

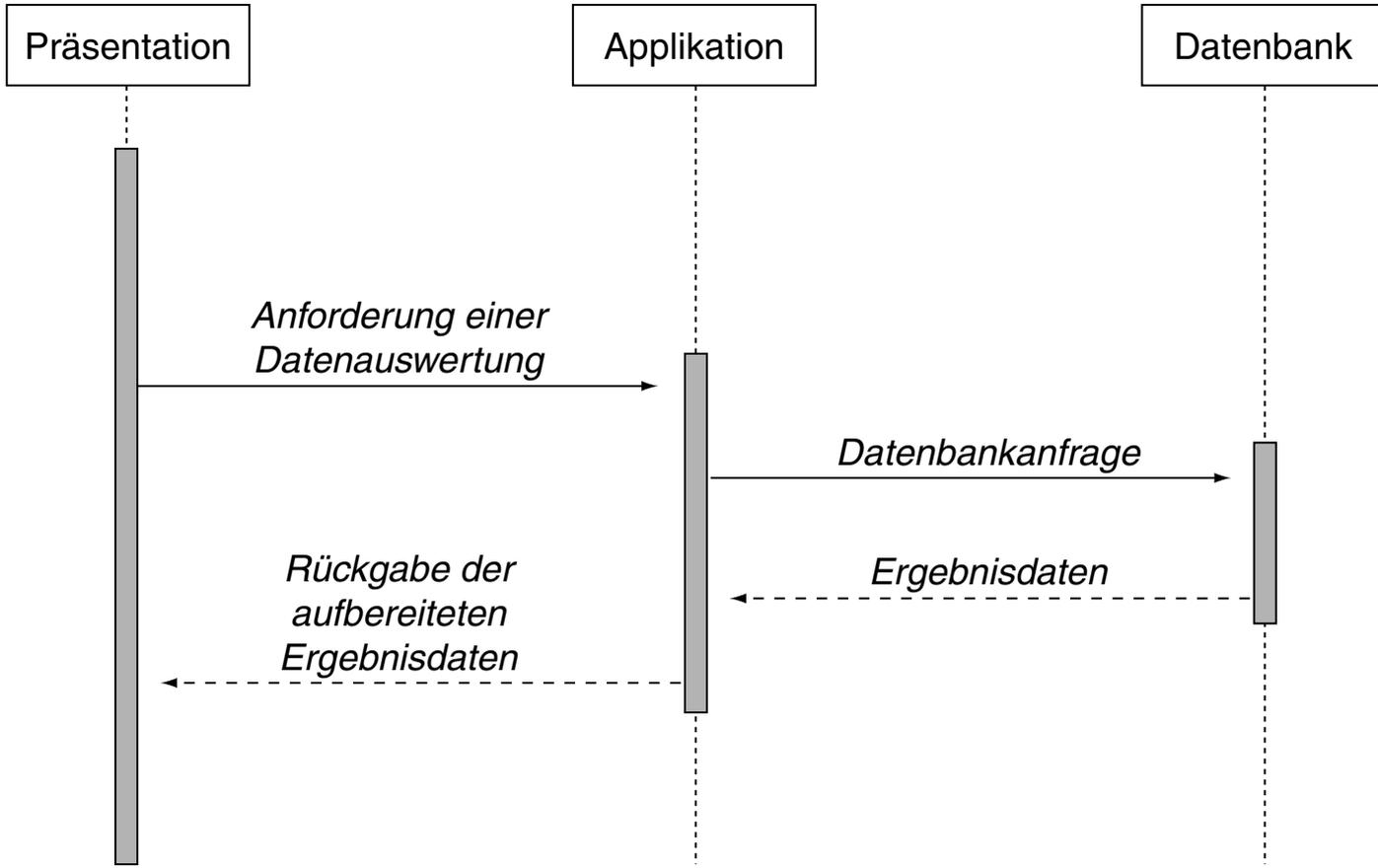
# Kapitel WT:VI

## VI. Architekturen und Middleware-Technologien

- ❑ Client-Server-Architekturen
- ❑ Ajax
- ❑ REST
- ❑ RPC, XML-RPC, Java RMI, DCOM
- ❑ **Web-Services**
- ❑ CORBA
- ❑ Message-oriented-Middleware MOM
- ❑ Enterprise Application Integration EAI

