Technische Universität Darmstadt

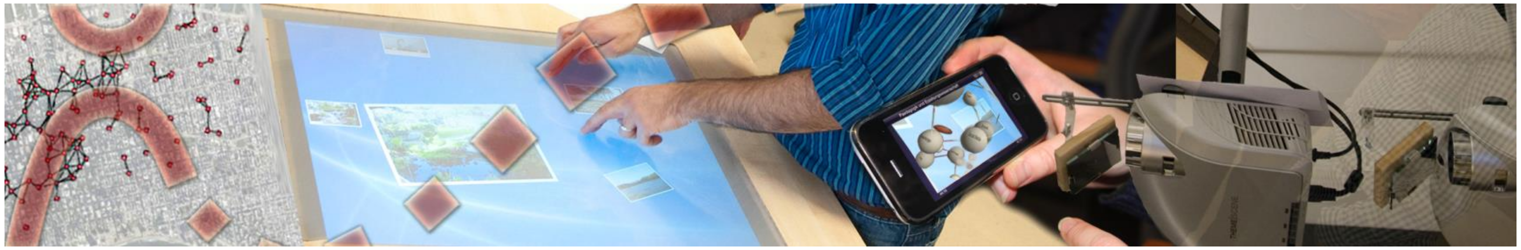
Telecooperation Lab
Prof. Dr. Max Mühlhäuser

TK1: Distributed Systems - Programming & Algorithms

Chapter 2: Distributed Programming

Section 1: Mainstream Paradigms

Lecturer: **Prof. Dr. Max Mühlhäuser, Dr. Immanuel Schweizer, Dr. Benedikt Schmidt**



2.1: MAINSTREAM PARADIGMS

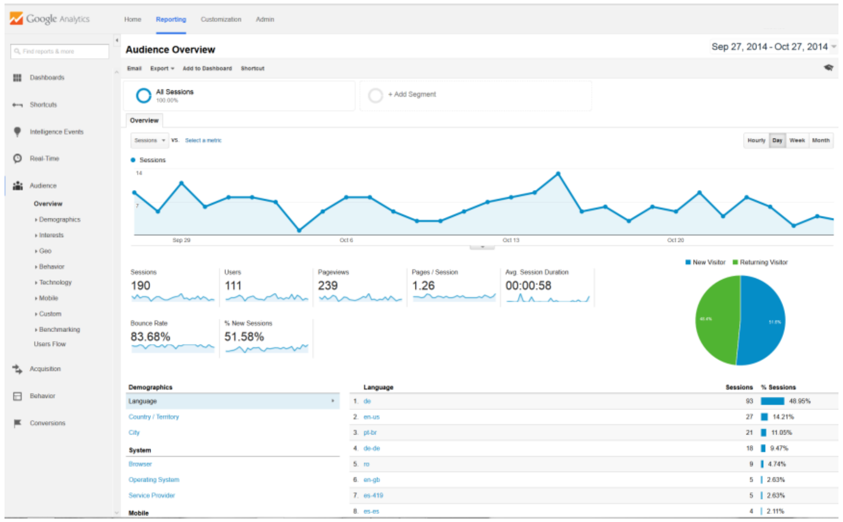
(1) IPC: Interprocess Communication

(2) Inlet: Distributed Programming Languages

(3) Web Services

TECHNISCHE UNIVERSITÄT DARMSTADT

Communication between Services



Audience Overview

Sep 27, 2014 - Oct 27, 2014

All Sessions 100.00%

Sessions 190
Users 111
Pageviews 239
Pages / Session 1.26
Avg. Session Duration 00:00:58
Bounce Rate 83.68%
% New Sessions 51.58%

Demographics
Language
Country / Territory
City
System
Browser
Operating System
Service Provider
Mobile

	Language	Sessions	% Sessions
1.	de	93	48.95%
2.	en-us	27	14.21%
3.	pt-br	21	11.05%
4.	de-de	18	9.47%
5.	ro	9	4.74%
6.	en-gb	5	2.63%
7.	es-419	5	2.63%
8.	es-es	4	2.11%

