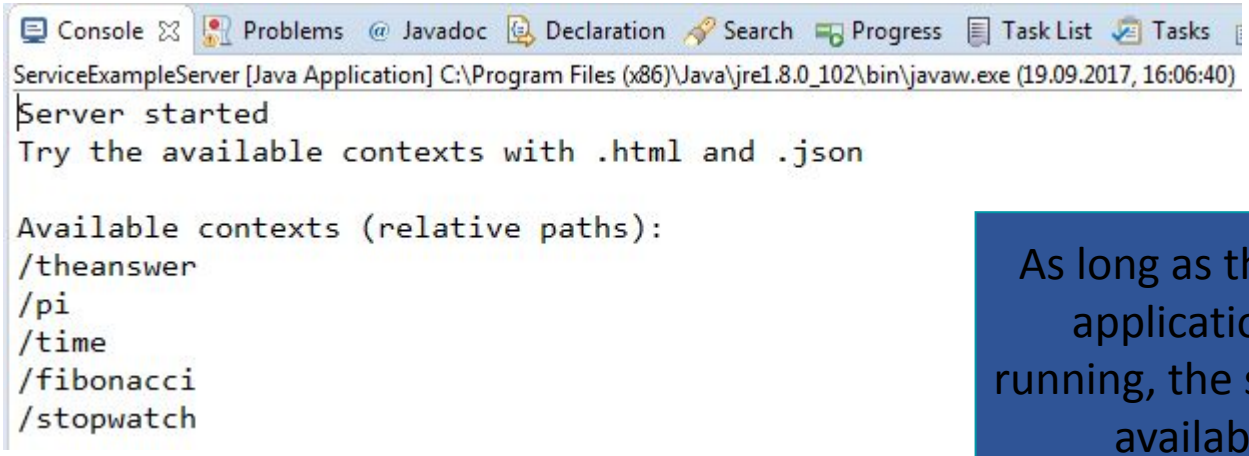
```java
package main;

import java.net.InetSocketAddress;

/**
 * This class creates a Java Server (the one built-in the JRE) and "deploys" the services on it.
 * @author patrik
 */
public class ServiceExampleServer {

    public static final int DEFAULT_SOCKET = 8001;

    public static void main(String[] args) {
        try {
            // If no argument has been passed, then the default socket will
            int usedsocket = DEFAULT_SOCKET;
            if(args.length>0)
                usedsocket = Integer.valueOf(args[0]);
            // Creates the server represented by an object
            HttpServer server = HttpServer.create(new InetSocketAddress(use
            // "Deploys" the services on the server at specific URLs
            server.createContext("/theanswer", new TheAnswerServ
            server.createContext("/pi", new PiService());
            server.createContext("/time", new CurrentTimeService());
            server.createContext("/fibonacci", new FibonacciNumberService(
            server.createContext("/stopwatch", new StopwatchService("/stopw
            // Starts the server. It will run until the application is stop
            server.start();
            System.out.println("Server started");
            System.out.println("Try the availa                    uts with .ht
            System.out.println("\nAvailable contexts (rel
            System.out.println("/theanswer");
            System.out.println("/pi");
            System.out.println("/time");
            System.out.println("/fibonacci");
            System.out.println("/stopwatch");
            System.out.println();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Here the server object is created and the individual Services are “deployed” at specific paths. (Yes, we also deploy some more example services here)

Start the server

The rest are some helpful outputs to the console.

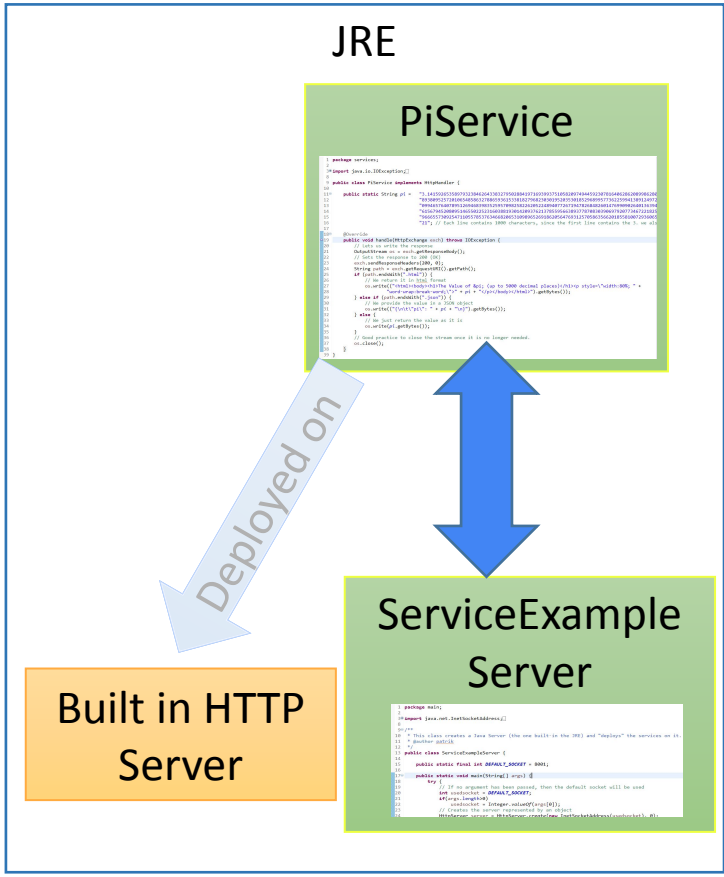
Web Service – By Example (1) – Deployment



As long as the Java application is running, the server is available

Web Service – By Example (1) – Deployment

- Deployment of the specific Service
  - The creation of the server and deployment of services is handled through its own class + main method (ServiceExampleServer).
  - The ServiceExampleServer uses the built in HTTP Server of JRE.
  - The PiService and ServiceExampleServer in the end run in the JRE.
  - Executing the ServiceExampleServer makes the whole thing work.
  - When the execution stops → the server stops too.



Web Service – By Example (1) – Jetty (embedded)

- Same as the previous service with one difference:
  - Instead of using the Http Server available in the JRE we will embed a Jetty server in our application.

Web Service – By Example (1) – Jetty (embedded)

```
1  package services;
2
3  import java.io.IOException;
12
14 * Provides the value of pi including the first 5000 decimal places.
17 public class PiService extends AbstractHandler {
18
19     public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862
20                                 "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497
21                                 "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639
22                                 "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182
23                                 "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606
24                                 "21"; // Each line contains 1000 characters, since the first line contains the 3. we als
25
26     public PiService() {
27         super();
28     }
29
30     @Override
31     public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)
32             throws IOException, ServletException {
33
34         // Set the response code
35         response.setStatus(HttpServletResponse.SC_OK);
36
37         PrintWriter out = response.getWriter();
38         if("/html".equals(target)) {
39             response.setContentType("text/html");
40             // We return it in html format
41             out.write("<html><body><h1>The Value of &pi; (up to 5000 decimal places)</h1><p style=\"width:80%; " +
42                     "word-wrap:break-word;\">" + pi + "</p></body></html>");
43         } else if ("/json".equals(target)) {
44             response.setContentType("application/json");
45             // We provide the value in a JSON object
46             out.write("{\n\t\"pi\": \"" + pi + "\"\n}");
47         } else {
48             response.setContentType("text/plain");
49             // We just return the value as it is
50             out.write(PiService.pi);
51         }
52         out.close();
53         baseRequest.setHandled(true);
54     }
55 }
```

Web Service – By Example (1) – Jetty (embedded)

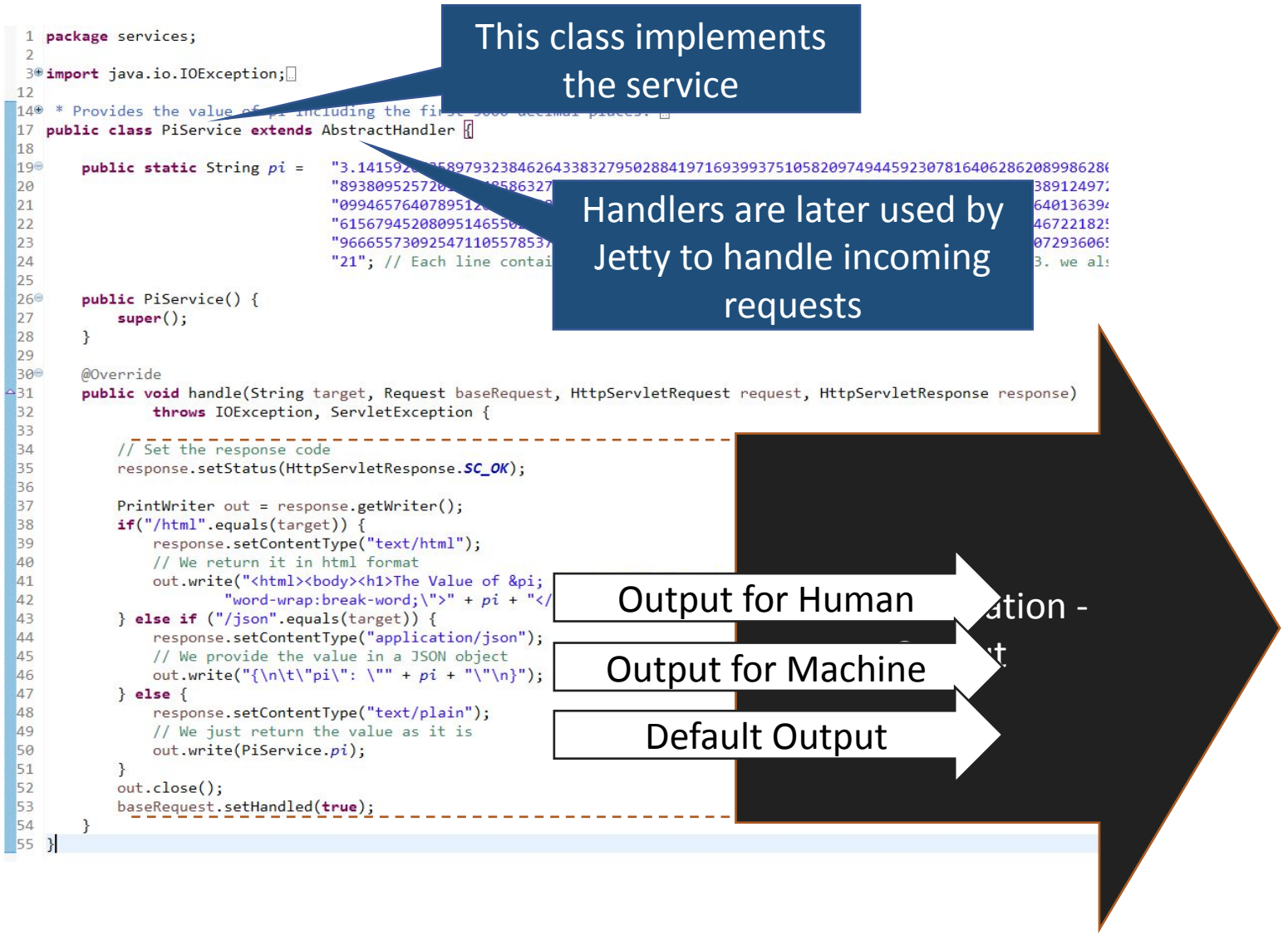
```java
package services;

import java.io.IOException;

 * Provides the value of pi including the first 5000 decimal places.
public class PiService extends AbstractHandler {

    public static String pi =   "3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280
                                "8938095257201…8586327…389124972
                                "0994657640789512…640136394
                                "6156794520809514655…467221825
                                "96665573092547110557853…072936065
                                "21"; // Each line contai… 3. we als

    public PiService() {
        super();
    }

    @Override
    public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {

        // Set the response code
        response.setStatus(HttpServletResponse.SC_OK);

        PrintWriter out = response.getWriter();
        if("/html".equals(target)) {
            response.setContentType("text/html");
            // We return it in html format
            out.write("<html><body><h1>The Value of &pi; (up to 5000 decimal p
                    "word-wrap:break-word;\">" + pi + "</p></body></html>");
        } else if ("/json".equals(target)) {
            response.setContentType("application/json");
            // We provide the value in a JSON object
            out.write("{\n\t\"pi\": \"" + pi + "\"\n}");
        } else {
            response.setContentType("text/plain");
            // We just return the value as it is
            out.write(PiService.pi);
        }
        out.close();
        baseRequest.setHandled(true);
    }
}
```

This class implements the service

Handlers are later used by Jetty to handle incoming requests

Communication - Output

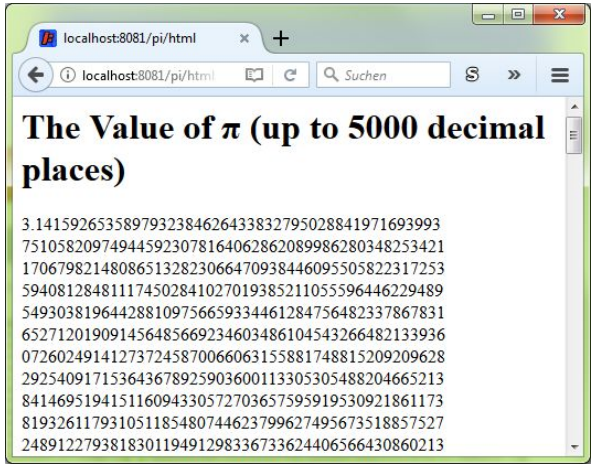
Web Service – By Example (1) – Jetty (embedded)




```java
package services;

import java.io.IOException;

/**
 * Provides the value of pi including the first 5000 decimal places.
 */
public class PiService extends AbstractHandler {

    public static String pi =   "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803"
                              + "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497"
                              + "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639"
                              + "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182"
                              + "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606"
                              + "21"; // Each line contains 100 digits. we als

    public PiService() {
        super();
    }

    @Override
    public void handle(String target, Request baseRequest, HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException {

        // Set the response code
        response.setStatus(HttpServletResponse.SC_OK);

        PrintWriter out = response.getWriter();
        if("/html".equals(target)) {
            response.setContentType("text/html");
            // We return it in html format
            out.write("<html><body><h1>The Value of &pi;
                    "word-wrap:break-word;\">" + pi + "</
        } else if ("/json".equals(target)) {
            response.setContentType("application/json");
            // We provide the value in a JSON object
            out.write("{\n\t\"pi\": \"" + pi + "\"\n}");
        } else {
            response.setContentType("text/plain");
            // We just return the value as it is
            out.write(PiService.pi);
        }
        out.close();
        baseRequest.setHandled(true);
    }
}
```

Web Service – By Example (1) – Jetty (embedded)

Output for Human



HTML JSON Text

Web Service – By Example (1) – Jetty (embedded) Deployment

- Just like with the JRE HttpServer, “creating” the server, deploying the service on it and then starting the server is handled by Java code.
  - It creates a Jetty Server object,
  - adds objects of the implemented services to specific sub-paths, called contexts in Jetty, of the server and
  - starts the server.

Web Service – By Example (1) – Jetty (embedded) Deployment

```java
package main;

import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.handler.ContextHandler;

import services.PiService;

public class StartServer {

    public static void main(String[] args) {
        try {
            ContextHandler ch = new ContextHandler();
            ch.setContextPath("/pi");
            ch.setHandler(new PiService());

            Server server = new Server(8081);
            server.setHandler(ch);
            server.start();
            server.dumpStdErr();
            server.join();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Web Service – By Example (1) – Jetty (embedded) Deployment

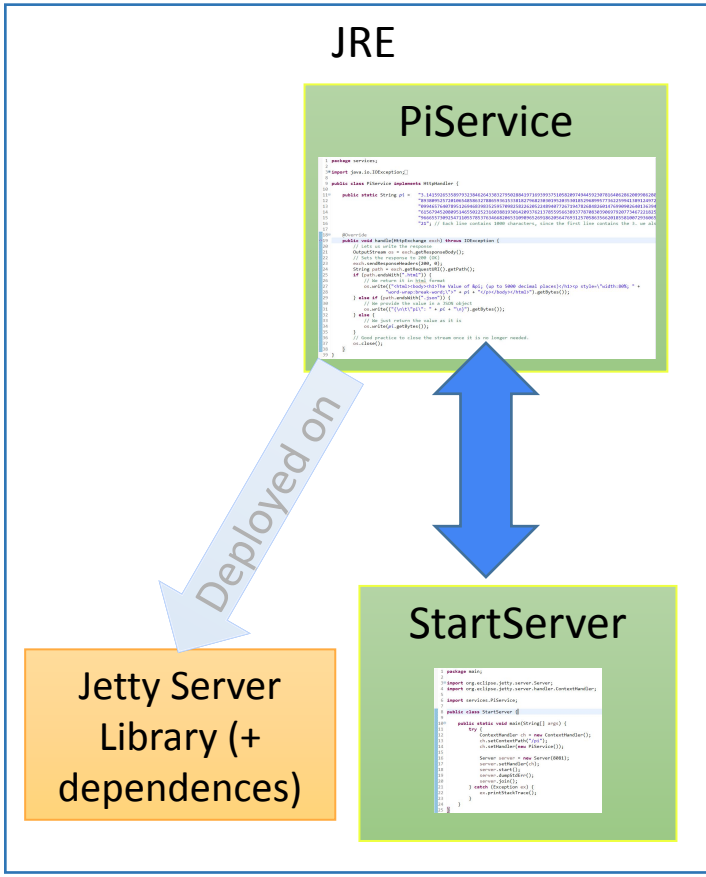
```java
package main;

import org.eclipse.jetty.server.Server;
import org.eclipse.jetty.server.handler.ContextHandler;

import services.PiService;

public class StartServer {

    public static void main(String[] args) {
        try {
            ContextHandler ch = new ContextHandler();
            ch.setContextPath("/pi");
            ch.setHandler(new PiService());

            Server server = new Server(8081);
            server.setHandler(ch);
            server.start();
            server.dumpStdErr();
            server.join();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
```

Create and configure the context

Create, configure and start the server

Web Service – By Example (1) – Jetty (embedded) Deployment

- Deployment of the specific Service
  - The creation of the server and deployment of services is handled through its own class + main method (StartServer).
  - The StartServer uses the Jetty library to provide an endpoint.
  - The PiService and StartServer in the end run in the JRE.
  - Executing the StartServer makes the whole thing work.
  - When the execution stops the server stops too.