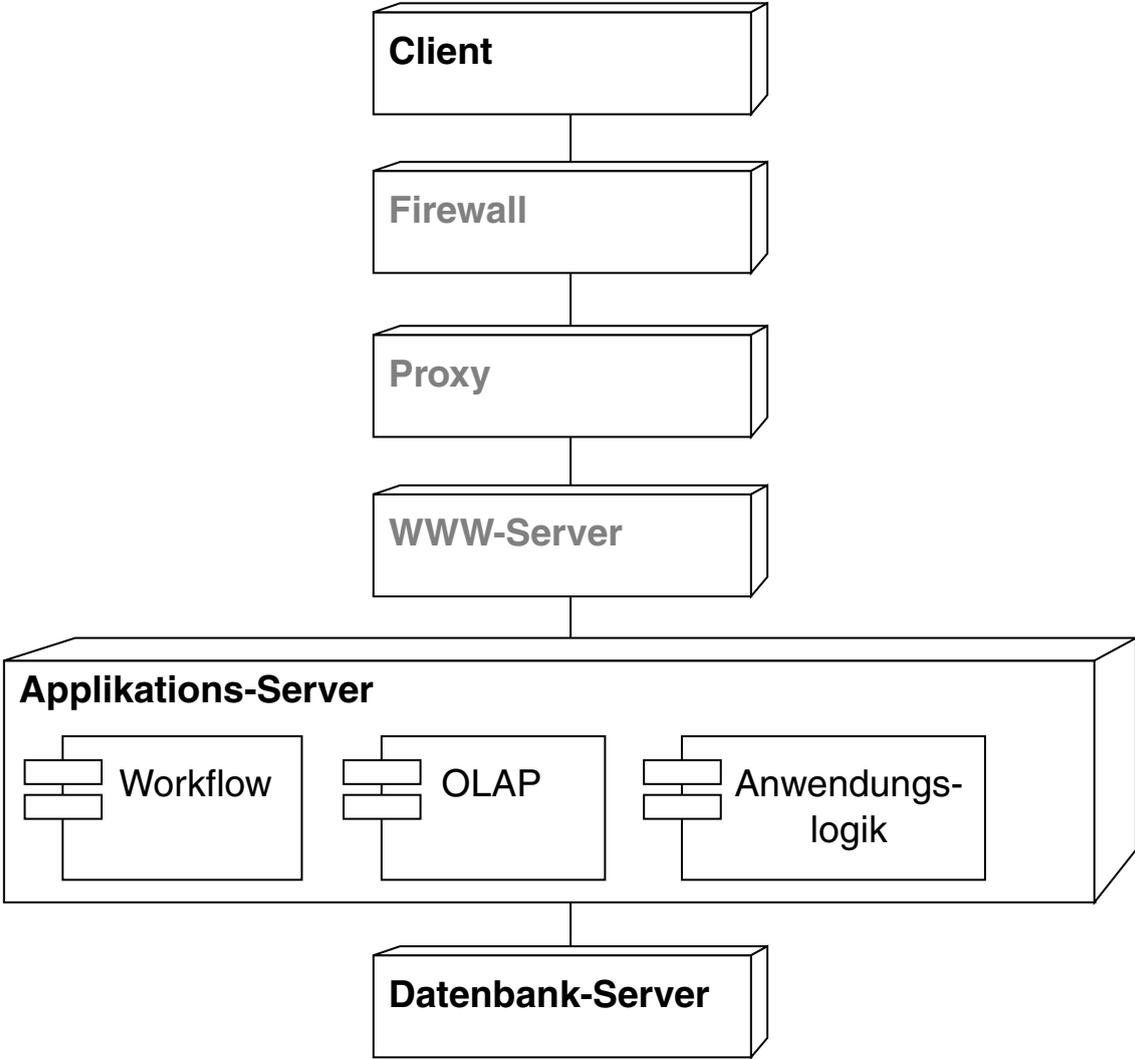
# Client-Server-Architekturen

## 3-Tier Architektur: Sequenzdiagramm



# Client-Server-Architekturen

## 3-Tier Architektur: Deployment-Diagramm



# Client-Server-Architekturen

Architekturmuster

Applikationslogik

Präsentationsschicht

Thin Client

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

HTML + JavaScript im  
Webbrowser

# Client-Server-Architekturen

## Architekturmuster

Thin Client

Portal

## Applikationslogik

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

im Portal-Applikationsserver

## Präsentationsschicht

HTML + JavaScript im  
Webbrowser

Portlets im Browser

# Client-Server-Architekturen

## Architekturmuster

Thin Client

Portal

Rich Thin Client

## Applikationslogik

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

im Portal-Applikationsserver

sowohl im Client als auch im  
Applikationsserver

## Präsentationsschicht

HTML + JavaScript im  
Webbrowser

Portlets im Browser

Java-AWT-GUI in speziellem  
Java Applet im Webbrowser

# Client-Server-Architekturen

## Architekturmuster

Thin Client

Portal

Rich Thin Client

Rich Fat Client

## Applikationslogik

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

im Portal-Applikationsserver

sowohl im Client als auch im  
Applikationsserver

clientseitiges Framework  
(plattformabhängig)

