AARHUS UNIVERSITET

Software Engineering and Architecture

Broker IPC over HTTP

History

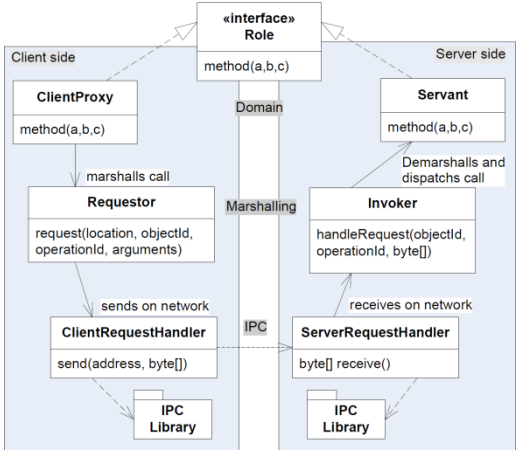
- In the 1990, there was a lot of hype about building distributed systems in the OO paradigm / Broker based
- CORBA
  - Version 1 - 1991
- Microsoft DCOM
  - About the same time ☺
- .Net remoting / Java RMI

- *Struggled with firewalls, security issues, architectural mismatch; and rather heavy-weight tooling*

AARHUS UNIVERSITET

Broker again

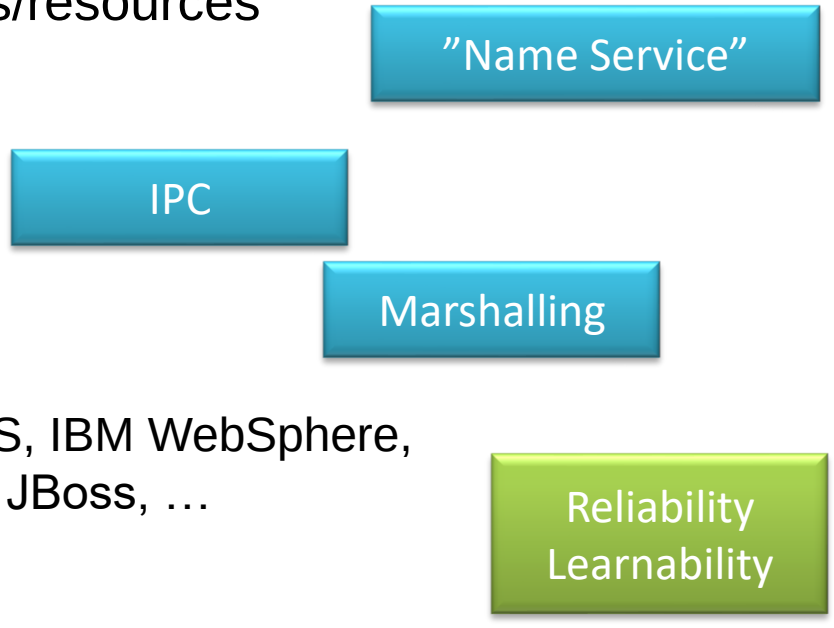
- CORBA/DCom are Broker implementations
  - IPC — *Transport bytes*
    - TCP-IP based, IIOP protocol (Corba/Java RMI)
  - Marshalling — *Encode msg*
    - IIOP
  - Proxy and Servant — *Programming*
    - *IDL compiler, generates ”stub” (proxy) and ”skeleton” (invoker + superclass for servant)*
  - Name Services — *Find object*
    - Naming Service (Corba)
    - RMIRegistry (Java)



AARHUS UNIVERSITET

Then Came WWW!