Web Service – By Example (1) – Spring (REST)

- Same as the previous service with some differences:
  - Instead of using an embedded Server (like HttpServer in JRE or the one from Jetty) we will use the Spring framework and deploy the service on an Apache Tomcat 8.5.
  - For easier development we use the Spring Tool Suite.
  - Also the following code will ONLY return the value of π as a JSON object (no fancy “for Human” and “Default”).

Web Service – By Example (1) – Spring (REST)

A class for objects representing π

```java
package com.example.main;

public class Pi {

    private static final String pi = "3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862"
            "89380952572010654858632788659361533818279682303019520353018529689957736225994138912497217752834791315155748572"
            "09946576407895126946839835259570982582262052248940772671947826848260147699090264013639443745530506820349625245"
            "61567945208095146550225231603881930142093762137855956638937787083039069792077346722182562599661501421503068038"
            "96665573092547110557853763466820653109896526918620564769312570586356620185581007293606598764861179104533488503"
            "21";

    public Pi() {
        super();
    }

    public String getPi() {
        return Pi.pi;
    }
}
```

A class implementing the service

```java
package com.example.main;

import org.springframework.web.bind.annotation.RequestMapping;

@RestController
public class PiServiceJson {

    @RequestMapping(path="/pi", method=RequestMethod.GET)
    public Pi getPi() {
        return new Pi();
    }
}
```

A class for bootstrapping everything

```java
package com.example.main;

import org.springframework.boot.SpringApplication;

@SpringBootApplication
public class ServiceExamplesApplication extends SpringBootServletInitializer {

    @Override
    protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
        return application.sources(ServiceExamplesApplication.class);
    }

    public static void main(String[] args) {
        SpringApplication.run(ServiceExamplesApplication.class, args);
    }
}
```

Web Service – By Example (1) – Spring (REST)

A class for objects representing π

```
1  package com.example.main;
2
3  public class Pi {
4
5      private static final
6              "893809525720
7              "099465764078
8              "615679452080
9              "966655730925
10             "21";
11
12     public Pi() {
13         super();
14     }
15
16     public String getPi()
17         return Pi.pi;
18     }
19 }
```

```
751
599
909
207
558
```

Unlike previously, communication is handled by Spring and we simply declare where it connects through annotations

Also some fiddling around with the pom.xml (for Maven) was necessary

A class implementing the service

```
1  package com.example.main;
2
3  import org.springframework.web.bind.annotation.RequestMapping;
6
7  @RestController
8  public class PiServiceJson {
9
10     @RequestMapping(path="/pi", method=RequestMethod.GET)
11     public Pi getPi() {
12         return new Pi();
13     }
14 }
```

A class for bootstrapping everything

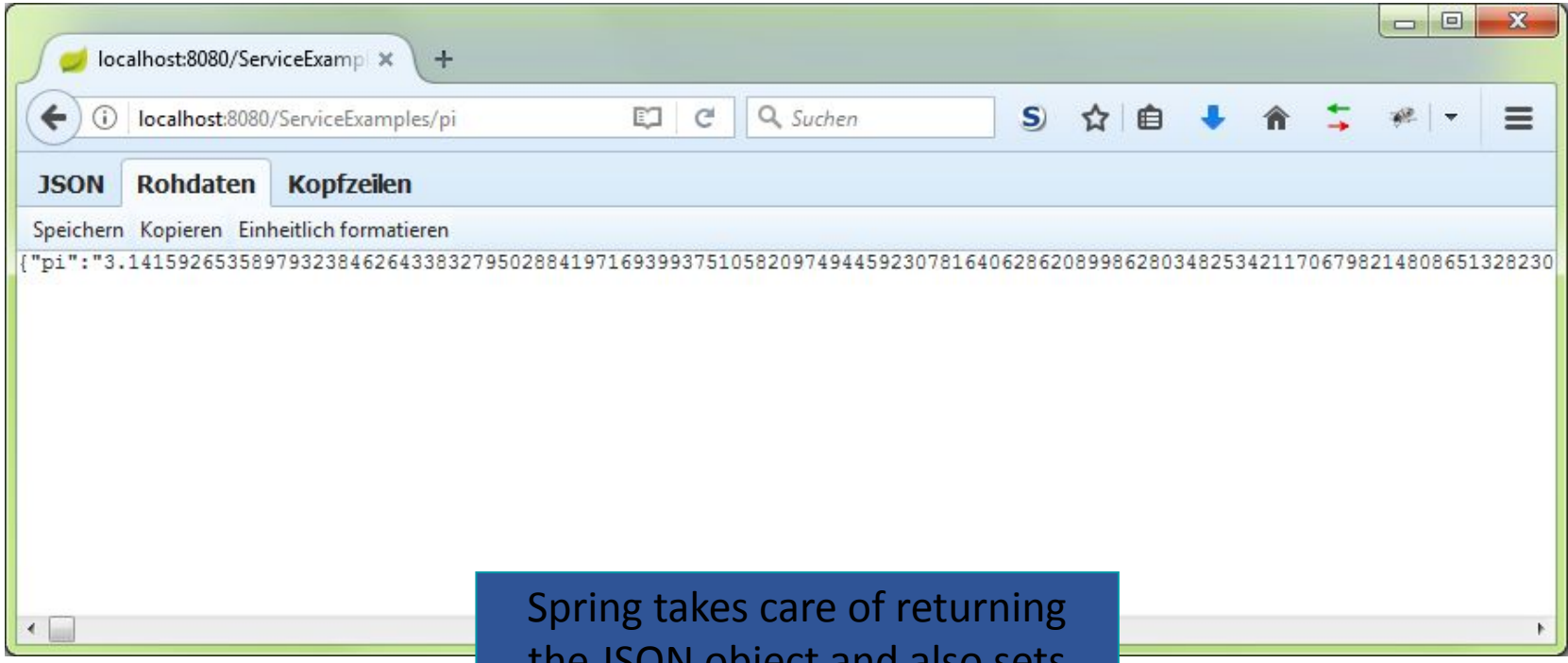
```
1  package com.example.main;
2
3  import org.springframework.boot.SpringApplication;
7
8  @SpringBootApplication
9  public class ServiceExamplesApplication extends SpringBootServletInitializer {
```

```
gApplicationBuilder application) {
ication.class);
```

```
on.class, args);
```

From the programmers point of view the service is very simple: It only returns an object of type Pi

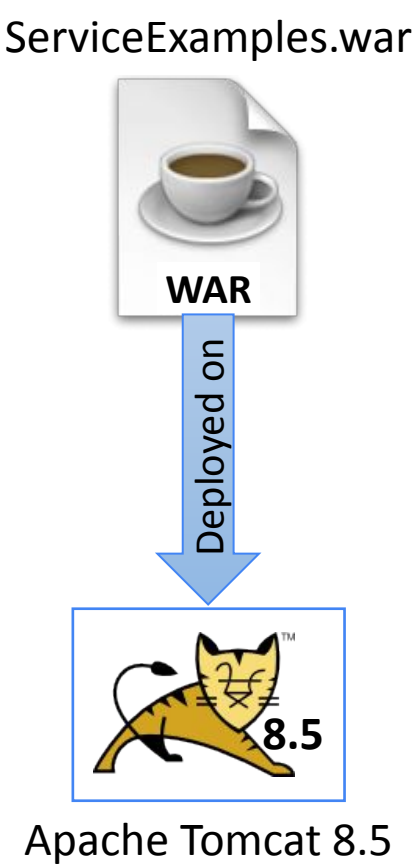
Web Service – By Example (1) – Spring (REST)



Spring takes care of returning the JSON object and also sets the content-type of the response correctly

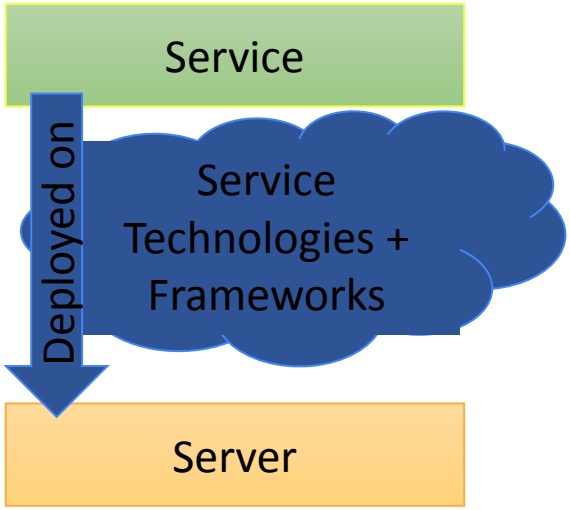
Web Service – By Example (1) – Spring (REST) Deployment

- In this case a .war has been created using Apache Maven.
  - Some configuration of the pom.xml for Maven was necessary.
- This .war has then been deployed on an Apache Tomcat 8.5 (i.e copied into the webapps folder of Tomcat).
  - Note: This has also been deployed on Jetty (not embedded) and tested and worked.
- Starting Tomcat, the service was available at http://localhost:8080/ServiceExamples/pi

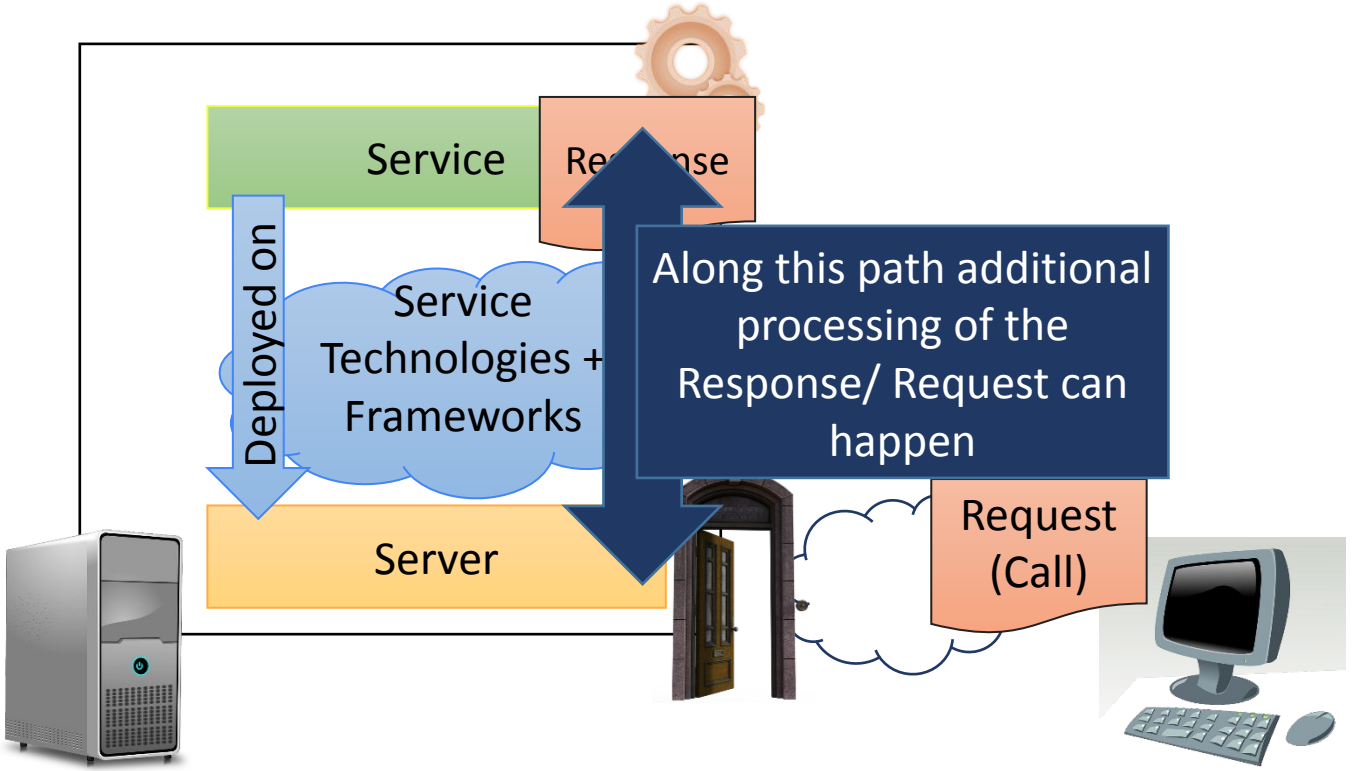


Web Service – Simplest Architecture

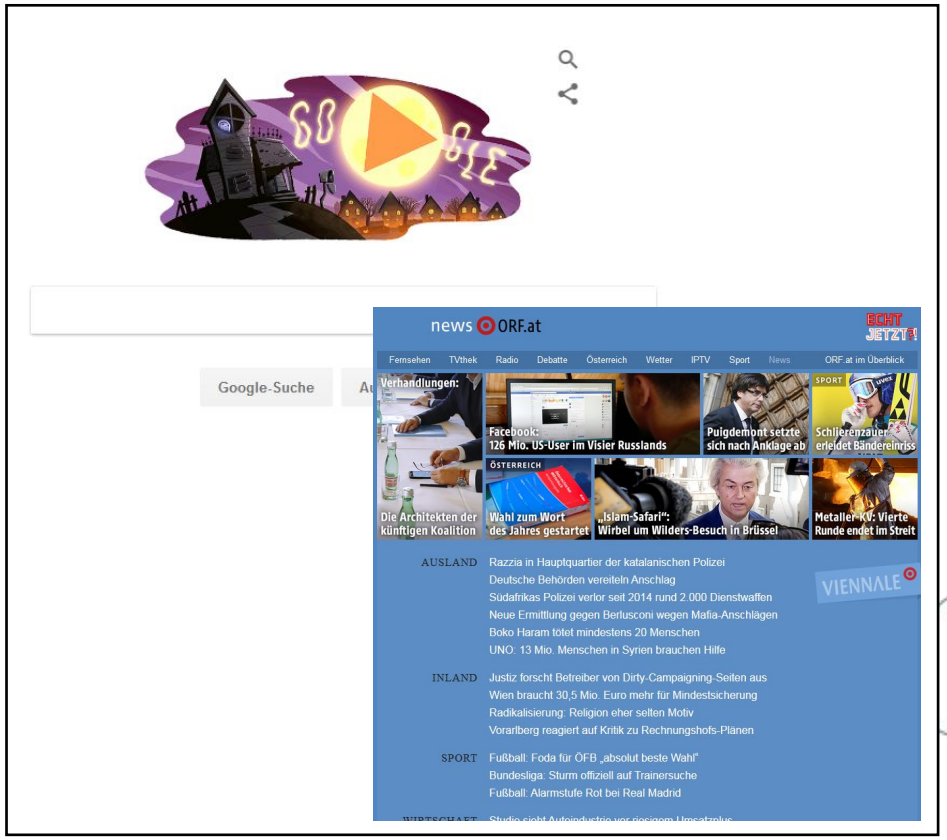
- The deployment is usually handled through a configuration, which specifies the details like the endpoint where the service is accessed.
  - However, the configuration can heavily vary based on the used technologies and frameworks. Sometimes it can be part of the service code (e.g. Spring), sometimes it's in a separate file (e.g. Servlets), or part of the server starting procedure (e.g. JRE HttpServer) etc.
  - For maintainability the deployment details are usually specified relative to the server (e.g. relative path).