## Präsentationsschicht

HTML + JavaScript im  
Webbrowser

Portlets im Browser

Java-AWT-GUI in speziellem  
Java Applet im Webbrowser

z.B. Java-SWT, JFace-GUI

# Client-Server-Architekturen

## Architekturmuster

Thin Client

Portal

Rich Thin Client

Rich Fat Client

Managed Client

## Applikationslogik

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

im Portal-Applikationsserver

sowohl im Client als auch im  
Applikationsserver

clientseitiges Framework  
(plattformabhängig)

Client-Java-Middleware mit  
Client-Applikationsserver für  
Softwareverteilung, etc.

## Präsentationsschicht

HTML + JavaScript im  
Webbrowser

Portlets im Browser

Java-AWT-GUI in speziellem  
Java Applet im Webbrowser

z.B. Java-SWT, JFace-GUI

clientseitiges Framework

# Client-Server-Architekturen

## Architekturmuster

Thin Client

Portal

Rich Thin Client

Rich Fat Client

Managed Client

OS-dependent  
Fat Client

## Applikationslogik

im Applikationsserver, z.B. JSP  
+ JavaBeans mit Tomcat,  
J2EE-EnterpriseJavaBeans

im Portal-Applikationsserver

sowohl im Client als auch im  
Applikationsserver

clientseitiges Framework  
(plattformabhängig)

Client-Java-Middleware mit  
Client-Applikationsserver für  
Softwareverteilung, etc.

überwiegend im Client:  
Desktop-Applikation

## Präsentationsschicht

HTML + JavaScript im  
Webbrowser

Portlets im Browser

Java-AWT-GUI in speziellem  
Java Applet im Webbrowser

z.B. Java-SWT, JFace-GUI

clientseitiges Framework

Windows- oder Linux-GUI

[[torsten-horn.de](http://torsten-horn.de)]

# Client-Server-Architekturen

Architekturmuster

Applikationslogik

Präsentationsschicht

Thin Client

datenintensiv (*data shipping*)

Portal

Rich Thin Client

Rich Fat Client

Managed Client

OS-dependent  
Fat Client

auftragsbezogen (*operation shipping*)



# Client-Server-Architekturen

## Implementierung von Architekturmustern

- Ajax.
  - dynamisches Web für Thin Clients
  
- RPC, XML-RPC, DCOM, Java RMI.
  - Clients, die in der Lage sind, ein High-Level-Protokoll über TCP/IP abzuwickeln
  - Kommunikation zwischen Server-Anwendungen
  
- Web-Services.
  - flexible RPC-Technologie zur Client-Anbindung
  - **Komposition** komplexer Anwendungen **durch Service-Verschaltung** zwischen verschiedenen Servern
  
- CORBA.
  - industrielle, professionelle Technologie zur Kopplung von Server-Anwendungen
  - Anbindung von clientseitigen Frameworks
  
- Message-oriented-Middleware MOM.
  - Technologie zur asynchronen Kopplung von Server-Anwendungen

## Bemerkungen:

- ❑ Verschiedene der Technologien, mit denen sich ein bestimmtes Architekturmuster implementieren lässt, werden mit dem Begriff „Middleware“ in Verbindung gebracht. Middleware – auch als „Architectural Glue“ bezeichnet – realisiert die Infrastruktur für und zwischen Komponenten. Es gibt verschiedene Kategorien von Middleware, je nach Granularität und Art der Komponenten.
- ❑ Middleware ist Software, welche die Erstellung verteilter Anwendungen dadurch vereinfacht, dass sie standardisierte Mechanismen zur Kommunikation von verteilten Komponenten zur Verfügung stellt.