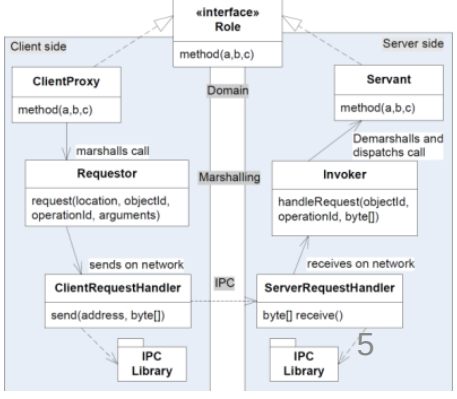
- Web was built for humans to browse web pages
- *But it is a strong infrastructure in its own right*
  - Network of named computers/resources
    - URLs
  - Well defined protocol
    - HTTP over TPC-IP,
    - … and XML
  - Strong infrastructure support
    - Apache Tomcat, Microsoft IIS, IBM WebSphere, Eclipse Jetty, Java Servlets, JBoss, …



AARHUS UNIVERSITET

Many of the pieces

- WWW as foundation for a Broker?
  - IPC — *Transport bytes*
    - TCP-IP connected machines talking HTTP
  - Marshalling — *Encode msg*
    - XML in HTTP messages
  - Proxy and Servant — *Programming*
    - Missing – not the intent of WWW
  - Location and Naming — *Find object*
    - URLs defining resources = data/information



AARHUS UNIVERSITET

The missing piece

- What to do with no programming / domain layer?
- Option 1: Program without it

Tedios, error prone, low level

  - Implement ‘doGet’ and ‘doPost’ (servlets), demarshall XML
  - Implement HTTP requests (client), marshall XML

- Option 2: Make a Broker on top of www
  - Simple Object Access Protocol / SOAP
  - Or FRDS.Broker’s URI Tunneling Broker

Reuse, but...

- Option 3: Do something completely different ☺
  - REST…

AARHUS UNIVERSITET

WebServices: SOAP

Simple Object Access Protocol

(Simple???)

The short version

(A longer one at the end of the slide deck)

AARHUS UNIVERSITET

Broker

- The Broker implementations in ‘web services’ tech stack
  - IPC — *Transport bytes*
    - TCP-IP based, **HTTP** protocol
  - Marshalling — *Encode msg*
    - **SOAP** – on the wire XML format
      - “simple object access protocol”
  - Proxy and Servant — *Programming*
    - **WSDL** = Web Service Description Language (XML)
  - Location and Naming — *Find object*
    - **UDDI** = Universal Description, Discovery and Integration

AARHUS UNIVERSITET

The Short Version

- You generate WSDL from your interface
  - Heavy tooling, and produce tons of un-analyzable code ☹
  - *Vendor lock-in* to UDDI servers
  - Does not utilize HTTP's built-in potential at all, it is just a way to punch through firewalls…