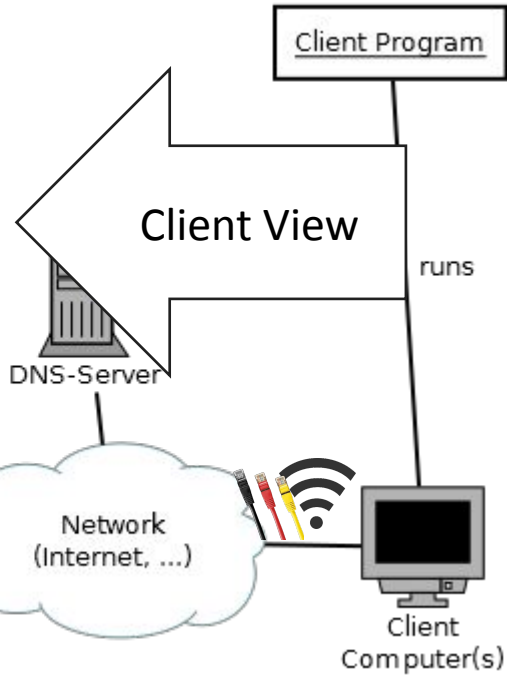


Web Service – Architecture

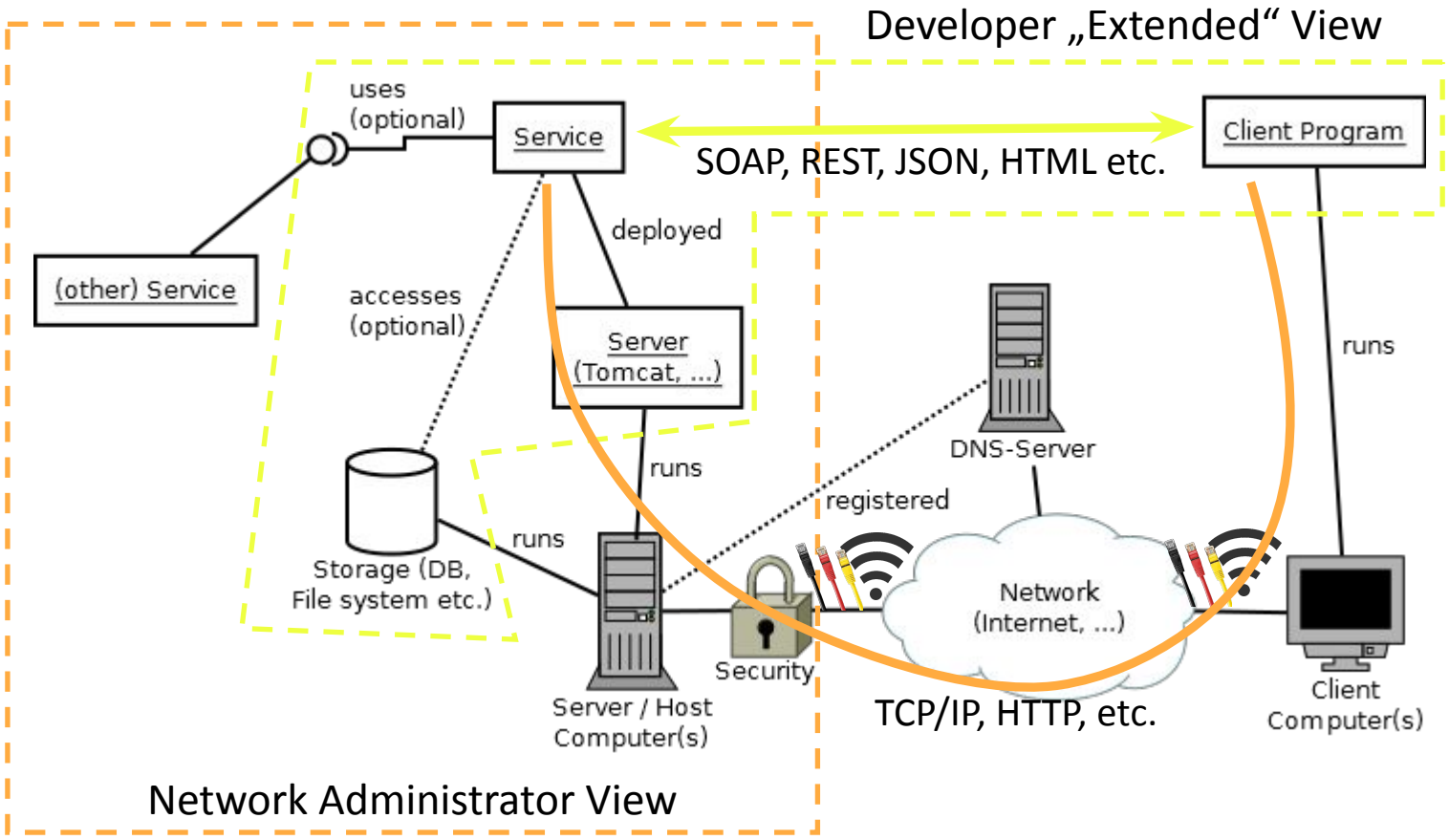


Web Service – General Architecture

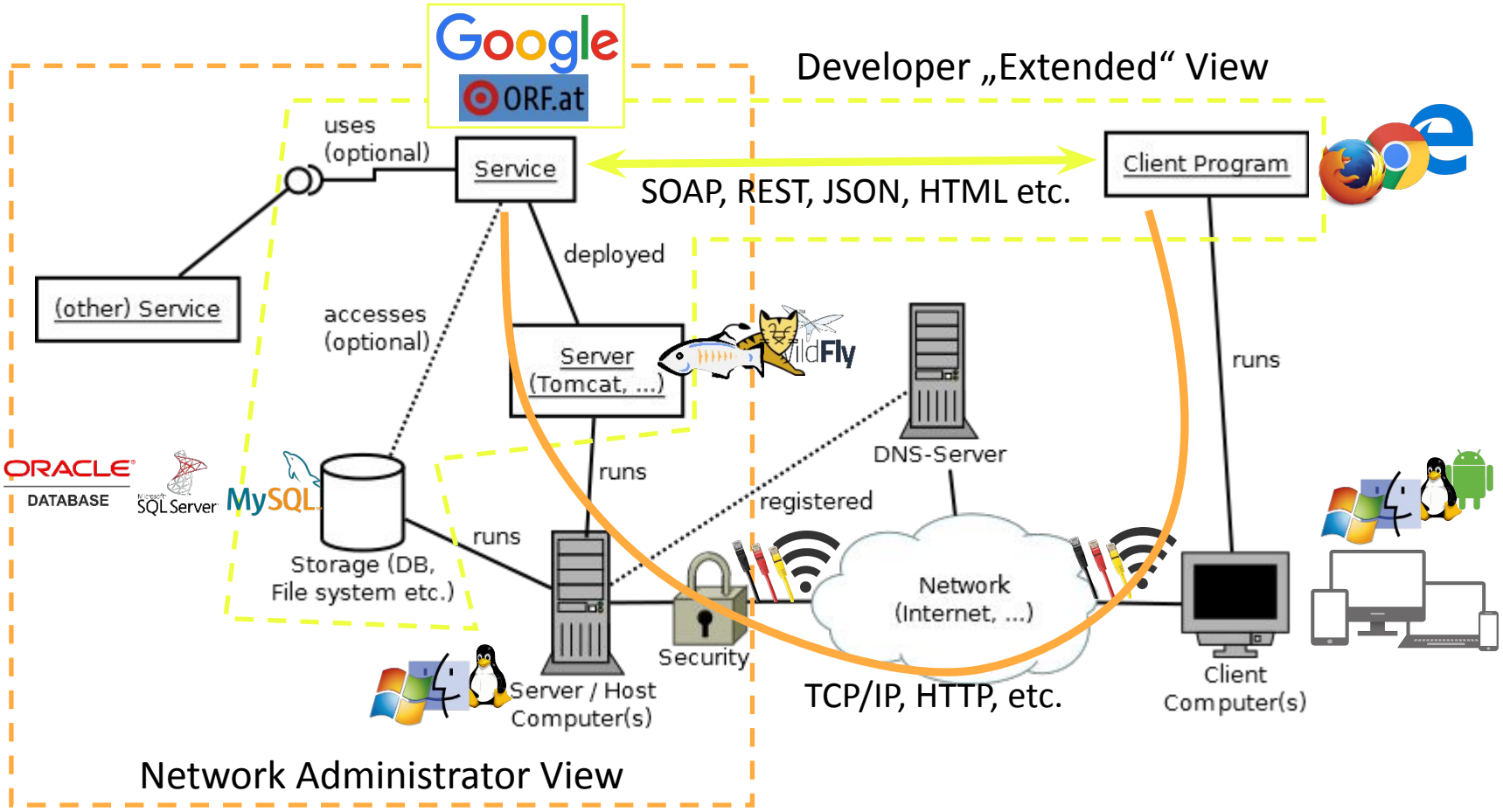




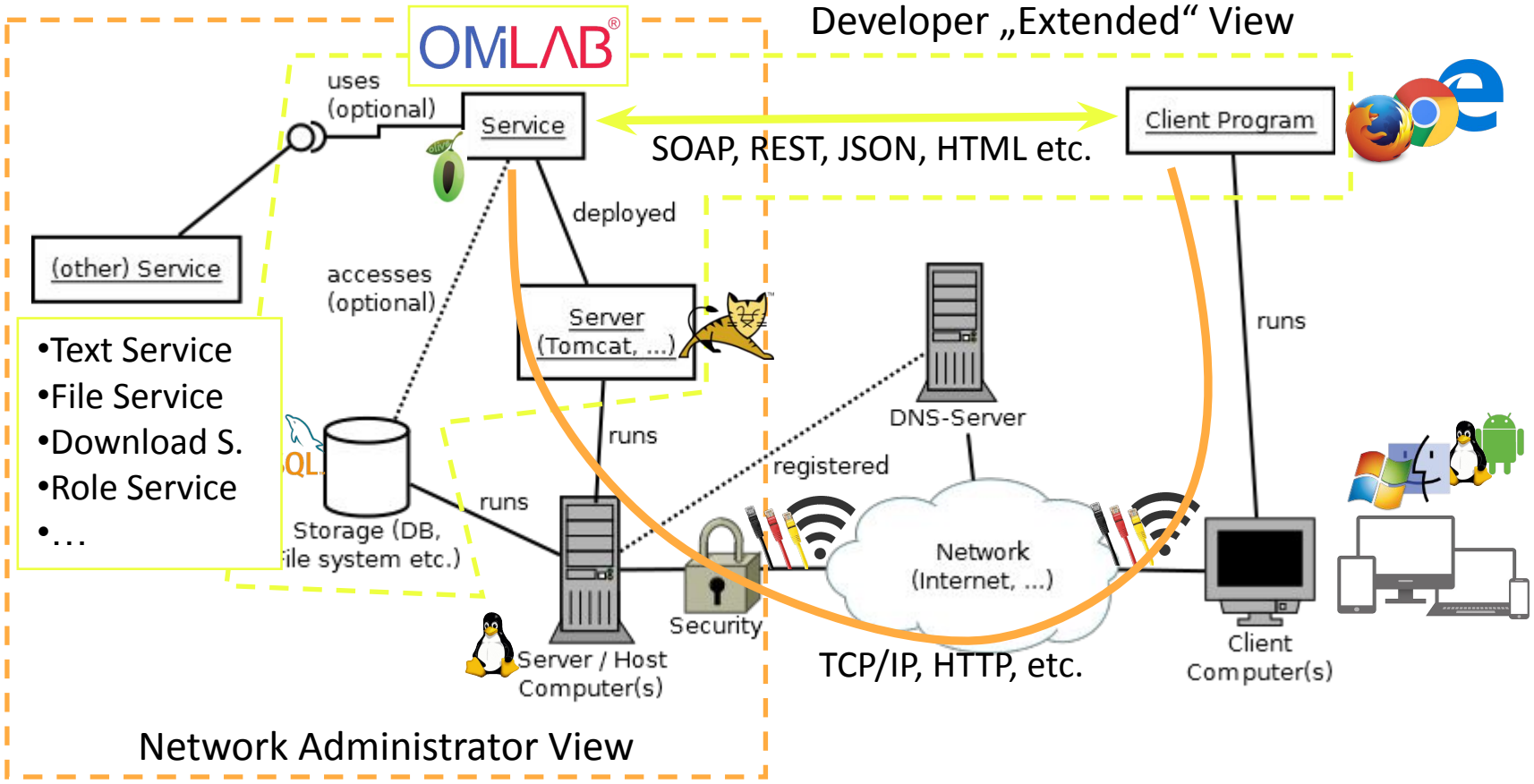
Web Service – General Architecture



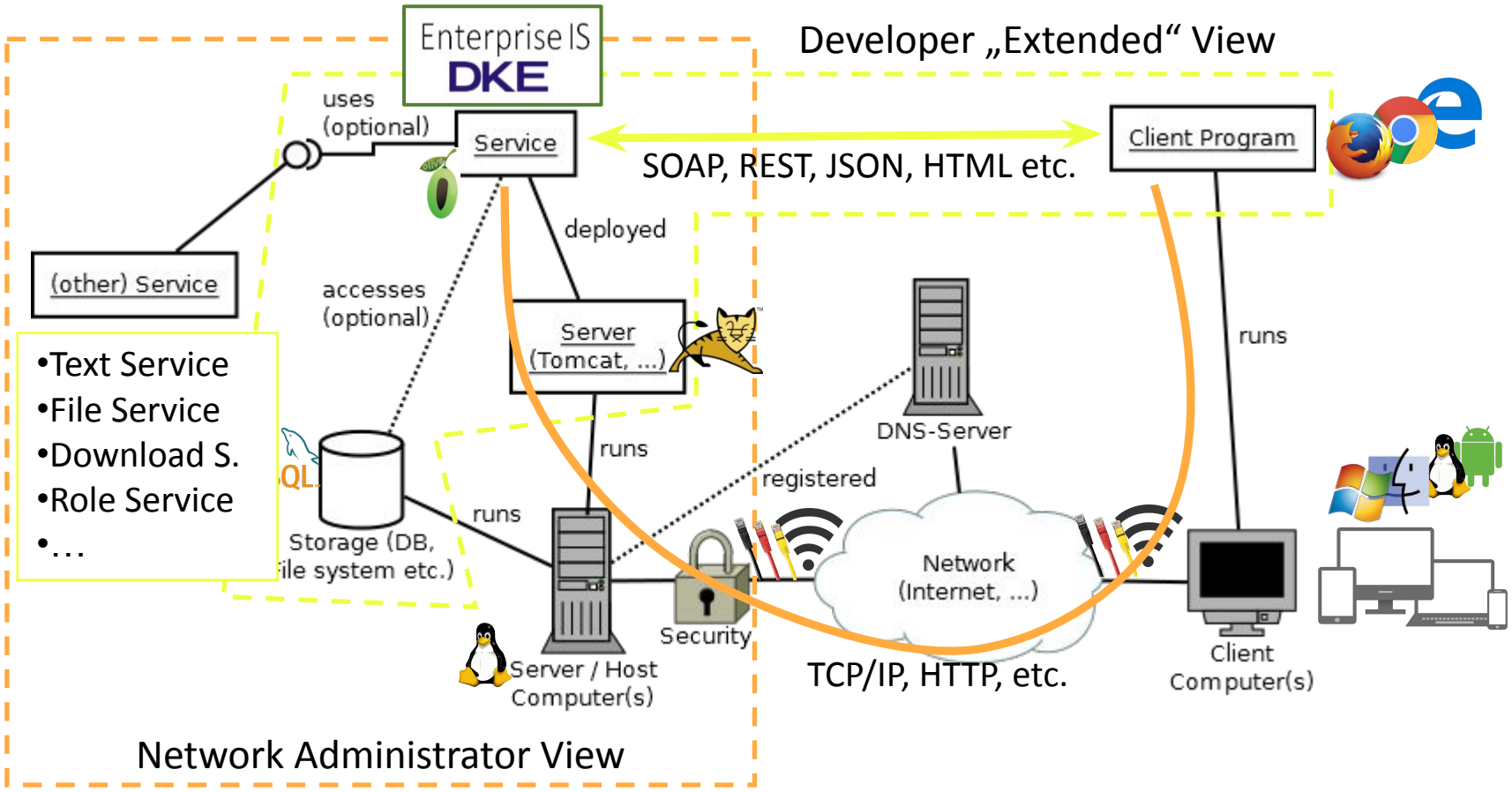
Web Service – General Architecture



Web Service – General Architecture



Web Service – General Architecture

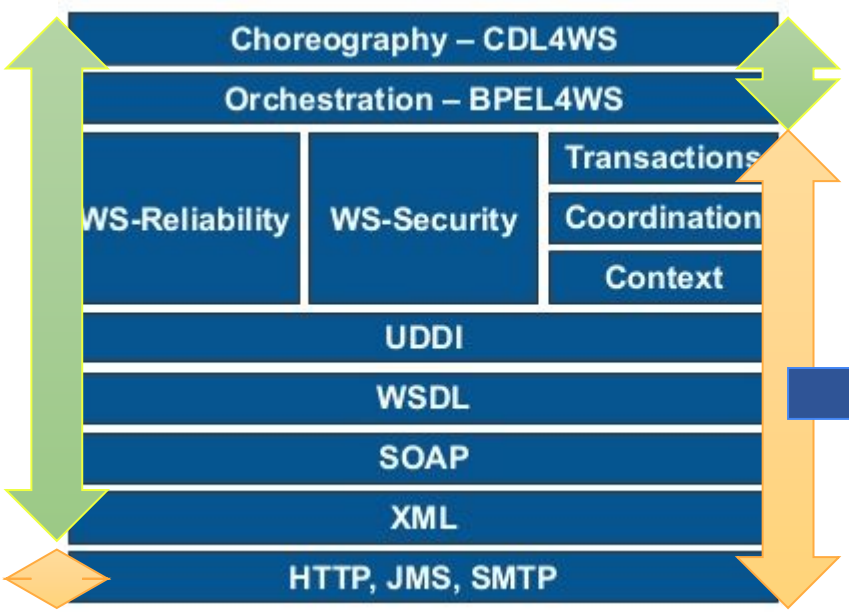


Web Service – Some Technologies

Technologies used for Web-Services

service concern

server + framework concern



For example, one approach for SOAP services with Spring automatically creates and publishes the WSDL

Using JRE built in HTTP Server

Using Spring Framework + Apache Tomcat

+ Configuration

Web Service – Requirements

- There is a cascade of requirements
  - The implemented Service requires some things, like a JRE, a specific server, some software integrated with the server (e.g. PHP) or some libraries (e.g. Java EE, Jackson, Spring etc.)
  - Those in turn require other things themselves, like a Server requires an operating system, a hardware that provides enough power to support the server and any deployed services or libraries depend on other libraries (e.g. require log4j or apache-commons)
- Also the specific case can influence the requirements
  - Is this a generally available service like www.google.com? Then it requires network access to the internet.
  - Or is it a company internal service? Then network access to the company intranet would be enough.
  - Again those can be realized differently, e.g. through a separate network or by configuring the security settings.