# Ajax

## Einführung

Ajax = **Asynchronous** JavaScript and XML

### Charakteristika:

- ❑ das synchrone Request-Response-Paradigma wird aufgebrochen
- ❑ Web-Seiten müssen nicht als Ganzes ersetzt, sondern können teilweise überladen werden. Schnittstelle: DOM-API
- ❑ auf klar definierten, offenen Standards basierend
- ❑ Browser- und plattformunabhängig
- ❑ es wird keine Art von „Ajax-Server“ benötigt, sondern auf bekannten Web-Server-Technologien aufgesetzt

### Anwendung:

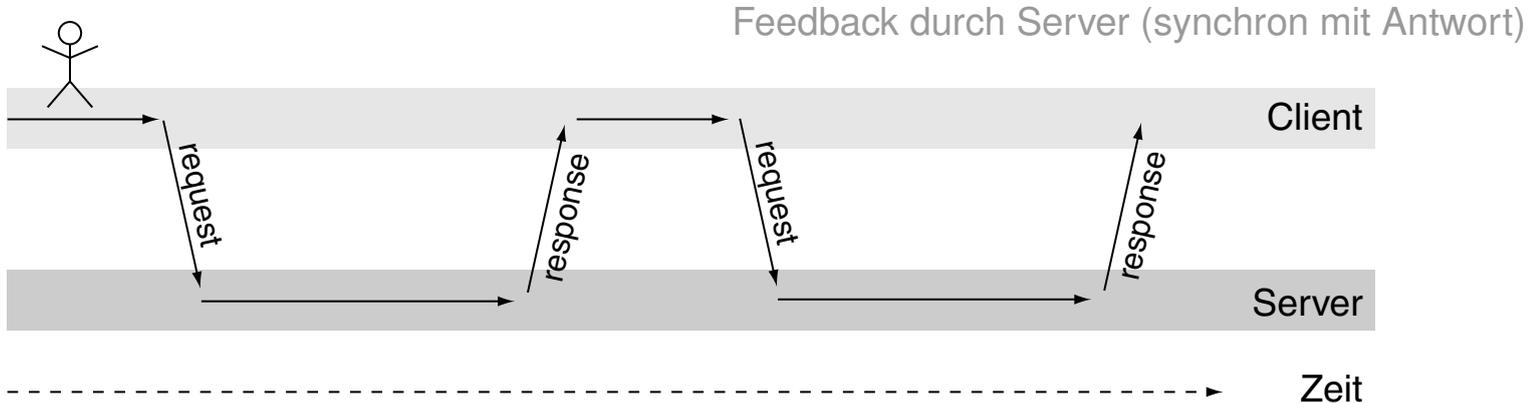
- ❑ Realisierung interaktiver Thin Clients
- ❑ standardisierte, einfache graphische Bedienelemente

## Bemerkungen:

- ❑ Die Kernidee von Ajax besteht darin, einen HTTP-Request nebenläufig auszuführen und das Ergebnis des Requests in den DOM-Seitenbaum des Browsers einzufügen.
- ❑ Teilweise wird Ajax als reine Client-Technologie bezeichnet. [[apache.org](http://apache.org)]
- ❑ Tatsächlich steht bei Ajax die Art und die Abwicklung der Kommunikation zwischen Client und Server im Vordergrund. Somit kann man Ajax als eine Technologie zur Umsetzung eines Architekturmusters verstehen: “Ajax isn’t a technology, it’s more of a pattern – a way to identify and describe a useful design technique.” [McCarthy 2005, [IBM](#)]