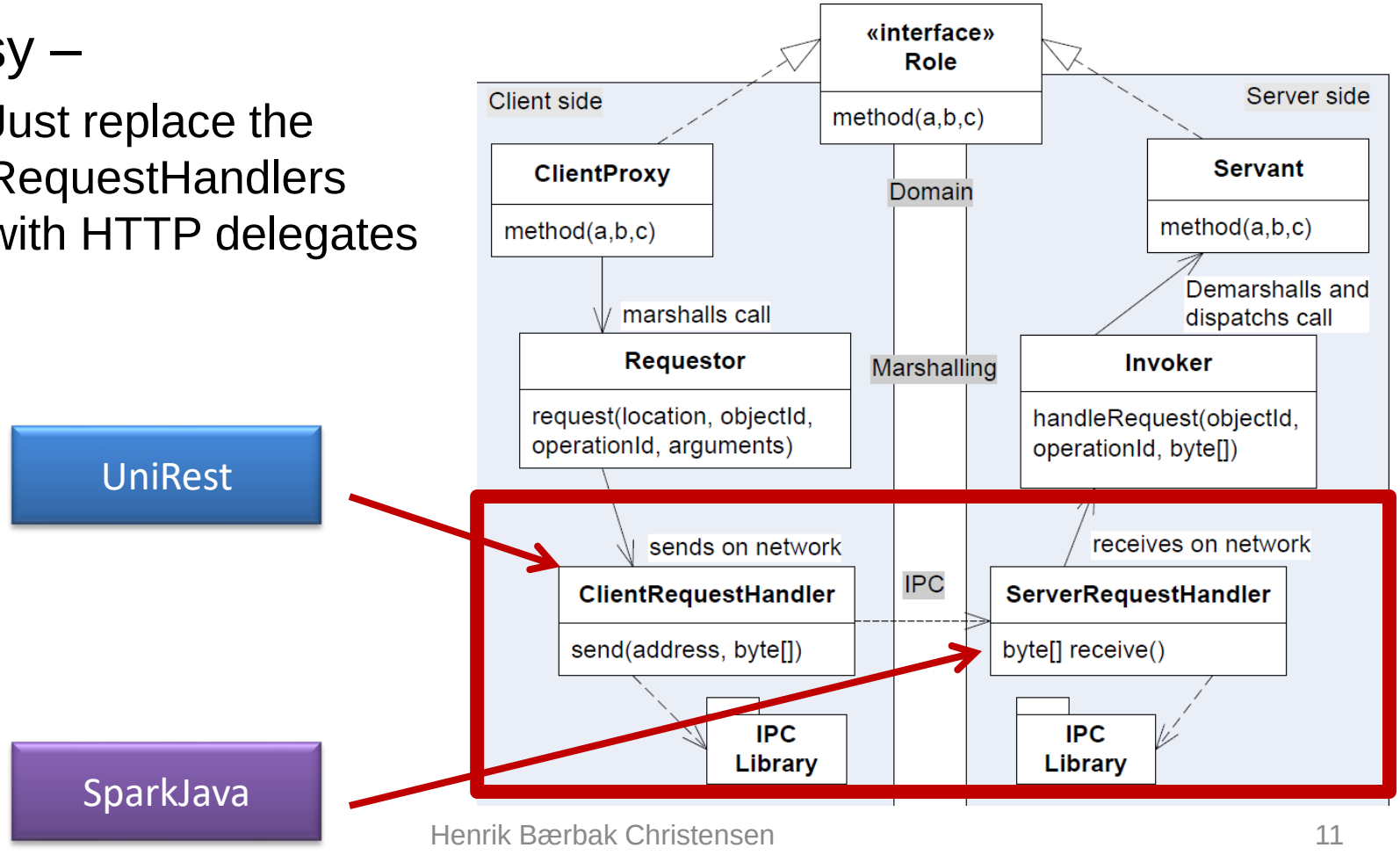
AARHUS UNIVERSITET

FRDS.Broker using HTTP

AARHUS UNIVERSITET

New Delegates in Broker

- Easy –
  - Just replace the RequestHandlers with HTTP delegates



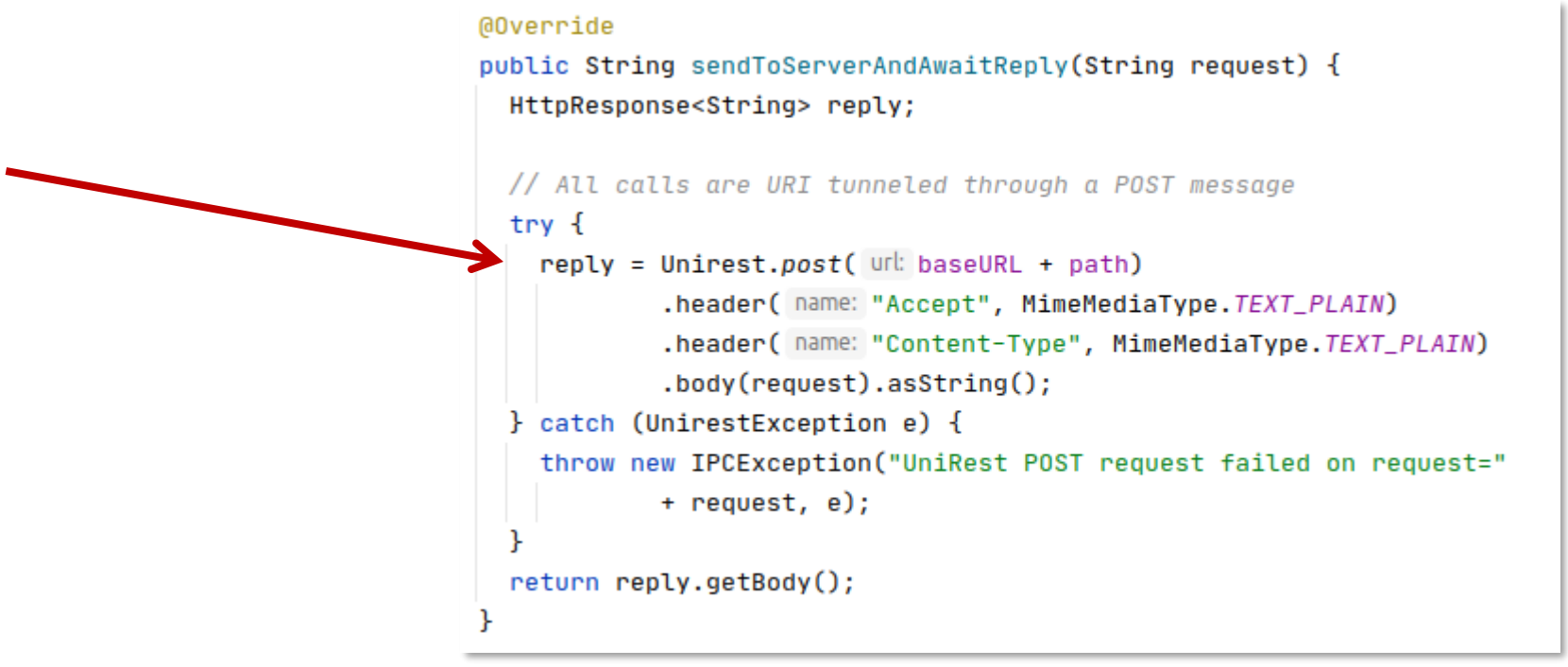
POST Command Objects

- All method calls are considered *command objects* that the ClientRequestHandler POST to one particular resource on the HTTP Server

- POST /hotstone
  - Body: Marshalled request (objId, OpName, arguments)

```
elServerRequestHandler :: method=POST, context=request, request
="operationName":"game_get-deck-size","payload":"[\"FINDUS\"]","objectId":"one-game","versionIdentity":1}
elServerRequestHandler :: method=handleRequest, context=reply,
reply={"payload":"4","statusCode":200,"versionIdentity":1}, responseTime_ms=1
2022-11-23T10:39:01.657+01:00 [INFO] frds.broker.ipc.http.UriTunnelServerRequestHandler :: method=POST, context=request, request
={"operationName":"game_get-hand-size","payload":"[\"PEDDERSEN\"]","objectId":"one-game","versionIdentity":1}
2022-11-23T10:39:01.658+01:00 [INFO] frds.broker.ipc.http.UriTunnelServerRequestHandler :: method=handleRequest, context=reply,
reply={"payload":"3","statusCode":200,"versionIdentity":1}, responseTime_ms=1
2022-11-23T10:39:01.660+01:00 [INFO] frds.broker.ipc.http.UriTunnelServerRequestHandler :: method=POST, context=request, request
={"operationName":"game_get-field-size","payload":"[\"PEDDERSEN\"]","objectId":"one-game","versionIdentity":1}
2022-11-23T10:39:01.661+01:00 [INFO] frds.broker.ipc.http.UriTunnelServerRequestHandler :: method=handleRequest, context=reply,
reply={"payload":"0","statusCode":200,"versionIdentity":1}, responseTime_ms=1
```

- POST to URL with the marshalled request



```
@Override
public String sendToServerAndAwaitReply(String request) {
  HttpResponse<String> reply;

  // All calls are URI tunneled through a POST message
  try {
    reply = Unirest.post( url: baseURL + path)
            .header( name: "Accept", MimeMediaType.TEXT_PLAIN)
            .header( name: "Content-Type", MimeMediaType.TEXT_PLAIN)
            .body(request).asString();
  } catch (UnirestException e) {
    throw new IPCException("UniRest POST request failed on request="
            + request, e);
  }
  return reply.getBody();
}
```