Web Service – Requirements

- The Web Service implementation.
- A Server for deploying the Web Service.
- Deployment configuration (at Web Service, Server and other artefacts).
- Security measures (e.g. Firewall, …) configured so the service is accessible.
- A working and sufficient network connection.
- Fitting hardware to run the Server and the Web Service.
  - Depends on the amount of requests that have to be processed (e.g. service used once a day vs. google.com)
  - Depends on difficulty of computational task the service takes over (e.g. calculate area of a rectangle vs. calculate aerodynamics of a car over a certain time)
- Operating system
- Many of the other details depend on:
  - How is the Web Service implemented? (Frameworks?)
  - What does the Web Service do? (e.g. remote medical operation high-speed response)
  - What Server is used for the deployment? (e.g. Tomcat? needs JRE)
  - Who will be using the service? (e.g. do I need a UI?)
  - What is the infrastructure where the Server/Web Service is deployed? (is there a Firewall?)

Web Service – By Example (1) – Requirements – JRE Http Server

- The Service classes implementing HttpHandler.
- A main method that creates an HttpServer object, deploys the services and starts the server.
- A fitting JRE installation (>=1.6).
- And the previously stated general requirements (proper hardware, operating system, configured security/firewall etc.).
  - Operating system? → irrelevant as long as the required JRE is installed
  - Network connection and security? → provided and configured so that the desired clients can access the service
  - Hardware requirements? → depend on the amount of clients that have to be served, the used OS, JRE and Server requirements etc.

Web Service – By Example (1) – Requirements – Jetty (embedded)

- The Service classes implementing AbstractHandler.
- A main method that creates a Jetty Server object, deploys the services and starts the server.
- The necessary Jetty Libraries
  - jetty-server
  - jetty-util
  - servlet-api
  - jetty-http
  - jetty-io
- A fitting JRE installation (>=1.8 for Jetty 9.4).
- And the previously stated general requirements (proper hardware, operating system, configured security/firewall etc.).
  - Operating system? → irrelevant as long as the required JRE is installed
  - Network connection and security? → provided and configured so that the desired clients can access the service
  - Hardware requirements? → depend on the amount of clients that have to be served, the used OS, JRE and Server requirements etc.

Web Service – By Example (1) – Requirements – Spring (REST)

- The code implementing the Service, fitting the Spring Framework.
- Apache Maven – when creating the Web Service, to take care of dependencies and create the deployable .war (alternative: Gradle).
- Apache Tomcat (e.g. Version 8.5) – to deploy the service, in this case as a .war (alternative: Jetty).
- Spring Framework libraries – automatically packed by Maven.
- A fitting JRE installation (e.g. >=1.7) – since the Server (Tomcat) needs it to run.
- (maybe not necessary, but helpful) Spring Tool Suite.
- And the previously stated general requirements (proper hardware, operating system, configured security/firewall etc.).
  - Operating system? → irrelevant as long as the required JRE is installed
  - Network connection and security? → provided and configured so that the desired clients can access the service
  - Hardware requirements? → depend on the amount of clients that have to be served, the used OS, JRE and Server (Tomcat about 1-2 GB of RAM?) requirements etc.
- (Note: There are probably many other ways to deploy a Spring Web Service. These are just the requirements for one of them.)

Web Service – Noticed Shifts in Java Service implementation

- Back in the days, implementing services in Java generally worked like this:
  - Implement the service.
  - Write a configuration file, with deployment details etc., for the desired framework/server.
  - Deploy the service on the server.
- Now there seem to be additional alternatives that might gain some traction:
  1. The deployment details can be handled inside the implemented service code, e.g. through annotations.
  2. Applications are not necessarily deployed on servers. Instead the server is part of the written code including the service (cf. the JRE HttpServer and the embedded Jetty examples).

Web Service – Noticed Shifts in Java Service implementation

- For change/alternative 1:
  - Previously deploying a Java Servlet required a web.xml, that specified which classes implemented servlets and under which relative URIs they should be accessible.
  - Now Java Servlets can instead use Java Annotations, like @WebServlet, to specify deployment details like names, servlet-mappings etc.
  - Spring is using such annotations as well to configure the written services (e.g. @RestController, @RequestMapping).
  - Possible downside: web.xml could be adapted when necessary without having to change the code (it was readable in the .war file). How do you change the Java Annotation of a compiled class? Unless the server provides some specific order for loading configurations (e.g. first web.xml if available, then Java Annotations) that might be difficult. Even then, what if only certain parts should be changed (cf. Jetty)?

Web Service – Example for Shift 1

- Previously:

```java
public class Teapot extends HttpServlet {
    private static final long serialVersionUID = 1L;

    private String[] lines;
    private int nextline;


    public void init() {
        this.lines = new String[] {
                "I'm a little teapot short and stout",
                "Here is my handle, here is my spout",
                "When I get all steamed up I just shout",
                "Tip me over and pour me out",
                "I am a very special pot, it is true",
                "Here is an example of what I can do",
                "I can turn my handle into a spout",
                "Tip me over and pour me out"
        };
        this.nextline = 0;
    }


    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        PrintWriter pw = response.getWriter();

        pw.println("<!DOCTYPE html>\n<html>\n\t<head>\n\t\t
        pw.println(this.lines[this.nextline]);
        this.nextline = (nextline+1)%this.lines.length;
        pw.println("</p>\n\t</body>\n</html>");
    }
}
```

Java implementation
 The compiled version is put in the package (.war).

web.xml also in the package (.war)

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

    <display-name>Teapot Application</display-name>
    <description>Sometimes all you want to be is a teapot.</description>

    <servlet>
        <servlet-name>Teapot</servlet-name>
        <servlet-class>teapot.Teapot</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Teapot</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

Web Service – Example for Shift 1

- Now possible:

Java implementation

```java
@WebServlet(name="Teapot", urlPatterns="/")
public class Teapot extends HttpServlet {
    private static final long serialVersionUID

    private String[] lines;
    private int nextline;


    public void init() {
        this.lines = new String[] {
                "I'm a little teapot short and stout",
                "Here is my handle, here is my spout",
                "When I get all steamed up I just shout",
                "Tip me over and pour me out",
                "I am a very special pot, it is true",
                "Here is an example of what I can do",
                "I can turn my handle into a spout",
                "Tip me over and pour me out"
        };
        this.nextline = 0;
    }


    public void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        response.setContentType("text/html");

        PrintWriter pw = response.getWriter();

        pw.println("<!DOCTYPE html>\n<html>\n\t<head>\n\t\t<title>Teapot</title>\n\t</head>\n\t<body>\n\t\t<p>");
        pw.println(this.lines[this.nextline]);
        this.nextline = (nextline+1)%this.lines.length;
        pw.println("</p>\n\t</body>\n</html>");
    }
}
```

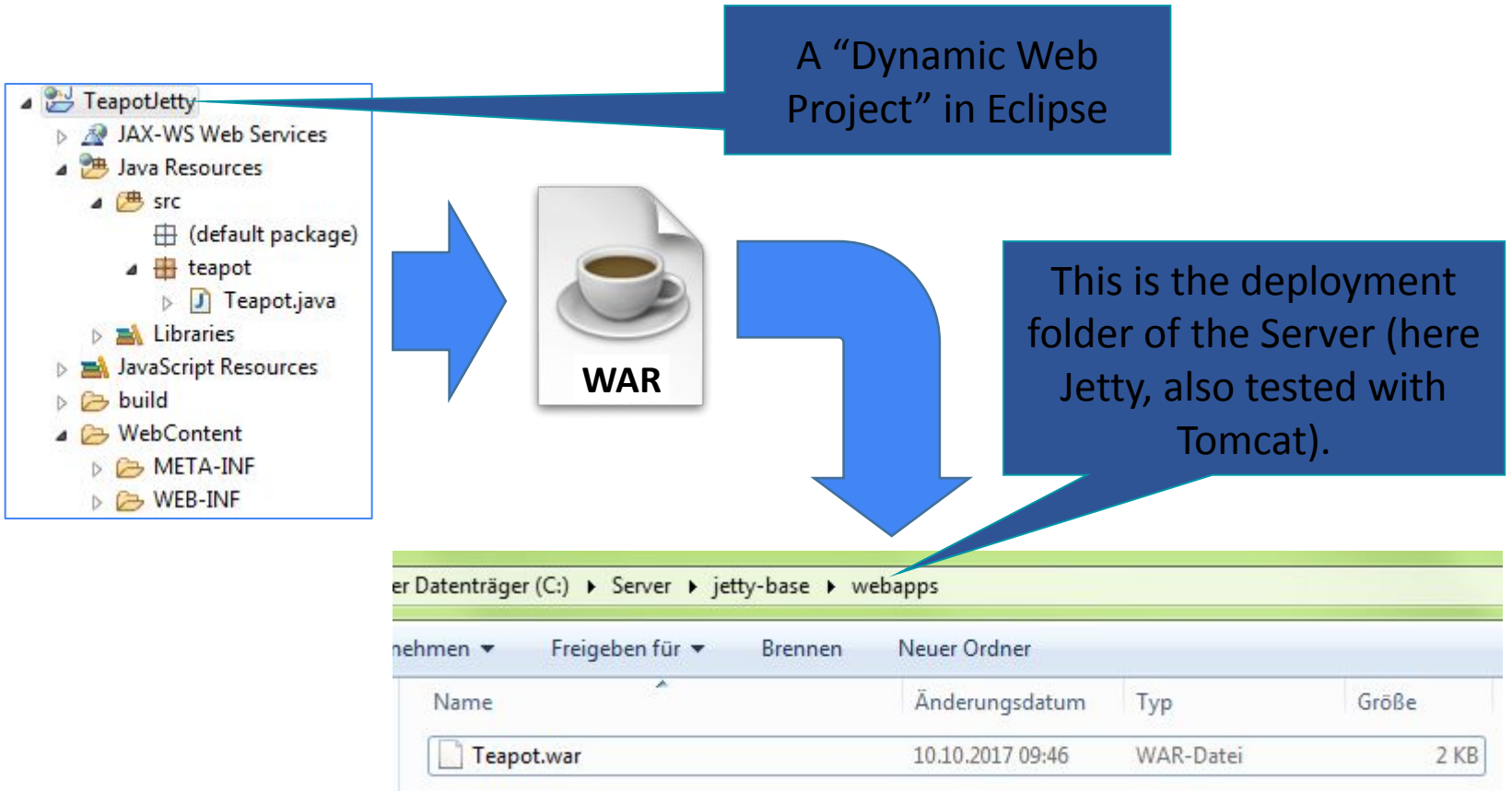
Instead of a web.xml we just have this small annotation. This works with recent Tomcat and Jetty versions.
The compiled version is again put in the package (.war), this time without a web.xml.

Web Service – Noticed Shifts in Java Service implementation

- For change/alternative 2:
  - Previously a service was created, packed and deployed on a server, like Tomcat. The service alone wouldn't run without being deployed on a server.
  - Now there are libraries that provide "embedded servers", where creating an executable .jar package from a service can be run directly and contains the necessary server.
  - For example Jetty's slogan "Don't deploy your application in Jetty, deploy Jetty in your application!" or Spring Boot which "Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)"
  - Possible downside: Server configuration is then responsibility of the Service developer. How do you change ports the server listens to later on?

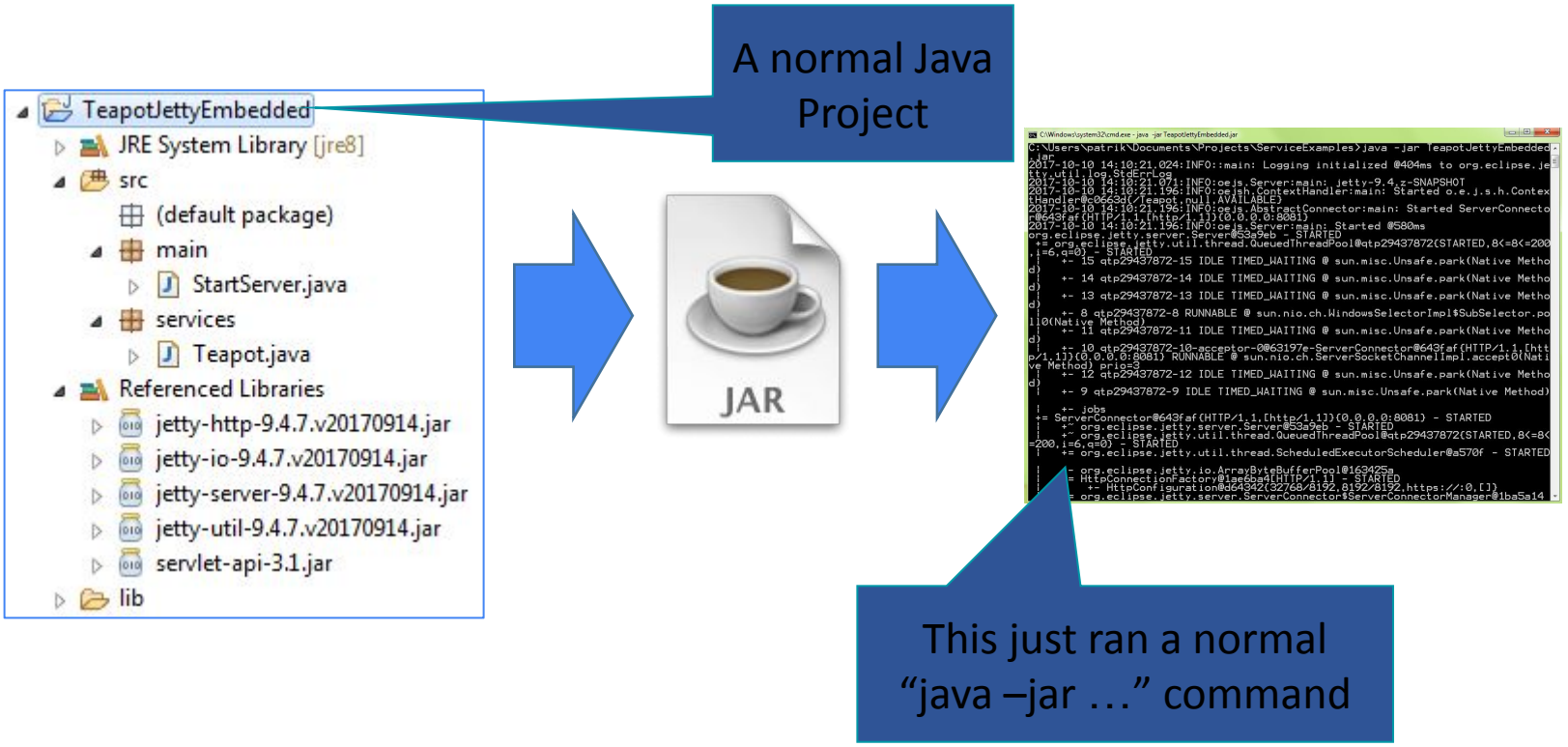
Web Service – Example for Shift 2

- Previously:



Web Service – Example for Shift 2

- Now possible:



Note: In this case the service will ALWAYS run on a Jetty server.

Web Service – By Example (2)

- Another simple service, which returns the current time.
- The time is based on the location and configuration of the server.
- The Service is implemented in Java, aiming to be deployed on the HTTP Server provided by the JRE (Java Runtime Environment).