Web Applications

- Computer <-> Human

Web Services

- Computer <-> Computer

Web Services

External Data Representation:	Interface Description:
SOAP	WSDL
XML	XML
HTTP or other transport	often provided over HTTP

- SOAP
  - Simple Object Access Protocol is no longer the official abbreviation since Version 1.2: not really *simple* & does not allow access to *objects*
  - Structured documents between systems, Remote Procedure Calls
- Web Service Description Language (WSDL)
  - Describes interface, etc. of a web service
- Universal Description, Discovery and Integration (UDDI)
  - Directory Service (Broker, Yellow Pages)

WSDL

- XML Document with the following elements:
  - **Types**
    - Definition of new types
  - **Message**
    - Definition of incoming and outgoing messages (request/reply)
  - **Port Type**
    - groups actions that logically belong together
  - **Operations**
    - Definition of the actions supported by a service
    - Definition of the messages expected and generated by an action
    - Definition of the interaction pattern (request/response, oneway, …)
  - **Binding**
    - defines protocol details, e.g., SOAP over HTTP
  - **Service**
    - Container for ports
  - **Port**
    - definition of the individual endpoints under which a service can be used

WSDL Elements

- **Types**: Definition of new types
- elementary types defined by XML schema
  (xmlns:xsd="**http://www.w3.org/2001/XMLSchema**")

```
<types>
  <xsd:schema>
    <!-- inserted: ShapeListService_schema1.xsd -->
    <xsd:complexType name="graphicalObject">
      <xsd:sequence>
        <xsd:element name="name" type="xsd:string"/>
                …
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

WSDL Elements

- **Message**: Definition of incoming and outgoing messages

```
<message name="newShape">
  <part name="arg0" type="tns:graphicalObject"/>
 </message>
 <message name="newShapeResponse">
  <part name="return" type="xsd:int"/>
 </message>

 <message name="numberOfShapes"/>
 <message name="numberOfShapesResponse">
  <part name="return" type="xsd:int"/>
 </message>

 …
```

WSDL Elements

- **Port Type:** groups operations that belong together ("interface")
- **Operation**: defines
  - Interaction pattern: request/reply, oneway
  - Structure of Request messages
  - Structure of Reply messages

```
<portType name="ShapeList">
  <operation name="newShape">
    <input message="tns:newShape"/>
    <output message="tns:newShapeResponse"/>
  </operation>
  <operation name="numberOfShapes">
    <input message="tns:numberOfShapes"/>
    <output message="tns:numberOfShapesResponse"/>
  </operation>
  …
 </portType>
```

WSDL Elements

- **Binding**: Defines protocol details; here: SOAP over HTTP
- **style/use**: Defines how SOAP messages will be generated
  - rpc/literal: no xsd-types in messages
  - rpc/encoded: xsd-types included in messages (for each argument, etc.)
  - document/*: include schema in SOAP → msg can be XML-validated directly

```
<binding name="ShapeListPortBinding" type="tns:ShapeList">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc"/>
    <operation name="newShape">
      <soap:operation soapAction="urn:newShape"/>
      <input>
        <soap:body use="literal" namespace="http://tk.informatik.tu-darmstadt.de"/>
      </input>
      <output>
        <soap:body use="literal" namespace="http://tk.informatik.tu-darmstadt.de"/>
      </output>
    </operation>
    …
</binding>
```

WSDL Elements

- **Service**: Container for Ports
- **Port**: Defines individual endpoints that allow to access the service
  - soap:address will be filled in automatically by the WS container at deployment time

```
<service name="ShapeListService">
  <port name="ShapeListPort" binding="tns:ShapeListPortBinding">
    <soap:address location="REPLACE_WITH_ACTUAL_URL"/>
  </port>
</service>
```

WS: Service Usage

- Use **wsimport** to generate stubs from WSDL
  - ShapeList.java
    - Java interface for ShapeList
  - ShapeListService.java
    - Factory class providing client stubs
    - Client stubs implement interface ShapeList
  - GraphicalObject
    - Schema-derived class GraphicalObject
  - ObjectFactory
    - Factory for schema-derived classes
- The generated classes are now used in client code

WS: Example Service