- *This is the standard implementation in the Broker library*

And SRH

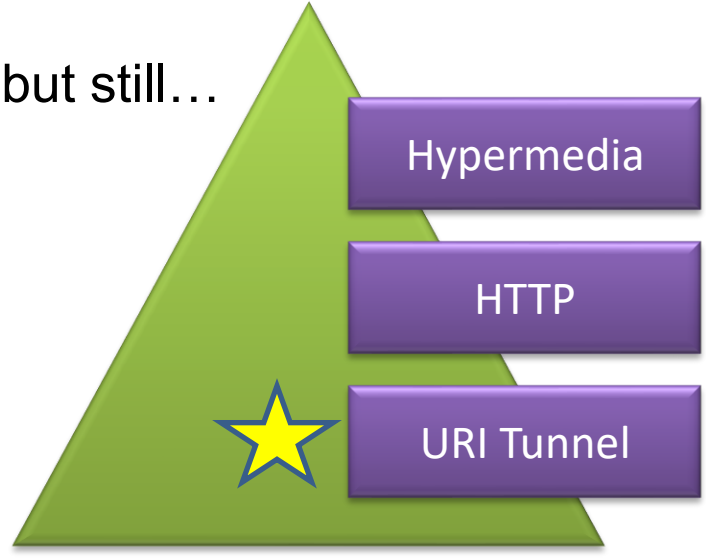
- Accept POST
- Do upcall
- … and reply
- *Standard impl. in the library*

```java
@Override
public void start() {
  // Set the port to listen to
  port(port);

  // POST is for all incoming requests
  post(tunnelRoute, (req,res) -> {
    long startTime = System.currentTimeMillis();
    String marshalledRequest = req.body();

    // The incoming marshalledRequest is the marshalled request to the invoker
    // Log the request, using a key-value format
    logger.info("method=POST, context=request, request={}", marshalledRequest);

    String reply = invoker.handleRequest(marshalledRequest);

    // Store the last verb and status code to allow spying during test
    lastVerb = req.requestMethod();

    // The reply is opaque - so we have no real chance of setting a proper
    // status code.
    res.status(HttpServletResponse.SC_OK);
    // Hmm, we also do not know the actual marshalling format but
    // just know it is textual
    res.type(MimeMediaType.TEXT_PLAIN);

    // response time in milliseconds for invoker upload is calculated
    long responseTime = System.currentTimeMillis() - startTime;
    logger.info("method=handleRequest, context=reply, reply={}, responseTime_ms={}",
            reply, responseTime);

    return reply;
  });
}
```

AARHUS UNIVERSITET

Discussion

- *All* methods are translated to POST on a single path /tunnel
  - You can set another path in the Broker, but still…
- That is, the Broker do not use
  - HTTP verbs at all
  - Resources/paths for anything



- (But it does use the HTTP Status codes)
- **WebServices/SOAP does the same thing.**

AARHUS UNIVERSITET

Discussion

- Actually the URI Tunnel code is quite a lot smaller than our socket code
  - And it is also *multi-threaded*, as SparkJava is based on Jetty which has a multi-threaded implementation…
    - Starts with a pool of 8 threads, but can increase to 200…

- Again, reusing well engineered frameworks saves time, money, and agony ☺

AARHUS UNIVERSITET

WebServices: SOAP

Simple Object Access Protocol

(Simple???)