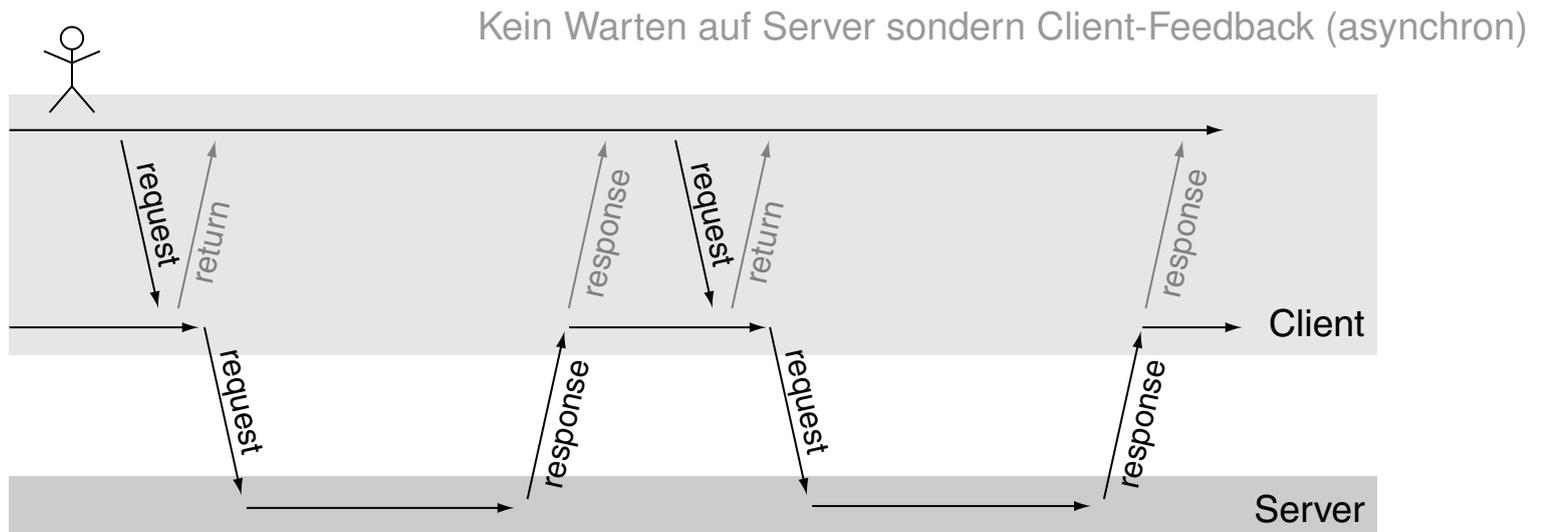
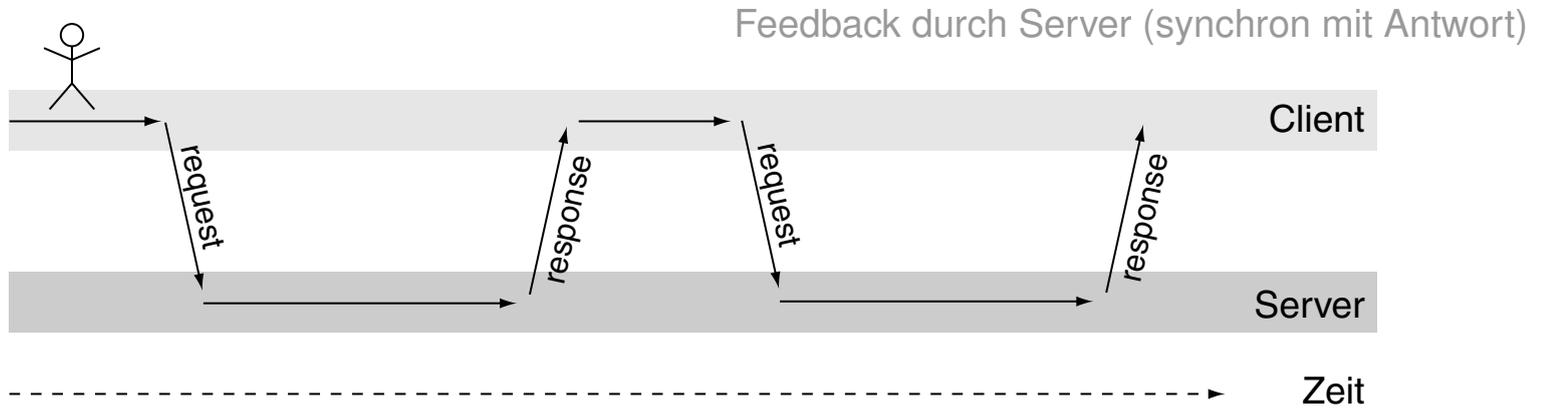
# Ajax [Garret 2005]

## Einführung



# Ajax [Garret 2005]

## Einführung



# Ajax

## Bestandteile einer Ajax-Anwendung

1. Event-Handler.
2. Callback-Funktion.
3. Server-Funktion.

# Ajax

## Bestandteile einer Ajax-Anwendung

### 1. Event-Handler.

- ❑ realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
- ❑ wird bei entsprechender Anwenderaktion aufgerufen
- ❑ instanziiert (bei jedem Aufruf) ein `XMLHttpRequest`-Objekt
- ❑ meldet eine Callback-Funktion im `XMLHttpRequest`-Objekt an
- ❑ ruft die zur Anwenderaktion gehörende Server-Funktion auf

### 2. Callback-Funktion.

### 3. Server-Funktion.

# Ajax

## Bestandteile einer Ajax-Anwendung

### 1. Event-Handler.

- ❑ realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
- ❑ wird bei entsprechender Anwenderaktion aufgerufen
- ❑ instanziert (bei jedem Aufruf) ein `XMLHttpRequest`-Objekt
- ❑ meldet eine **Callback-Funktion** im `XMLHttpRequest`-Objekt an
- ❑ ruft die zur Anwenderaktion gehörende Server-Funktion auf