```
package de.tu_darmstadt.informatik.tk;

import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.soap.SOAPBinding;

@WebService(targetNamespace="http://tk.informatik.tu-darmstadt.de", name="ShapeList")
@SOAPBinding(style=SOAPBinding.Style.RPC)
public class ShapeListImpl
{
    public int newShape(GraphicalObject g) { … }
    public int numberOfShapes() { … }
    public int getVersion() { … }
    public int getGOVersion(int i) { … }
    public GraphicalObject getAllState(int i) { … }
}
```

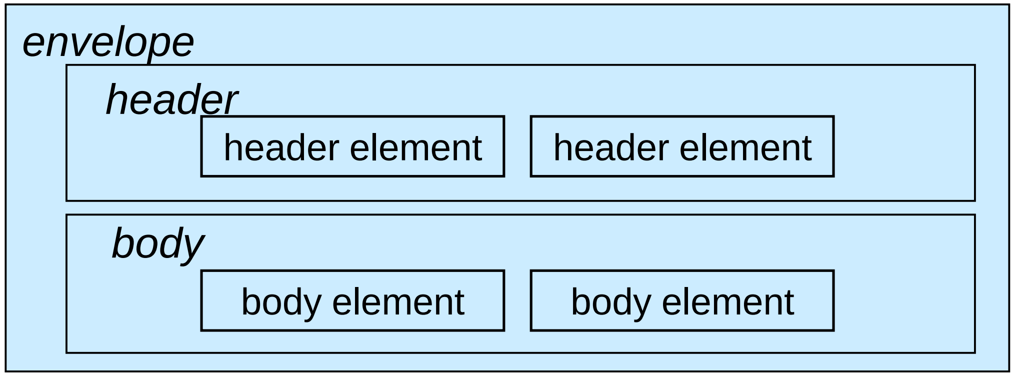
WS: Example Client

```
import java.net.URL;
import javax.xml.namespace.QName;
import de.tu_darmstadt.informatik.tk.*;

public class Client {
    public static void main(String args[]) {
        try {
            ShapeListService service = new ShapeListService(
                        new URL("http://localhost:8080/shapelist/shapelist?wsdl"),
                        new QName("http://tk.informatik.tu-darmstadt.de",
                                  "ShapeListService"));
            ShapeList shapeList = service.getShapeListPort();
            System.out.println(shapeList.getVersion());
        }
        catch(Exception x) {
            x.printStackTrace();
        }
    }
}
```

SOAP

- Communication between nodes via messages
- Message consists of
  - Envelope
    - The enclosing entity of a message
    - Defines namespaces
  - Header
    - Contains metadata for the body
    - Many WS-* extensions add additional information here
  - Body
    - Contains the payload
    - Further specifications define the body structure



SOAP: Example

RPC Request

```
POST /shapelist/shapelist HTTP/1.1
Content-Type: text/xml;charset="utf-8"
User-Agent: JAX-WS RI 2.1.4-b01-
Host: localhost:8080
Connection: keep-alive
Content-Length: 182

<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getVersion xmlns:ns2="http://tk.informatik.tu-darmstadt.de"/>
  </S:Body>
</S:Envelope>
```

RPC Reply

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Transfer-Encoding: chunked

e8
<?xml version="1.0" ?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <ns2:getVersionResponse xmlns:ns2="http://tk.informatik.tu-darmstadt.de">
      <return>1</return>
    </ns2:getVersionResponse>
  </S:Body>
</S:Envelope>
0
```

UDDI

- XML-based registry for businesses worldwide
  - **White pages**
    - Information about a business
    - Name, contact information, textual description
  - **Yellow pages**
    - Information about the business sector through standardized industry classification systems
      - North American Industrial Classification System (NAICS)
      - Universal Standard Products and Services Classification (UNSPSC)
      - Geographic Classification System (GCS)
  - **Green pages**
    - Technical information about services
    - Reference to WSDL description
- UDDI also defines a Web Service API for accessing it
  - Java API: JAXR (registry)
- Original plan was a global, replicated directory for all Web Services
  - UDDI Business Registry (UBR) project discontinued by Microsoft, IBM & SAP since 2006