Web Service – By Example (2)

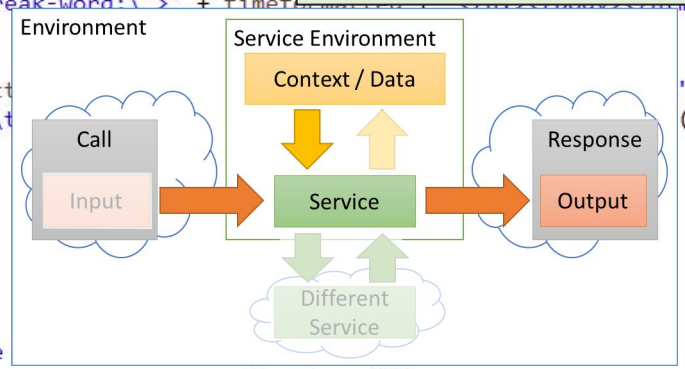
```java
package services;

import java.io.IOException;

public class CurrentTimeService implements HttpHandler {

    @Override
    public void handle(HttpExchange exch) throws IOException {
        // Lets us write the response
        OutputStream os = exch.getResponseBody();
        if ("GET".equals(exch.getRequestMethod())) {
            // Sets the response to 200 (OK)
            exch.sendResponseHeaders(200, 0);
            // Get the current date and time
            LocalDateTime dt = LocalDateTime.now();
            String timeformatted = (String.format("%02d", dt.getHour()) + ":" + String.format("%02d", dt.getMinute()) +
                    ":" + String.format("%02d", dt.getSecond()));
            String path = exch.getRequestURI().getPath();
            if (path.endsWith(".html")) {
                // We return it in html format
                os.write(("<html><body><h1 style=\"word-wrap:break-word;\">" + timeformatted + "</h1></body></html>").getBytes());
            } else if (path.endsWith(".json")) {
                // We provide the value in a JSON object
                os.write(("{\n\t\"formatted\": \"" + timeformatted + "\",\n\t\"hour\": " + dt.getHour() + ",\n\t" +
                        "\"minute\": " + dt.getMinute() + ",\n\t\"second\": " + dt.getSecond() + "\n}").getBytes());
            } else {
                // We write the time in HH:MM:SS format
                os.write(timeformatted.getBytes());
            }
        } else {
            // Sets the response to 405 (Method Not Allowed)
            exch.sendResponseHeaders(405, 0);
            os.write("405 - Method Not Allowed, please only use the GET method.".getBytes());
        }
        // Good practice to close the stream once it is no longer needed.
        os.close();
    }
}
```

Web Service – By Example (2)

```java
package services;

import java.io.IOException;

public class CurrentTimeService implements HttpHandler {

    @Override
    public void handle(HttpExchange exch) throws IOException {
        // Lets us write the response
        OutputStream os = exch.getResponseBody();
        if ("GET".equals(exch.getRequestMethod())) {
            // Sets the response to 200 (OK)
            exch.sendResponseHeaders(200, 0);
            // Get the current date and time
            LocalDateTime dt = LocalDateTime.now();
            String timeformatted = (String.format("%02d", dt.getHour()) + ":" + Stri
                    ":" + String.format("%02d", dt.getSecond()));
            String path = exch.getRequestURI().getPath();
            if (path.endsWith(".html")) {
                // We return it in html format
                os.write(("<html><body><h1 style=\"word-wrap:break-word;\">" + timef
            } else if (path.endsWith(".json")) {
                // We provide the value in a JSON object
                os.write(("{\n\t\"formatted\": \"" + timeformatt
                        "\"minute\": " + dt.getMinute() + ",\n\t
            } else {
                // We write the time in HH:MM:SS format
                os.write(timeformatted.getBytes());
            }
        } else {
            // Sets the response to 405 (Method Not Allowed)
            exch.sendResponseHeaders(405, 0);
            os.write("405 - Method Not Allowed, please only use
        }
        // Good practice to close the stream once it is no longer needed.
        os.close();
    }
}
```

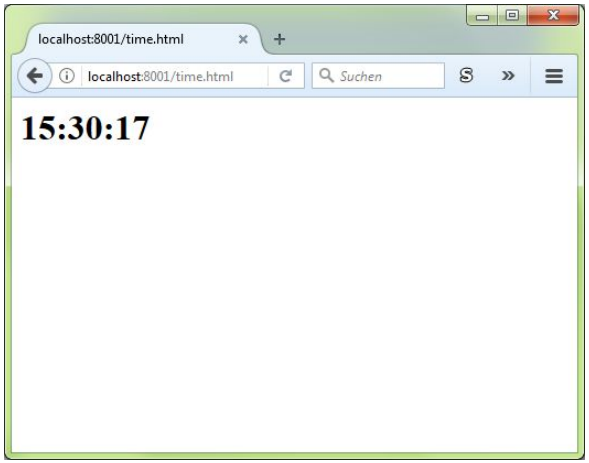
Uses the local time of the service environment

The remainder deals with communication, the output more specifically



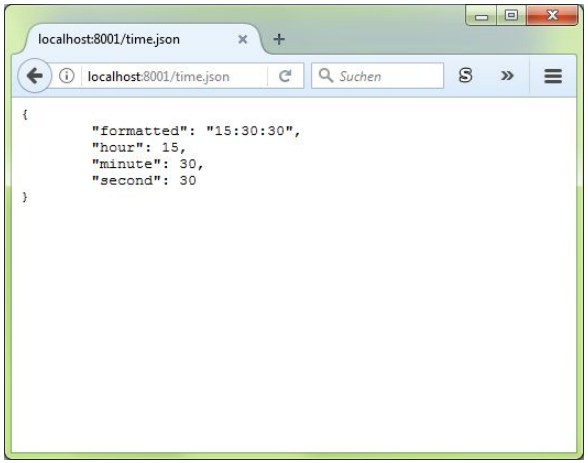
Web Service – By Example (2)

Output for Human



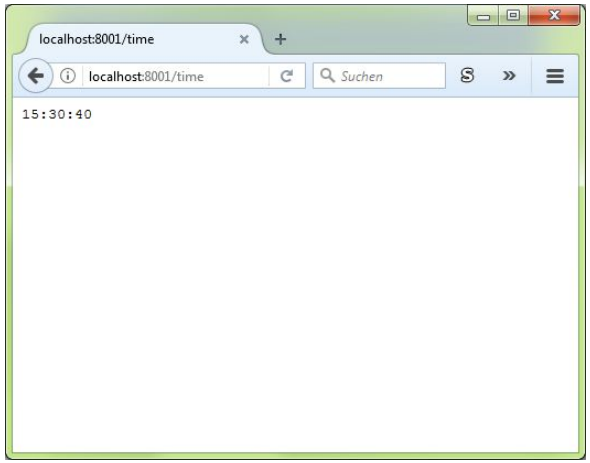
HTML

Output for Machine



JSON

Default Output



Text

Web Service – By Example (3)

- Yet again a simple service, this time it returns a specific Fibonacci number.
- As such it takes one input and based on that determines the Fibonacci number to provide as the response.
  - The input specifies which Fibonacci number should be returned (the 1st? The 2nd? The 3rd? …).
- The Service is implemented in Java, aiming to be deployed on the HTTP Server provided by the JRE (Java Runtime Environment).

Web Service – By Example (3)

```java
package services;

import java.io.IOException;

public class FibonacciNumberService implements HttpHandler {

    @Override
    public void handle(HttpExchange exch) throws IOException {
        // Lets us write the response
        OutputStream os = exch.getResponseBody();
        if ("GET".equals(exch.getRequestMethod())) {
            // Get the query parameters, note that we force to start and end it with & so we have proper delimiters between the key-value pairs
            String query = "&" + exch.getRequestURI().getQuery() + "&";
            int startpos = query.indexOf("&n=");
            if (startpos < 0) { // If the n parameter is missing
                // Sets the response to 400 (Bad Request)
                exch.sendResponseHeaders(400, 0);
                os.write("400 - Bad Request, parameter n is missing.".getBytes());
                os.close();
                return;
            }
            int number = Integer.parseInt(query.substring(startpos+3, query.indexOf("&", startpos+1)));
            if (number <= 0) { // If they are asking for a negative Fibonacci number ... which doesn't exist
                // Sets the response to 400 (Bad Request)
                exch.sendResponseHeaders(400, 0);
                os.write(("400 - Bad Request, the first Fibonacci number is 1, not " + number).getBytes());
                os.close();
                return;
            }
            // Determine the Fibonacci number
            // Note that fib1 and fib2 should be interpreted as "n" and "n-1"
            BigInteger fib = new BigInteger("1"), fib1 = new BigInteger("1"), fib2 = new BigInteger("1");
            for(int i=3; i<=number; i++) { // this is only executed if they ask for a Fibbonaci number >= 3
                fib = fib1.add(fib2);
                fib1 = fib2;
                fib2 = fib;
            }
            // Sets the response to 200 (OK)
            exch.sendResponseHeaders(200, 0);
            String path = exch.getRequestURI().getPath();
            if (path.endsWith(".html")) {
                // We return it in html format
                os.write(("<html><body><p>Fibonacci <b>#" + number + "</b> is <b style=\"word-wrap:break-word;\">" + fib.toString() + "</b></p></body></html>").getBytes());
            } else if (path.endsWith(".json")) {
                // We provide the value in a JSON object
                os.write(("{\n\t\"number\": " + number + ",\n\t\"value\": " + fib.toString() + "\n}").getBytes());
            } else {
                // Write the determined Fibonacci number
                os.write(fib.toString().getBytes());
            }
        } else {
            // Sets the response to 405 (Method Not Allowed)
            exch.sendResponseHeaders(405, 0);
            os.write("405 - Method Not Allowed, please only use the GET method.".getBytes());
        }
        // Good practice to close the stream once it is no longer needed.
        os.close();
    }
}
```

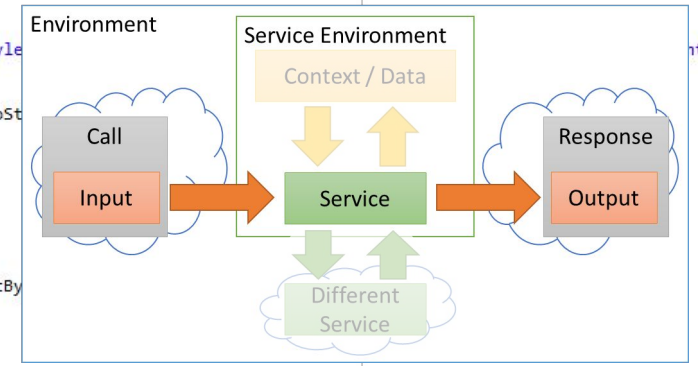
Web Service – By Example (3)

```java
package services;

import java.io.IOException;

public class FibonacciNumberService implements HttpHandler {

    @Override
    public void handle(HttpExchange exch) throws IOException {
        // Lets us write the response
        OutputStream os = exch.getResponseBody();
        if ("GET".equals(exch.getRequestMethod())) {
            // Get the query parameters, note that we force to start and end it with & so we have proper delimiters between the key-value pairs
            String query = "&" + exch.getRequestURI().getQuery() + "&";
            int startpos = query.indexOf("&n=");
            if (startpos < 0) { // If the n parameter is missing
                // Sets the response to 400 (Bad Request)
                exch.sendResponseHeaders(400, 0);
                os.write("400 - Bad Request, parameter n is missing.".getBytes());
                os.close();
                return;
            }
            int number = Integer.parseInt(query.substring(startpos+3, query.indexOf("&", startpos+1)));
            if (number <= 0) { // If they are asking for a negative Fibonacci number ... which doesn't exist
                // Sets the response to 400 (Bad Request)
                exch.sendResponseHeaders(400, 0);
                os.write(("400 - Bad Request, the first Fibonacci number is 1, not " + number).getBytes());
                os.close();
                return;
            }
            // Determine the Fibonacci number
            // Note that fib1 and fib2 should be interpreted as "n" and "n-1"
            BigInteger fib = new BigInteger("1"), fib1 = new BigInteger("1"), fib2 = new BigInteger("1");
            for(int i=3; i<=number; i++) { // this is only executed if they ask for a Fibonacci number >= 3
                fib = fib1.add(fib2);
                fib1 = fib2;
                fib2 = fib;
            }
            // Sets the response to 200 (OK)
            exch.sendResponseHeaders(200, 0);
            String path = exch.getRequestURI().getPath();
            if (path.endsWith(".html")) {
                // We return i
                os.write(("<ht                                                    o style                                                    ntml>").getBytes());
            } else if (path.en
                // We provide
                os.write(("{\n                                                    ib.toSt
            } else {
                // Write the d
                os.write(fib.t
            }
        } else {
            // Sets the respon
            exch.sendResponseHeaders(405, 0);
            os.write("405 - Method Not Allowed, please only use the GET method.".getBy
        }
        // Good practice to close the stream once it is no longer needed.
        os.close();
    }
}
```

Here the input is determined that has been passed with the request

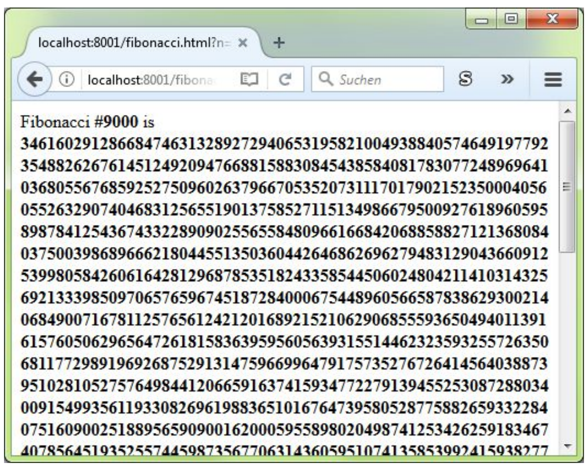
Here the value is calculated based on the input

The remainder deals with handling the rest of the communication

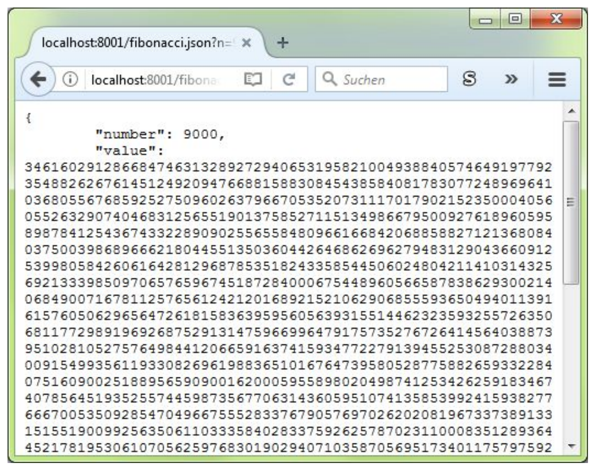


Web Service – By Example (3)

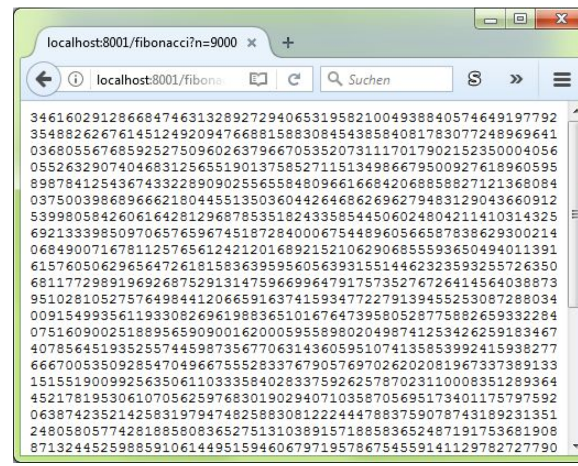
Output for Human	Output for Machine	Default Output



HTML



JSON



Text

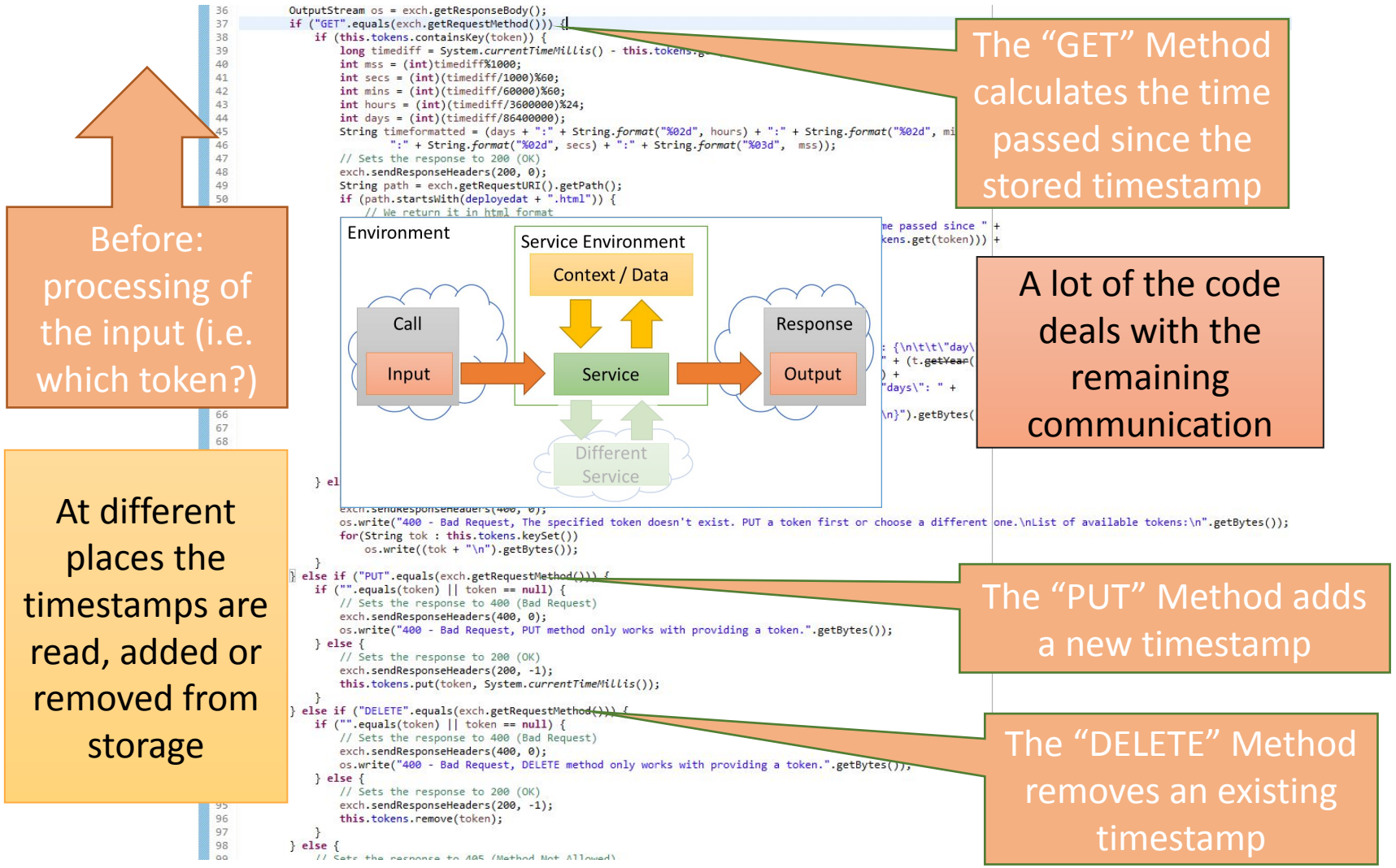
Web Service – By Example (4)

- This time a more sophisticated stopwatch service is provided.
- It specifies how much time has passed since a specific moment.
- Without any additional parameters it will return the duration since the service has been started.
- Additionally different methods can be used to specify additional timers by adding more characters to the called URL (referred to as “tokens”):
  - PUT token → Adds a new timer or resets an existing timer for the token.
  - GET token → Returns the duration for the token timer.
  - DELETE token → Removes the existing token timer if it exists.
- The Service is implemented in Java, aiming to be deployed on the HTTP Server provided by the JRE (Java Runtime Environment).