The long version…

AARHUS UNIVERSITET

Broker

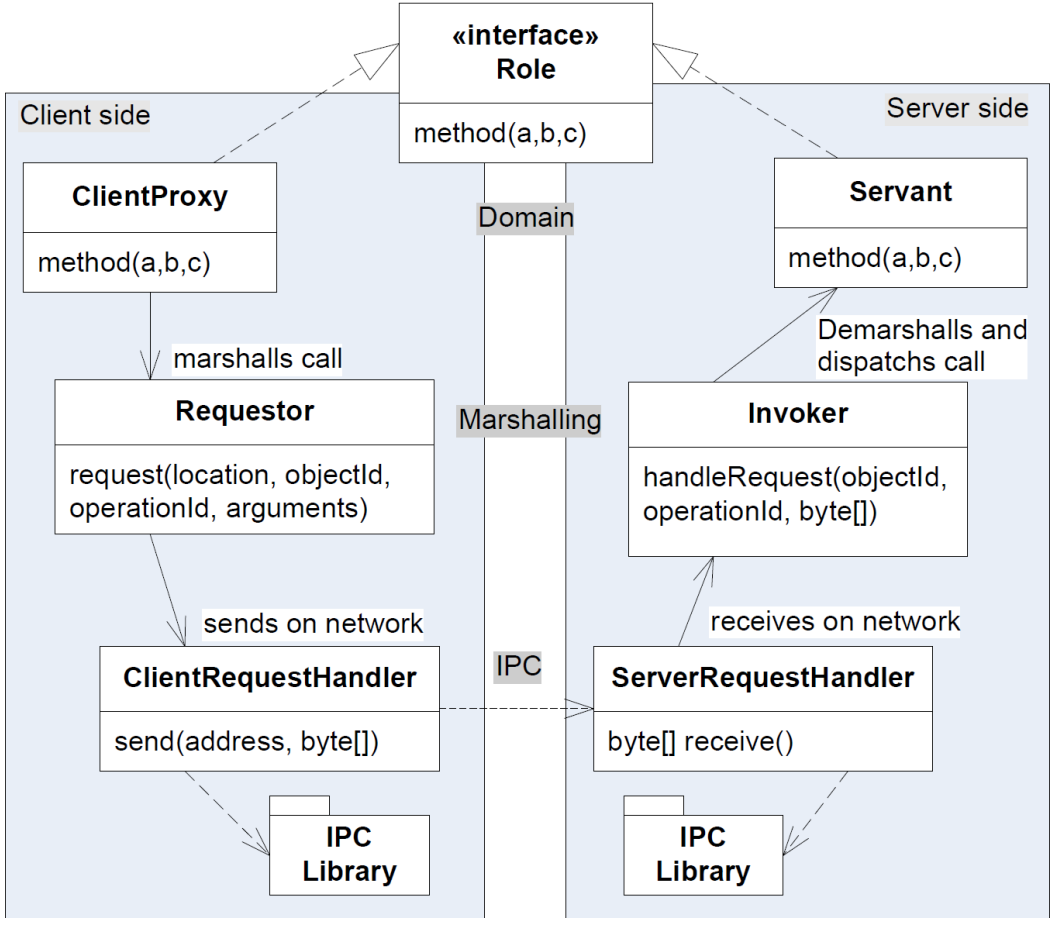
- The Broker implementations
  - IPC — *Transport bytes*
    - TCP-IP based, **HTTP** protocol
  - Marshalling — *Encode msg*
    - **SOAP** – on the wire XML format
  - Proxy and Servant — *Programming*
    - **WSDL** = Web Service Description Language (XML)
  - Location and Naming — *Find object*
    - **UDDI** = Universal Description, Discovery and Integration