REST

- **REpresentational State Transfer** (REST) is an architectural style for distributed systems
- It aims to capture the characteristics of the Web which made the Web successful
  - Resources in the Web are accessed using an URL, e.g., http://www.boeing.com/aircraft/747
  - The Web server returns a **representation**, e.g., a HTML document
  - This representation places the client in a **state**
  - The client may access another resource by selecting a hyperlink
  - This way, the client changes (**transfer**s) state with each resource representation
- REST was originally intended for information and media access, not RPC
- Many web services use REST (Yahoo: Flickr, del.icio.us; eBay and Amazon support REST and SOAP)

REST

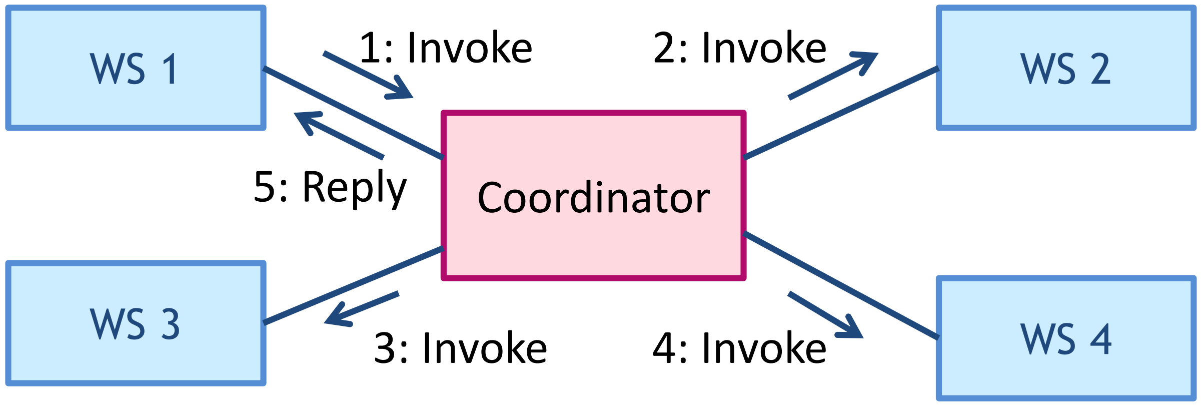
- Claim: scalability & growth of the Web is the direct result of a few key design principles:
  - Application state and functionality are abstracted into **resources**
  - Every resource is **uniquely addressable**
  - All resources share a **uniform interface** for transferring state between client and resource
    - A constrained set of **well-defined operations** (GET, PUT, POST, …)
    - A constrained set of **content types** (text/plain, text/html, image/png, ...)
  - A protocol which is
    - **Client-server**
    - **Stateless**
    - **Cacheable**
    - **Layered** (allow for intermediaries: proxies, gateways, firewalls, etc.)

REST

- REST-style APIs
  - REST APIs can be used from virtually any language, no libraries needed
  - XML generation/parsing in application code
    - … if XML is used – REST does not require that
  - Basically, this is IPC