### 2. Callback-Funktion.

- ❑ realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
- ❑ wird via `readyState`-Events vom `XMLHttpRequest`-Objekt aufgerufen
- ❑ filtert bezüglich `readyState` 4 und HTTP-O.K.-Status-Code 200
- ❑ verarbeitet die zurückgegebene XML-Datei der Server-Funktion oder ruft für die Verarbeitung eine weitere JavaScript-Funktion auf und manipuliert den DOM

### 3. Server-Funktion.

# Ajax

## Bestandteile einer Ajax-Anwendung

### 1. Event-Handler.

- ❑ realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
- ❑ wird bei entsprechender Anwenderaktion aufgerufen
- ❑ instanziiert (bei jedem Aufruf) ein `XMLHttpRequest`-Objekt
- ❑ meldet eine **Callback-Funktion** im `XMLHttpRequest`-Objekt an
- ❑ ruft die zur Anwenderaktion gehörende **Server-Funktion** auf

### 2. Callback-Funktion.

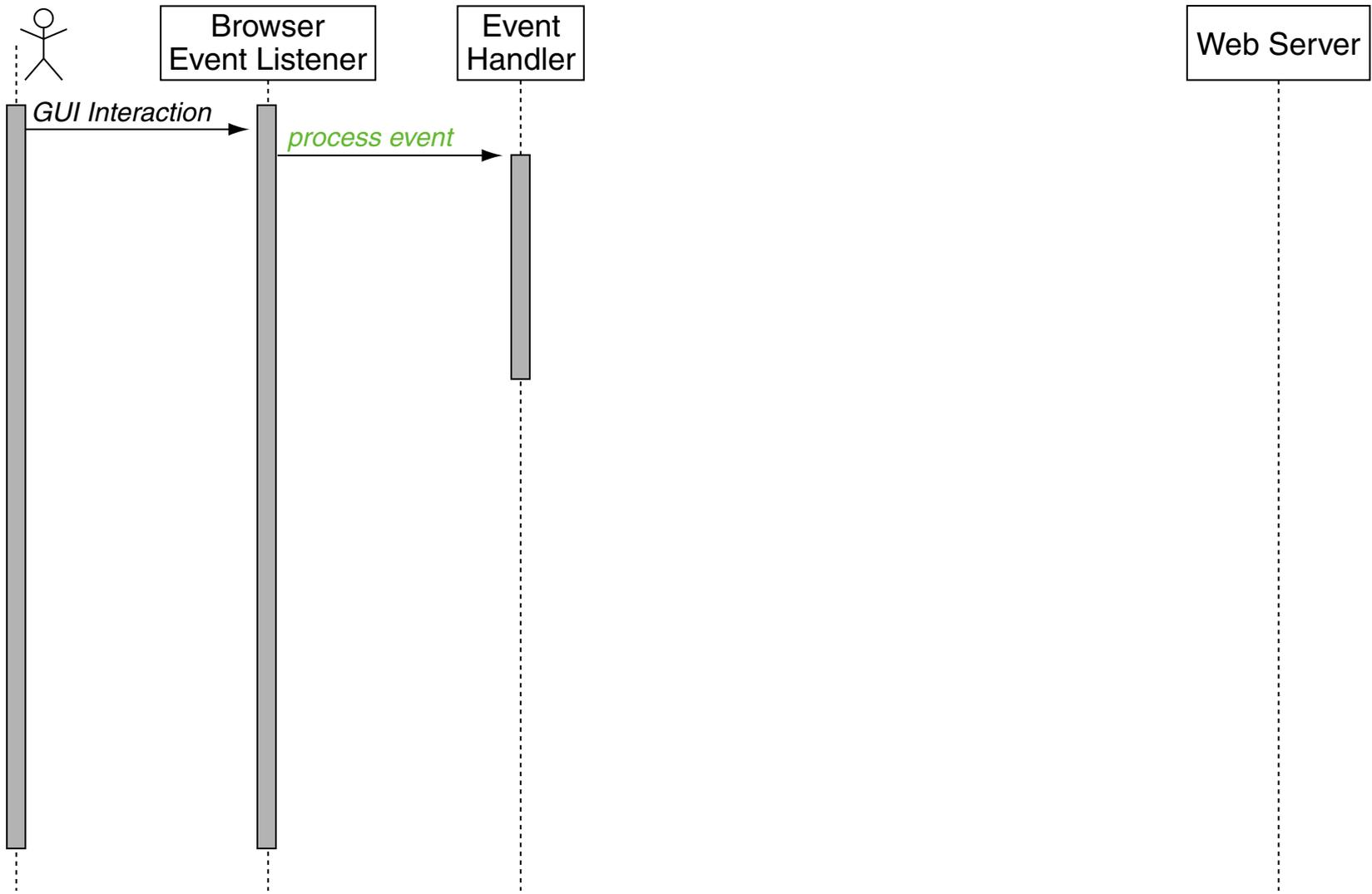
- ❑ realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
- ❑ wird via `readyState`-Events vom `XMLHttpRequest`-Objekt aufgerufen
- ❑ filtert bezüglich `readyState` 4 und HTTP-O.K.-Status-Code 200
- ❑ verarbeitet die zurückgegebene XML-Datei der Server-Funktion oder ruft für die Verarbeitung eine weitere JavaScript-Funktion auf und manipuliert den DOM