The three layers

WSDL

URLs + SOAP

HTTP



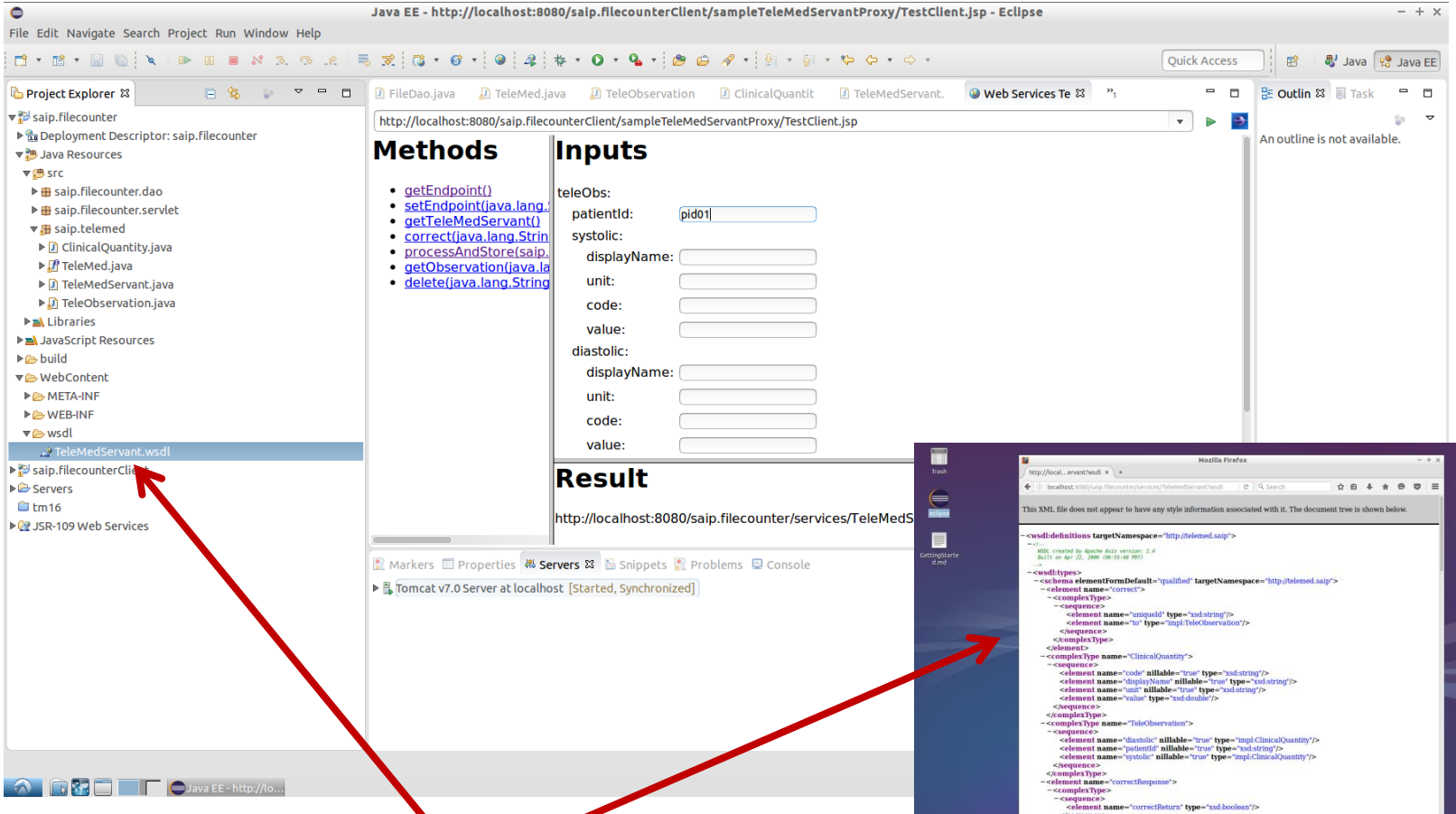
AARHUS UNIVERSITET

Example: TeleMed in SOAP

- Learning Goal: *Produce WSDL for TeleMed*
- I…
  - Browsed heavily to find Eclipse tutorials
    - http://www.vogella.com/tutorials/EclipseWTP/article.html
    - http://www.java2blog.com/2013/03/soap-web-service-example-in-java-using.html
  - Copy-n-Pasted TeleMed interface + few Domain classes into project on a VM (avoid polluting my machine's IDE)
  - Nullified actual implementations
    - No business functionality, not the architectural question