Web Service – By Example (4)

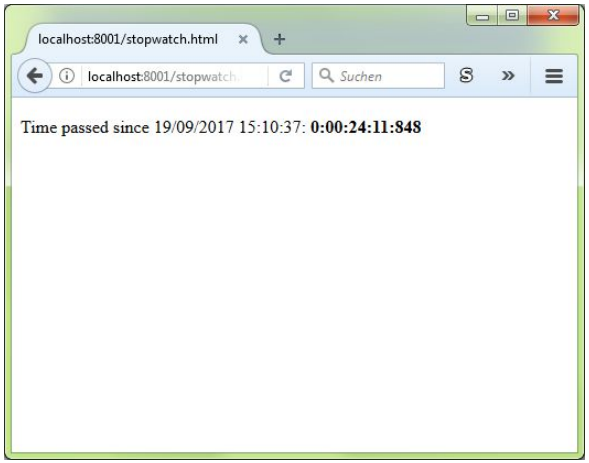
```
36          OutputStream os = exch.getResponseBody();
37          if ("GET".equals(exch.getRequestMethod())) {
38              if (this.tokens.containsKey(token)) {
39                  long timediff = System.currentTimeMillis() - this.tokens.get(token);
40                  int mss = (int)timediff%1000;
41                  int secs = (int)(timediff/1000)%60;
42                  int mins = (int)(timediff/60000)%60;
43                  int hours = (int)(timediff/3600000)%24;
44                  int days = (int)(timediff/86400000);
45                  String timeformatted = (days + ":" + String.format("%02d", hours) + ":" + String.format("%02d", mins) +
46                          ":" + String.format("%02d", secs) + ":" + String.format("%03d",  mss));
47                  // Sets the response to 200 (OK)
48                  exch.sendResponseHeaders(200, 0);
49                  String path = exch.getRequestURI().getPath();
50                  if (path.startsWith(deployedat + ".html")) {
51                      // We return it in html format
52                      os.write(("<html><body><h1>" + token + "</h1><p style=\"word-wrap:break-word;\">Time passed since " +
53                              (new SimpleDateFormat("dd/MM/yyyy HH:mm:ss")).format(new Timestamp(this.tokens.get(token))) +
54                              ": <b>" + timeformatted + "</b></p></body></html>").getBytes());
55                  } else if (path.startsWith(deployedat + ".json")) {
56                      Timestamp t = new Timestamp(this.tokens.get(token));
57                      // We provide the value in a JSON object
58                      String tokenjson = "";
59                      if (!"".equals(token))
60                          tokenjson = "\",\n\t\"token\": \"" + token;
61                      os.write(("{\n\t\"formatted\": \"" + timeformatted + tokenjson + "\",\n\t\"start\": {\n\t\t\"day\": " +
62                              t.getDate() + ",\n\t\t\"month\": " + (t.getMonth()+1) + ",\n\t\t\"year\": " + (t.getYear()+1900) +
63                              ",\n\t\t\"hour\": " + t.getHours() + ",\n\t\t\"minute\": " + t.getMinutes() +
64                              ",\n\t\t\"second\": " + t.getSeconds() + "\n\t},\n\t\"duration\": {\n\t\t\"days\": " +
65                              days + ",\n\t\t\"hours\": " + hours + ",\n\t\t\"minutes\": " + mins +
66                              ",\n\t\t\"seconds\": " + secs + ",\n\t\t\"milliseconds\": " + mss + "\n\t}\n}").getBytes());
67                  } else {
68                      // Send the time passed
69                      os.write(timeformatted.getBytes());
70                  }
71              } else {
72                  // Sets the response to 400 (Bad Request)
73                  exch.sendResponseHeaders(400, 0);
74                  os.write("400 - Bad Request, The specified token doesn't exist. PUT a token first or choose a different one.\nList of available tokens:\n".getBytes());
75                  for(String tok : this.tokens.keySet())
76                      os.write((tok + "\n").getBytes());
77              }
78          } else if ("PUT".equals(exch.getRequestMethod())) {
79              if ("".equals(token) || token == null) {
80                  // Sets the response to 400 (Bad Request)
81                  exch.sendResponseHeaders(400, 0);
82                  os.write("400 - Bad Request, PUT method only works with providing a token.".getBytes());
83              } else {
84                  // Sets the response to 200 (OK)
85                  exch.sendResponseHeaders(200, -1);
86                  this.tokens.put(token, System.currentTimeMillis());
87              }
88          } else if ("DELETE".equals(exch.getRequestMethod())) {
89              if ("".equals(token) || token == null) {
90                  // Sets the response to 400 (Bad Request)
91                  exch.sendResponseHeaders(400, 0);
92                  os.write("400 - Bad Request, DELETE method only works with providing a token.".getBytes());
93              } else {
94                  // Sets the response to 200 (OK)
95                  exch.sendResponseHeaders(200, -1);
96                  this.tokens.remove(token);
97              }
98          } else {
99              // Sets the response to 405 (Method Not Allowed)
```

Web Service – By Example (4)



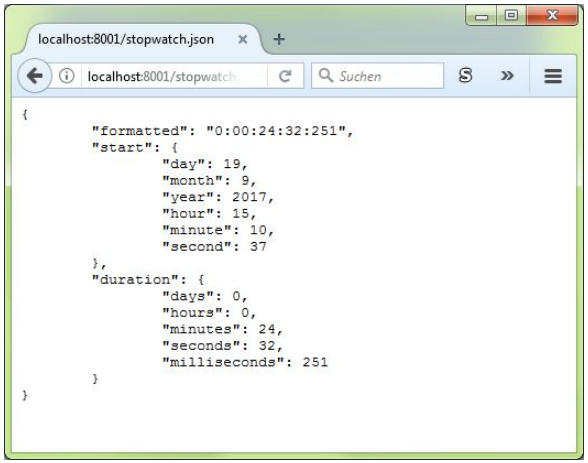
Web Service – By Example (4)

Output for Human



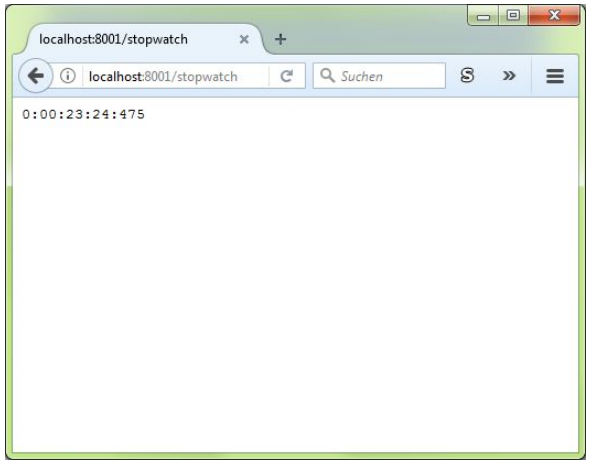
HTML

Output for Machine



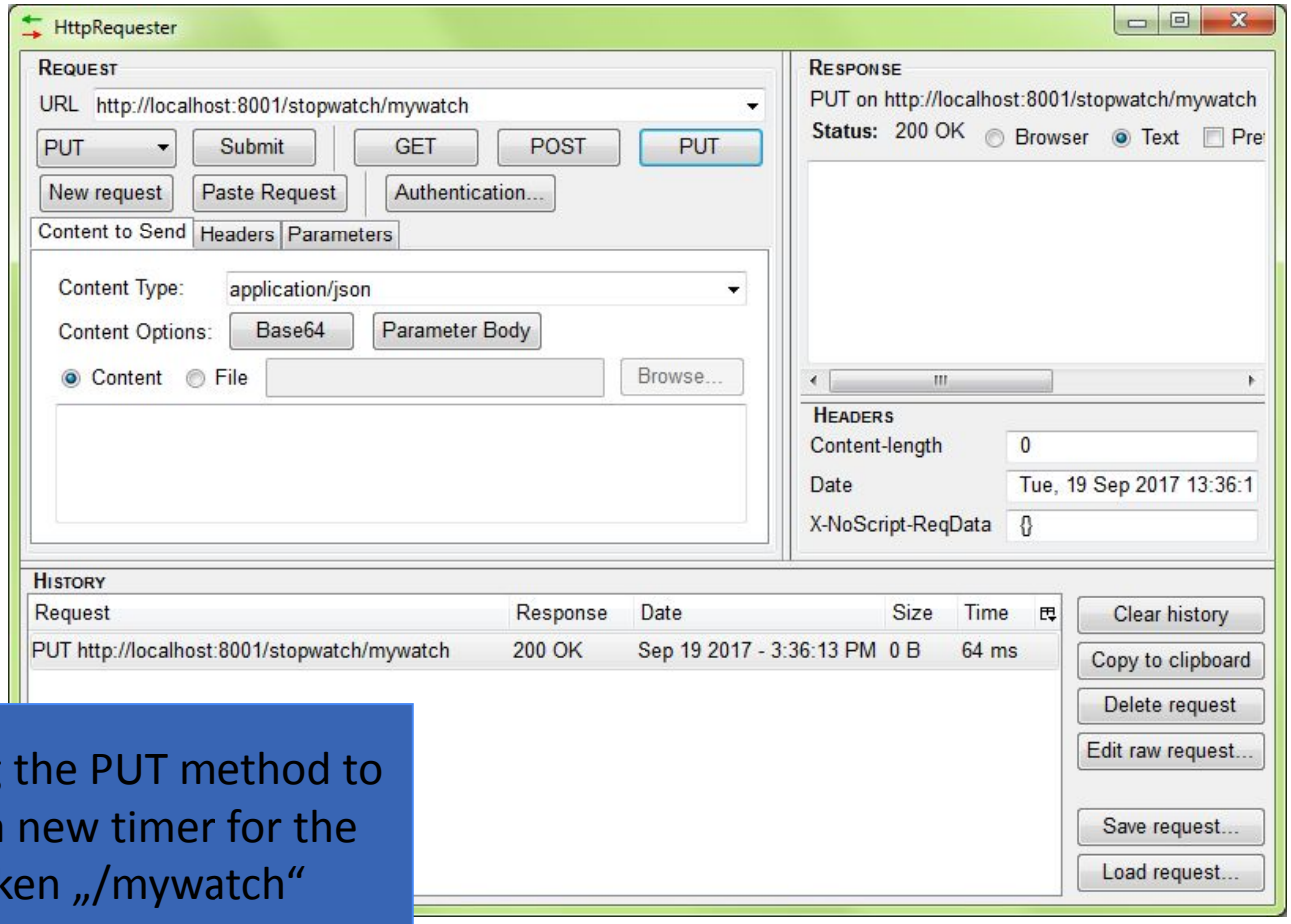
JSON

Default Output



Text

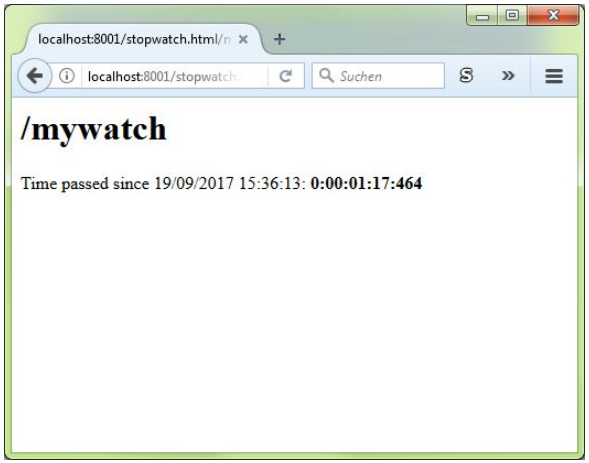
Web Service – By Example (4)



Calling the PUT method to add a new timer for the token „/mywatch“

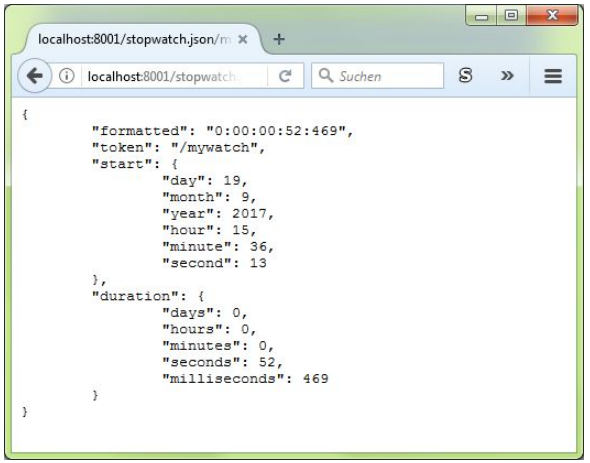
Web Service – By Example (4)

Output for Human



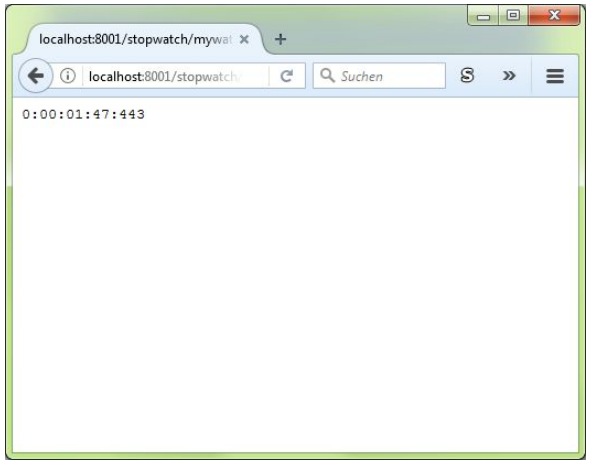
HTML

Output for Machine



JSON

Default Output



Text

Why the OMiLAB Portal is not only a HTML web-page

- There are web services available that provide different benefits, e.g.:
  - TextService: provides means of writing, storing and displaying content (text, images etc.)
  - DownloadService: provides means for uploading and downloading files (like images, PDF-documents, a modelling toolkit etc.)
  - RoleService: allows to assign different roles to users in the context of projects.
  - PSM: ties other services together to manage and display projects on OMiLAB.
- The OMiLAB portal then uses those services to provide its functionality, like creating projects, finding modelling projects, displaying details about a certain project, allowing downloads of modelling tools etc.
  - The OMiLAB portal itself provides different services, and in turn uses other services to accomplish its tasks.
  - The architecture of the OMiLAB portal is based around Microservices.
- Does that mean that e.g. deploying WordPress on my own server also “is not only a HTML web-page”?

What are „only HTML web-pages“?

- At first web-sites were like books, they had pages with the entire content.
  - Those pages were simply deployed on the server as .html files, with all their content in them.
  - Pretty much static pages, coded directly or using an HTML-Editor (Word, Dreamweaver).
  - Examples like https://tools.ietf.org/html/rfc3986 , which is pretty much an HTML document available in the web or http://info.cern.ch/hypertext/WWW/TheProject.html which is “first website” (http://info.cern.ch/)
  - They might have had some UI enhancements like changing visuals or collapsed content entries, but overall the page contained the entire content and the browser took care of visualizing it.

What are „only HTML web-pages“?

- But then things changed
  - Pages became dynamic
  - Websites turned to Content-Management-Systems (e.g. WordPress, Drupal, …), where the content of the site could be edited through the administrative-pages of the site. Similarly: YouTube
  - Online forums, Blogs etc.
  - The actual content also wasn't directly stored in the pages, instead put in some separate storage and the pages would load the content from there. This required programming/scripting languages (PHP, CGI, Java, …) to be available.
  - And they also started to provide more than just the content, they provided services (i.e. the "service" provided by the webpage is not just "show the content") like Google's image search or Google Docs or JSONLint (for validating JSON) or Wikipedia (both providing content and collaboratively working on the content)

What are „only HTML web-pages“?