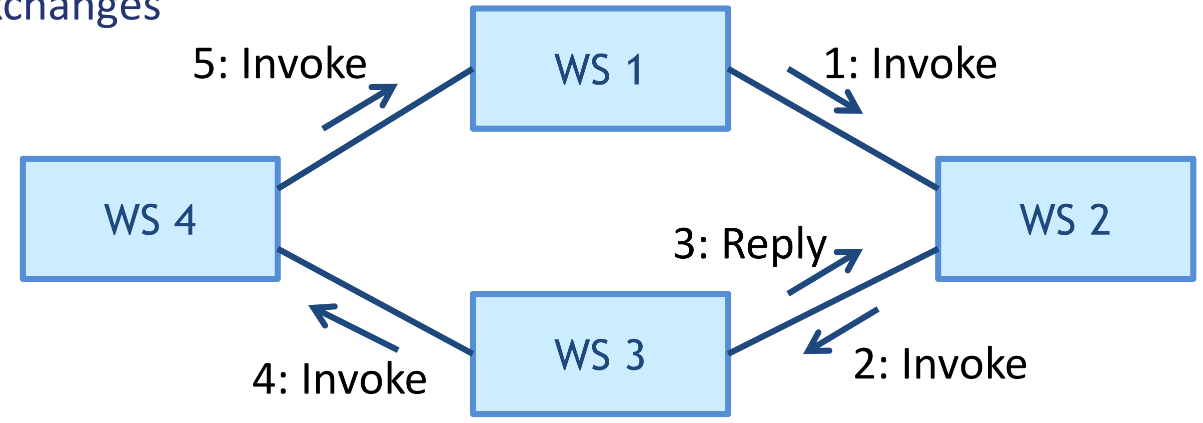
Service Composition

- Web services can be combined in two ways:
  - Orchestration
  - Choreography
- Orchestration
  - A central process coordinates the execution of different web services involved in the operation
  - Usually used in private business processes



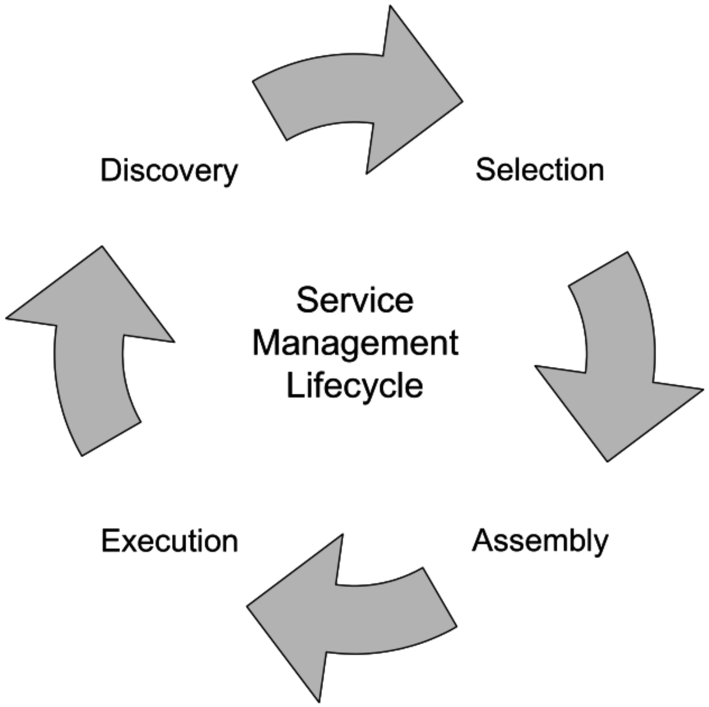
Service Composition

- Choreography
  - Each web service knows when to execute operations and with whom to interact („peer-to-peer“)
  - Usually used in public/inter-organization business processes
  - More difficult: Each participant must be aware of
    - Business process
    - Operations to execute
    - Messages to exchange
    - Timing of message exchanges



Service Management Lifecycle

- Only single services considered up to now
- Service Management Lifecycle
  - Discovery
    - Finding services
    - E.g., by UDDI
  - Selection
    - Selection of suitable services
    - E.g., does service suit a task in the workflow
  - Assembly
    - Interconnecting services
    - Composition in turn is a service
  - Execution
    - Execution of the composition



Web Services vs. RPC

- Both provide
  - Interface defintion
  - Libraries for boilerplate code

- **RPC**
  - Low overhead (at least with modern approaches, e.g., ProtoBuffers)
  - High concurrency (Futures etc.)
  - Custom RPC protocol

- **Web Services**
  - Interoperable
    - XML, HTTP(S), etc.
  - High(er) overhead

Summary: Mainstream Paradigms

- **IPC**
  - Sockets, Messages Queues
- **RPC**
  - Marshalling: CORBA, XML/SOAP, Java
  - RPC Failure Semantics
  - Asynchronous Calls
- **Web Services**
  - SOAP
  - REST