AARHUS UNIVERSITET

Example



WSDL

```
- <wsdl:definitions targetNamespace="http://telemed.saip">
  - <!--
      WSDL created by Apache Axis version: 1.4
      Built on Apr 22, 2006 (06:55:48 PDT)
    -->
  - <wsdl:types>
    - <schema elementFormDefault="qualified" targetNamespace="http://telemed.saip">
      - <element name="correct">
        - <complexType>
          - <sequence>
              <element name="uniqueId" type="xsd:string"/>
              <element name="to" type="impl:TeleObservation"/>
            </sequence>
          </complexType>
        </element>
      - <complexType name="ClinicalQuantity">
        - <sequence>
            <element name="code" nillable="true" type="xsd:string"/>
            <element name="displayName" nillable="true" type="xsd:string"/>
            <element name="unit" nillable="true" type="xsd:string"/>
            <element name="value" type="xsd:double"/>
          </sequence>
        </complexType>
      - <complexType name="TeleObservation">
        - <sequence>
            <element name="diastolic" nillable="true" type="impl:ClinicalQuantity"/>
            <element name="patientId" nillable="true" type="xsd:string"/>
            <element name="systolic" nillable="true" type="impl:ClinicalQuantity"/>
          </sequence>
        </complexType>
```

Data types

```
- <wsdl:binding name="TeleMedServantSoapBinding" type="impl:TeleMedServant">
    <wsdlsoap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
  - <wsdl:operation name="correct">
      <wsdlsoap:operation soapAction=""/>
    - <wsdl:input name="correctRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output name="correctResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="processAndStore">
      <wsdlsoap:operation soapAction=""/>
    - <wsdl:input name="processAndStoreRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output name="processAndStoreResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="getObservation">
      <wsdlsoap:operation soapAction=""/>
    - <wsdl:input name="getObservationRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output name="getObservationResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  - <wsdl:operation name="delete">
      <wsdlsoap:operation soapAction=""/>
    - <wsdl:input name="deleteRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
    - <wsdl:output name="deleteResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
- <wsdl:service name="TeleMedServantService">
  - <wsdl:port binding="impl:TeleMedServantSoapBinding" name="TeleMedServant">
      <wsdlsoap:address location="http://localhost:8080/saip.filecounter/services/TeleMedServant"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Operations

Binding

AARHUS UNIVERSITET

Another example

- IHE XDSb Repository WSDL

```
− <wsdl:definitions name="DocumentRepositoryService" targetNamespace="http://tempuri.org/">
  − <wsp:Policy wsu:Id="XDSRepository_HTTP_Endpoint_policy">
    − <wsp:ExactlyOne>
      − <wsp:All>
          <wsoma:OptimizedMimeSerialization/>
          <wsaw:UsingAddressing/>
        </wsp:All>
      </wsp:ExactlyOne>
    </wsp:Policy>
    <wsdl:import namespace="urn:ihe:iti:xds-b:2007" location="http://a02447:1026/XdsService/XDSRepository?wsdl=wsdl0"/>
    <wsdl:types/>
  − <wsdl:binding name="XDSRepository_HTTP_Endpoint" type="i0:XDSRepository">
      <wsp:PolicyReference URI="#XDSRepository_HTTP_Endpoint_policy"/>
      <soap12:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    − <wsdl:operation name="ProvideAndRegisterDocumentSet">
        <soap12:operation soapAction="urn:ihe:iti:2007:ProvideAndRegisterDocumentSet-b" style="document"/>
      − <wsdl:input>
          <soap12:body use="literal"/>
        </wsdl:input>
      − <wsdl:output>
          <soap12:body use="literal"/>
        </wsdl:output>
      </wsdl:operation>
```

Why not?

- Basically WSDL+SOAP+UDDI is the Broker which simply use WWW and HTTP as *raw IPC*
  - Aka **URI Tunneling**
  - Avoid the firewall and security issues though

- But it is heavyweight
  - Tool intensive to make even simple service
  - Bulky message format, low analyzability
  - Does not utilize HTTP's built-in potential at all, it is just a way to punch through firewalls…

AARHUS UNIVERSITET

Richardson's Maturity model

- From low maturity to high maturity
  - URI Tunnel
    - Just use HTTP as IPC layer
      - SOAP, WSDL, WebServices
      - And our URI Tunnel Broker!
  - HTTP
    - Use CRUD Verbs on resources
  - Hypermedia
    - Use links to define workflows
    - Aka 'HATEOAS'