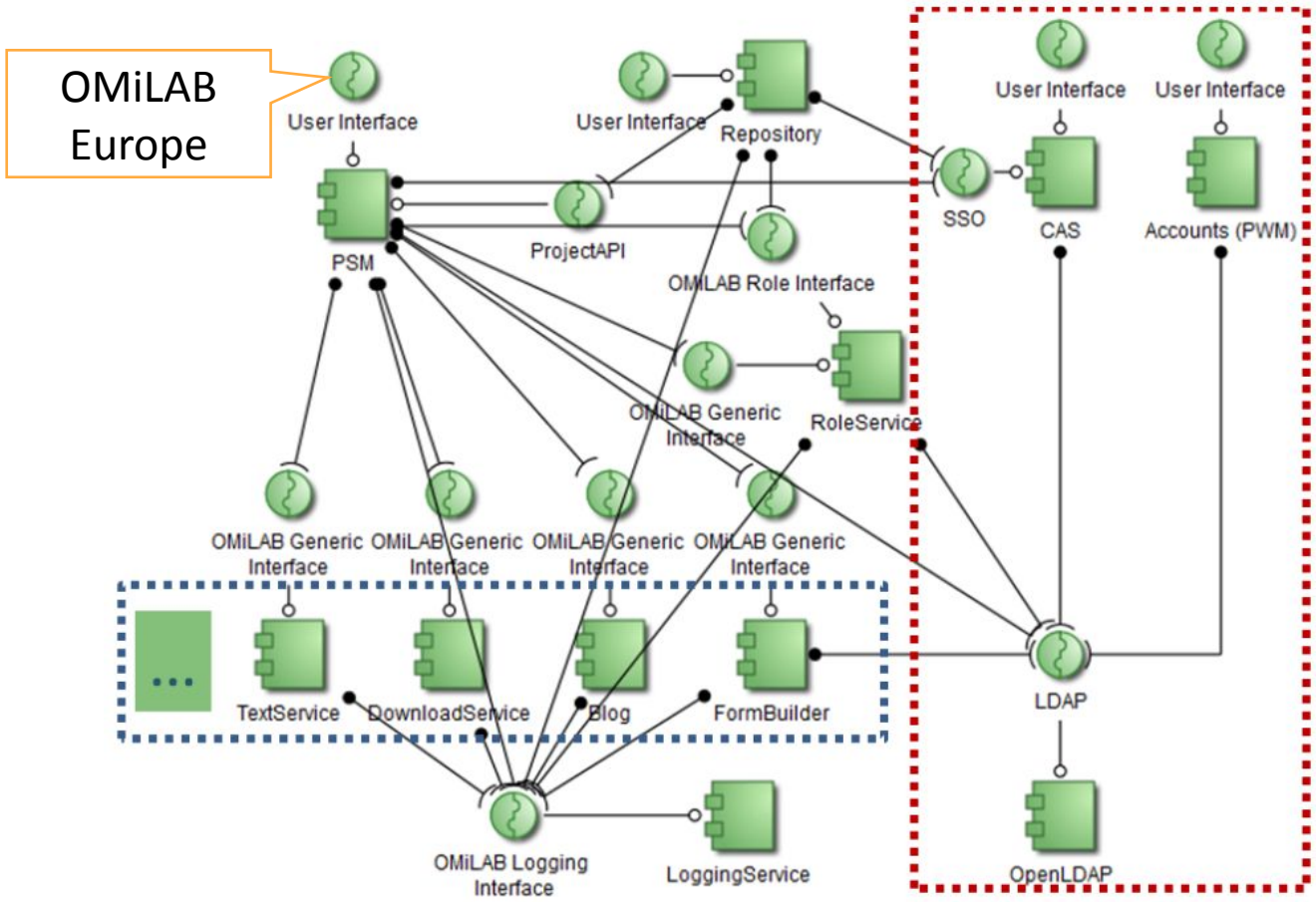
- So where does „only HTML web-pages“ end?
  - Whose perspective is relevant to answer that question?
  - From the “Client/Consumer” side … I can’t really tell if http://orf.at is more than “only HTML web-pages”.
- Are there (currently) maybe three stages of web-sites?
  - Static-content: The entire content is inside the page. This is simple file retrieval from the internet.
  - Dynamic-content: The content of a page changes (based on parameters etc.), like loading the actual text content from a database. The file to retrieve has to be generated first (e.g. www.imdb.com where the data is most likely stored in a database).
  - Interactive-content: Content that the “Client/Consumer” can interact with (e.g. JSONLint https://jsonlint.com/). Besides just the retrieved file, there is also additional dynamic behavior on the client side.
- So maybe the question is: Static/Dynamic on Server/Client side?

Microservices

- Microservice architecture style – „developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.”
- “There is a bare minimum of centralized management of these services”
- “services are independently deployable and scalable, each service also provides a firm module boundary, even allowing for different services to be written in different programming languages.”
- Services as components
  - independently deployed
  - explicit interfaces
  - remote calls with appropriate communication
  - organized around business capabilities
  - independent replacement and upgradeability
  - as products (not projects) – team owns Microservice over entire lifetime
  - decentralized data management
  - design for failure – respond to other service unavailability as gracefully as possible

[Source https://martinfowler.com/articles/microservices.html]

OMiLAB – Services and Dependencies



Add/remove as needed

Used for OMiLAB account across several OMiLABs (i.e. PSMs)

No Box: Needed to run a OMiLAB

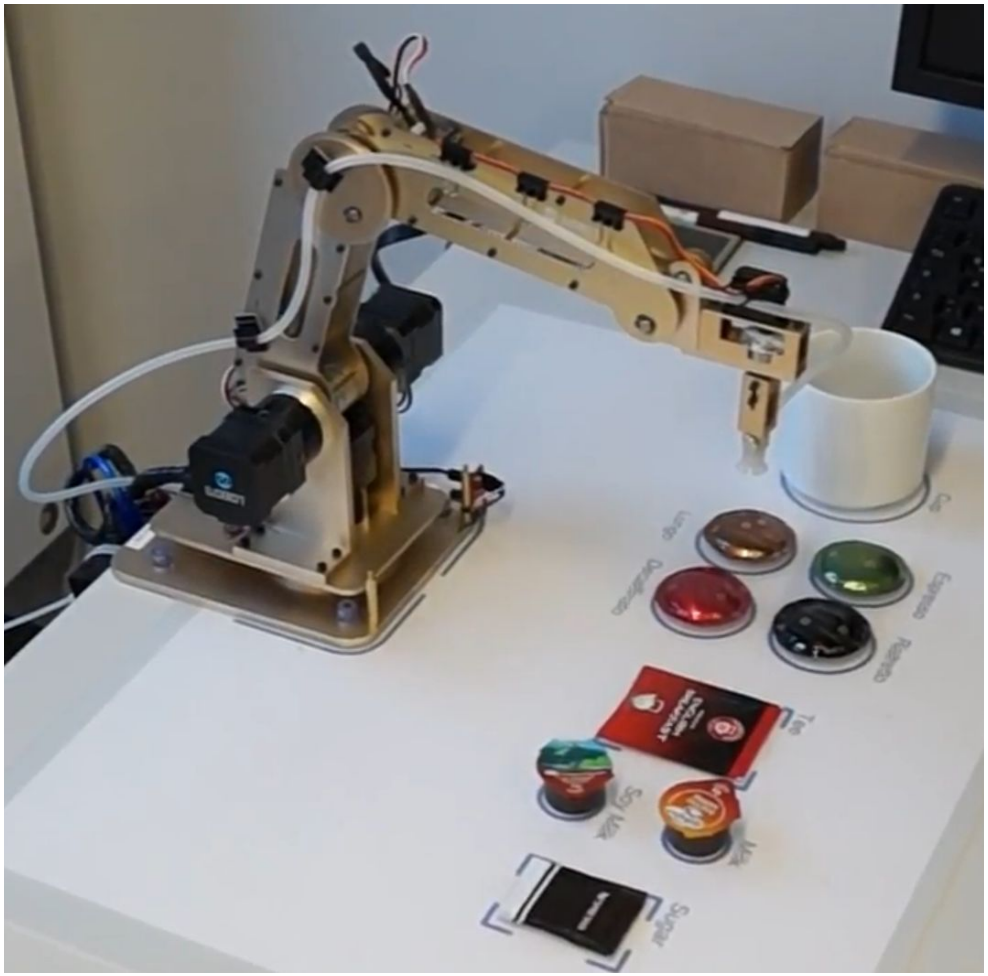
OMiLAB – Additional Services provided

- OMiLAB also provides additional services.
- Some are intended mainly for humans, like the GraphRep Generator, the Model Annotator or OMiLAB TV.
- Others provide an interface better suited to be called from other programs, like the OMiLAB Robotic Arms.
- Following an example where the OMiLAB Robotic Arm is used together with the Bee-Up tool.

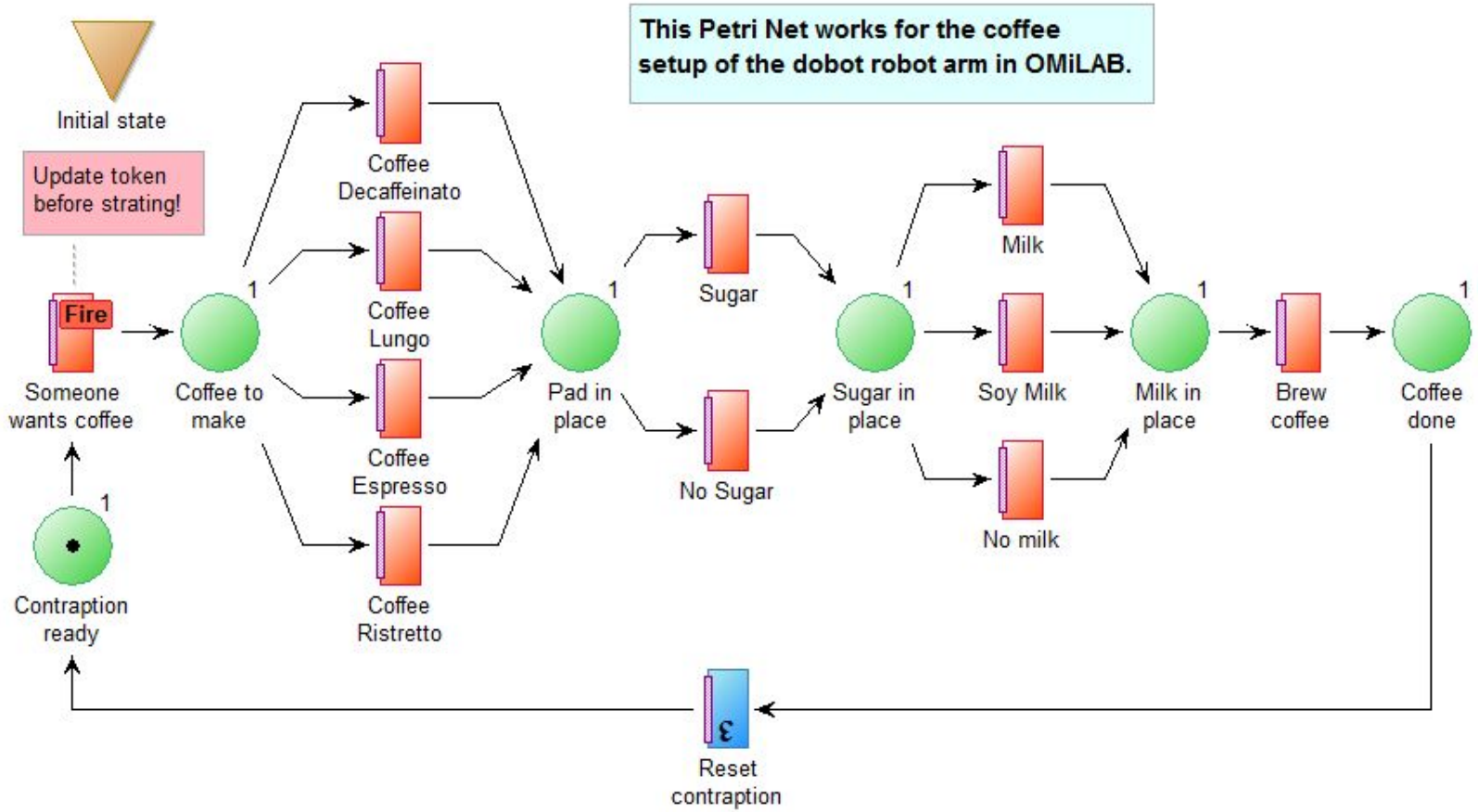
Preparing coffee

- The idea is to prepare coffee with the help of a robotic arm and a model describing the process and the different possibilities for creating the coffee.
- The robotic arm used is omiarm1, which provides a set of REST services to control the movement and operations of the arm, uses a pump to suck onto and handle light objects and has a coffee setup prepared.
- Petri Nets have been chosen as the modelling language, and the case was described using the Bee-Up modelling tool, which has different capabilities for executing/simulating Petri Nets, provides functionalities to execute code when firing transitions and can also perform HTTP calls.

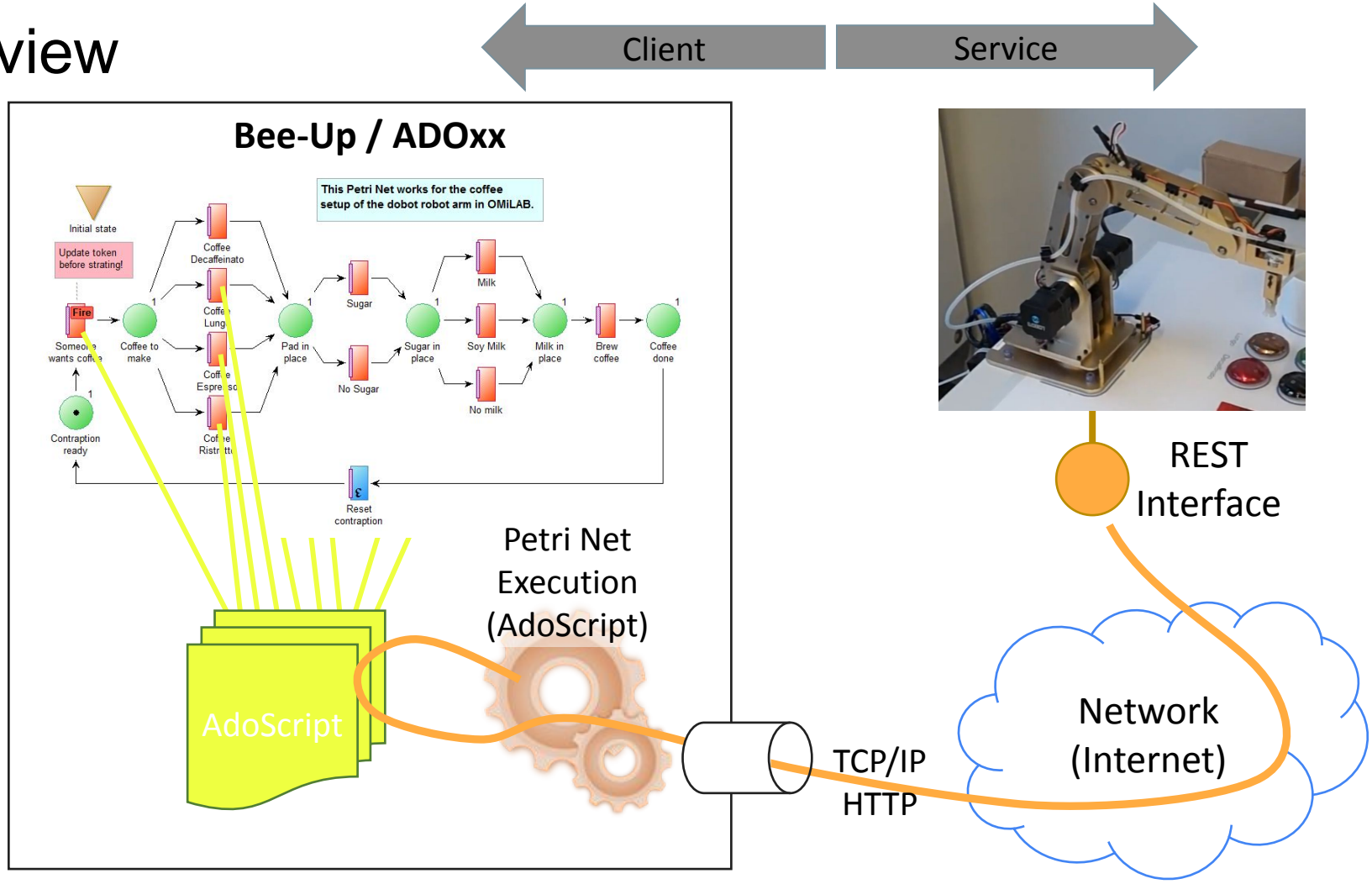
omiarm1



Petri Net Model

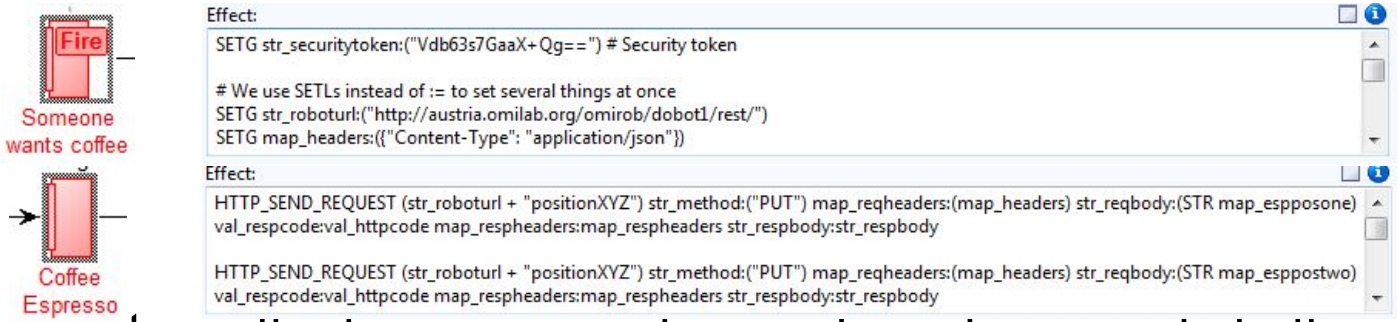


Overview


Client
Service
Bee-Up / ADOxx
This Petri Net works for the coffee setup of the dobot robot arm in OMiLAB.
Initial state
Update token before strating!
Fire
Coffee to make
Coffee Decaffeinato
Coffee Lungo
Coffee Espresso
Coffee Ristretto
Pad in place
Sugar
No Sugar
Sugar in place
Milk
Soy Milk
No milk
Milk in place
Brew coffee
Coffee done
Contraption ready
Reset contraption
Petri Net Execution (AdoScript)
AdoScript
TCP/IP
HTTP
Network (Internet)
REST Interface

Details

- The transitions contain a special „Effect“ code which is executed when they fire.



- This code sets up the basic parameters and sends requests to the robot arm to perform certain operations (move to position, turn on valve etc.)
- Additionally an extension is used which allows and simplifies HTTP calls.

Details

- The details about the position of things (e.g. where is the espresso pad) have been determined beforehand and specified in the code. Therefore the positioning template is necessary for this experiment.
- The transitions can be executed manually (by clicking on their “Fire” button) or using one of the simulation capabilities for Petri Nets available in Bee-Up.
- Upon firing the code specified in the transition is executed, which in turn calls the REST service provided by the arm to perform different operations.
- In the model one transition actually sends several calls to the robotic arm service to perform certain operations
  - e.g. the ”Coffee Espresso” transition moves the arm over the pad, picks it up, moves to the cup, drops the pad and then returns to a “neutral” position

Questions

- Is the line between where one service ends and the next one starts blurring?
  - For example the OMiLAB Robot arms → they have different URLs for their different functions, are those different services? are they the same? on which level is decided where the boundaries of one service are? on the deployment?
  - Think of it like modern computers with remote access: Several virtual computers all running on the same hardware. Are they different computers? are they the same computer (since same hardware)? Is the hardware shared or are the resources split up between the virtual computers?
  - For example: The previously presented “PiService” → if it was deployed under /constants/pi; now think there is also a different class “EService” (which returns Euler's number) which is deployed under /constants/e; So those can be considered two services. What if there was instead only one class “ConstantsService” that was deployed under /constants, and depending on whether it is followed by /pi or /e it would return a different number. Is that one service? Are those two services? Is maybe one service “an accessible unit of function that provides a benefit”, meaning that “ConstantsService” would actually be two services?
  - Is there actually a “Fuzzy Set” for “what a service is”?

Questions

- Could it be that services aren‘t quantifiable? “Service” just is, but the boundaries are fluid.
- As an analogy: like a Beach. A beach is a bunch of sand at some water.

This is a Beach

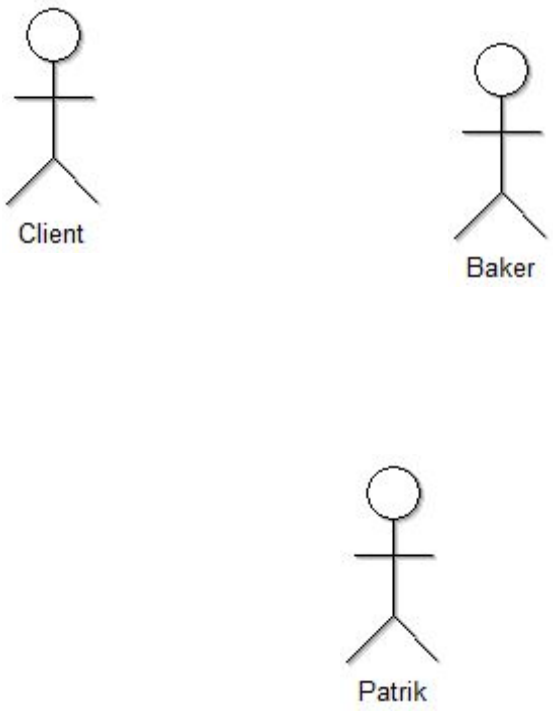
This is also a Beach

Are this both beaches together?
Is this the same beach?
Is this one beach?
Who draws the lines?

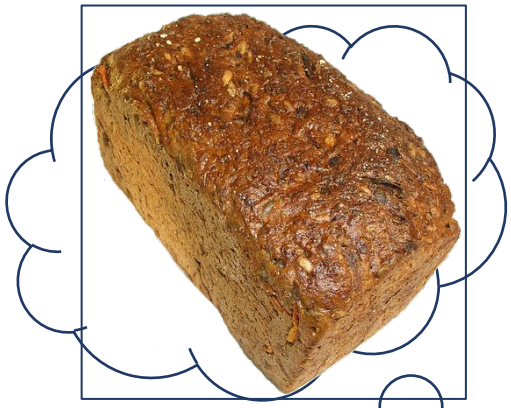
Questions

- Are Services always „somewhere else?“
  - Most certainly a service is “somebody else”, e.g. Doctor/Barber, but doctors can make house-calls.
  - Is JDOM (an external Java component that allows easier processing of XML) a service?

Other Service Examples



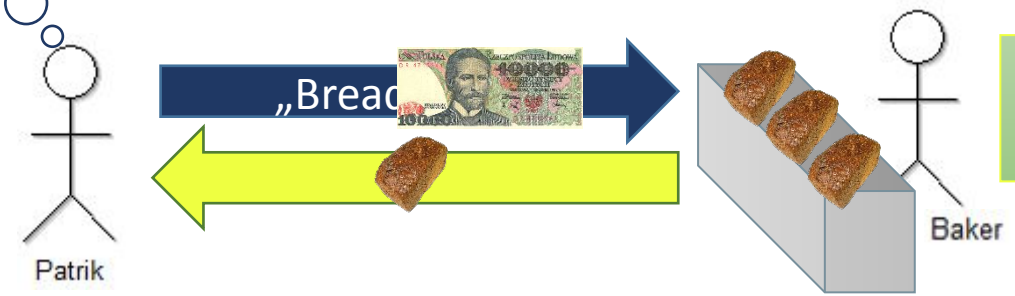
Baker



Behind every product there is a/are several service/s?

Service ?

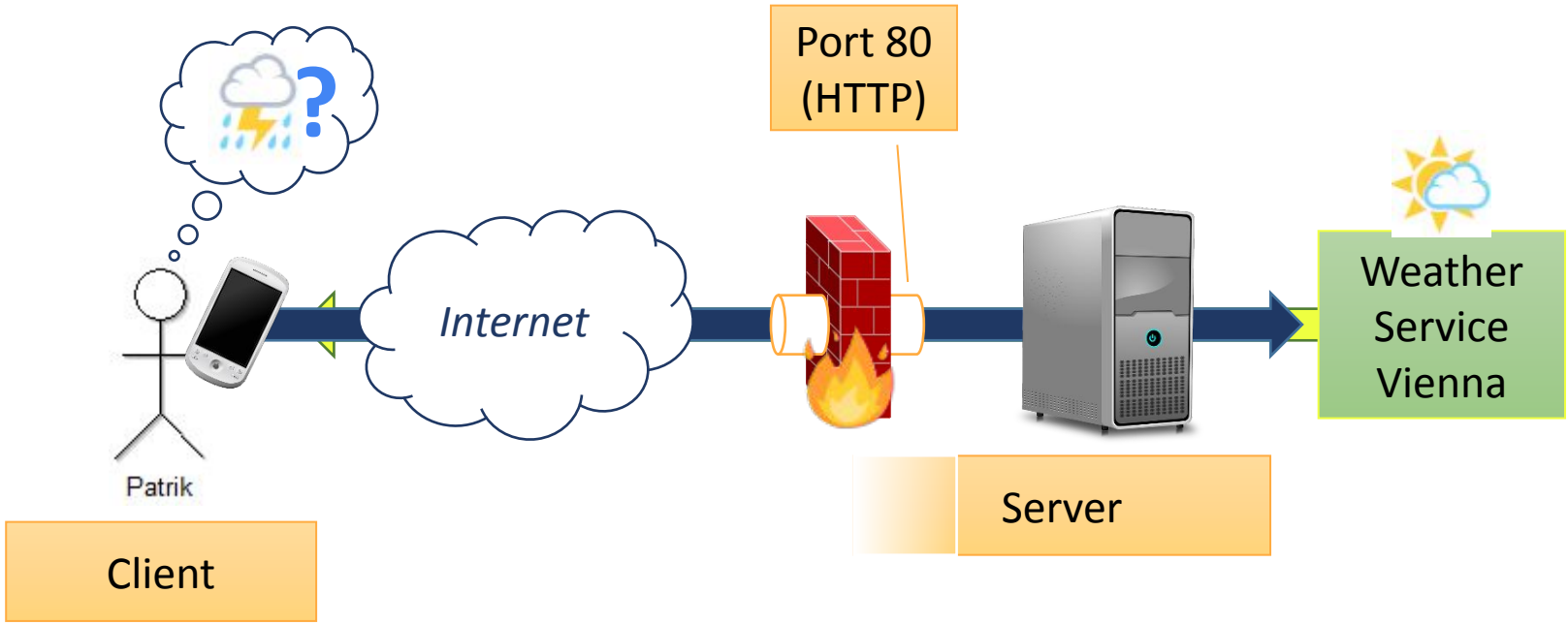
Baking and Selling



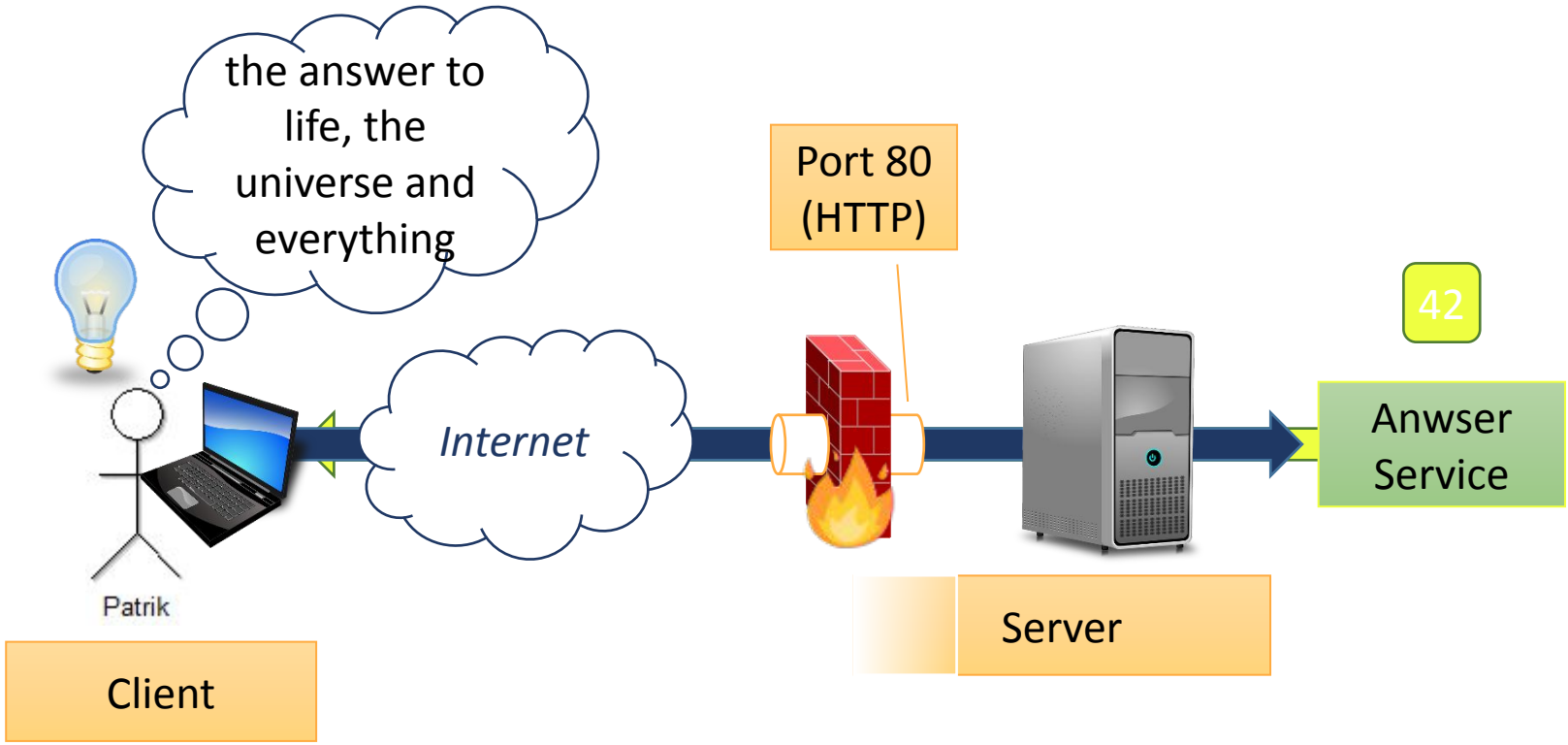
Client

Server

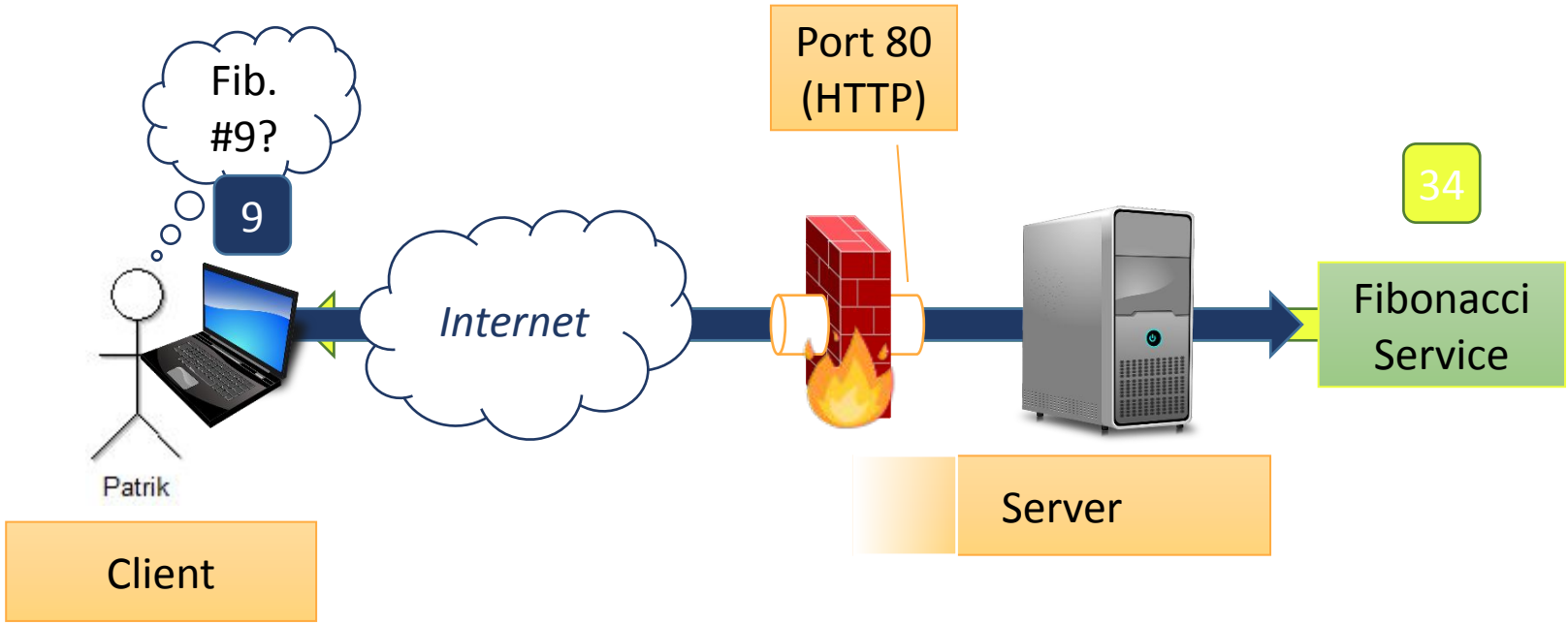
Web Service – Weather – Online



The Answer Web Service



Web Service – Fibonacci Number



Web Service – Fibonacci Number