### 3. Server-Funktion.

- ❑ wird mittels Standardtechnologie (CGI, PHP-Script, Servlet, etc.) auf einem Web-Server zur Verfügung gestellt
- ❑ generiert eine XML-Datei als Rückgabewert

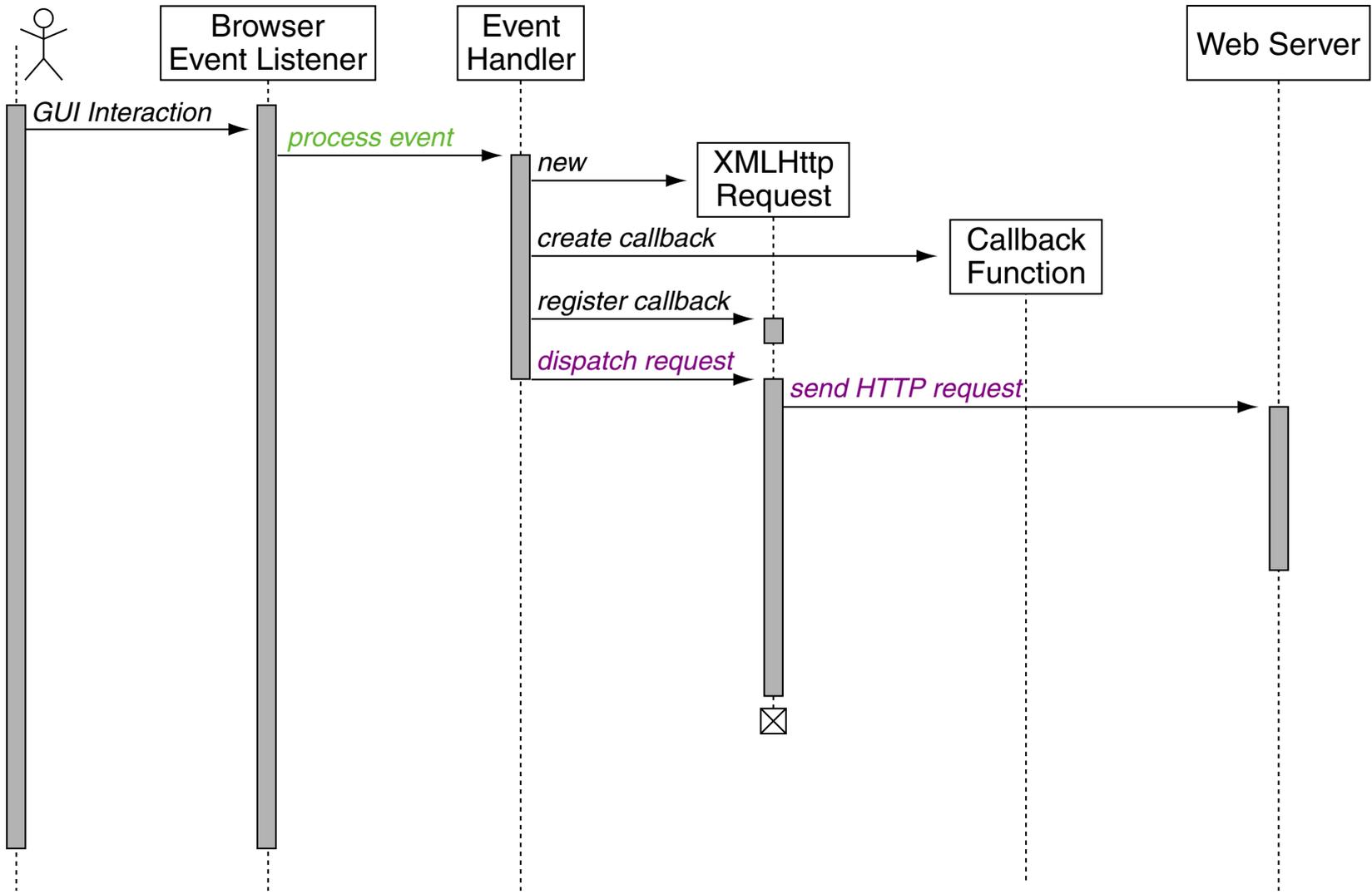
# Ajax

## Ablauf einer Ajax-Interaktion



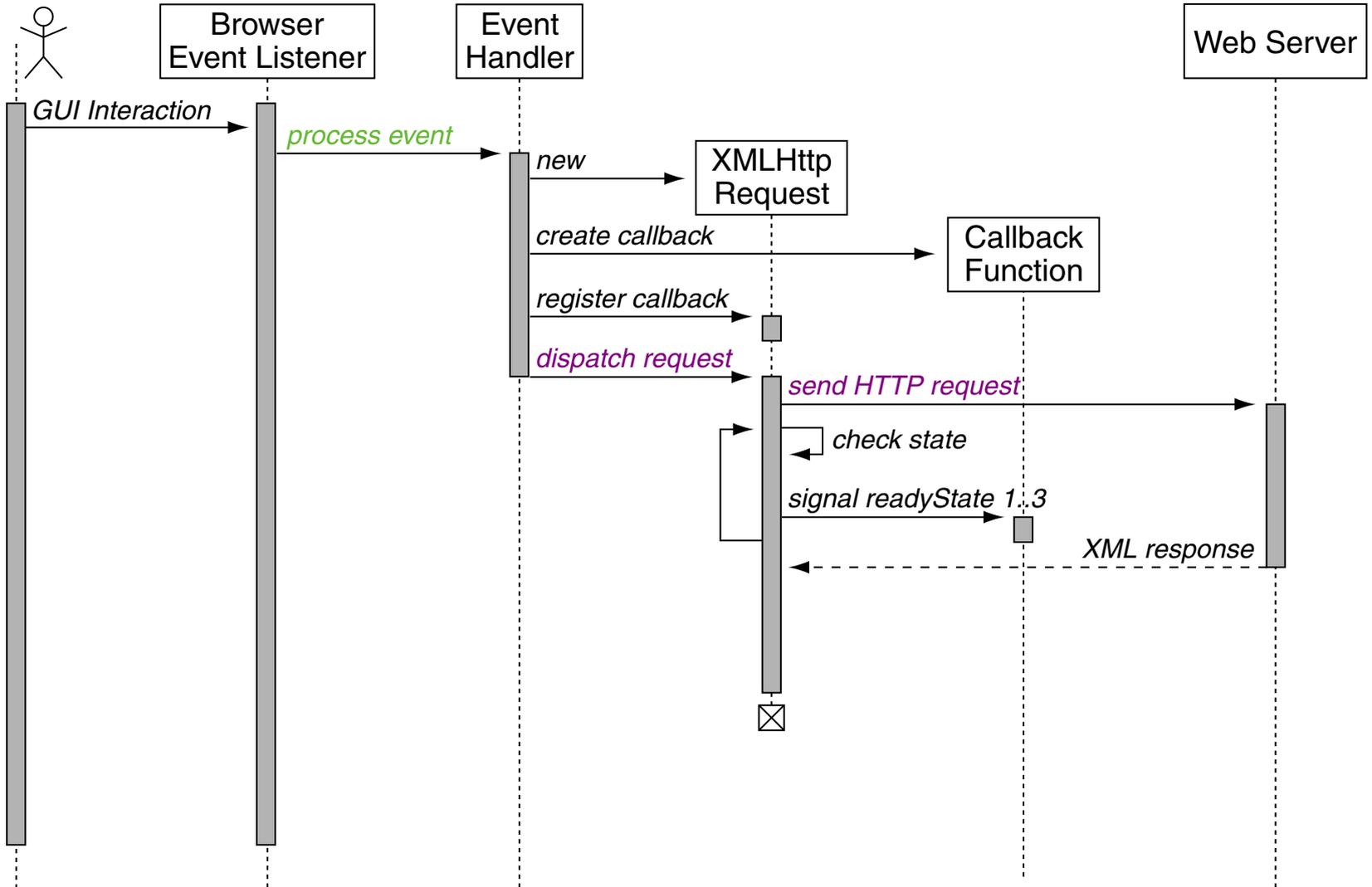
# Ajax

## Ablauf einer Ajax-Interaktion (Fortsetzung)



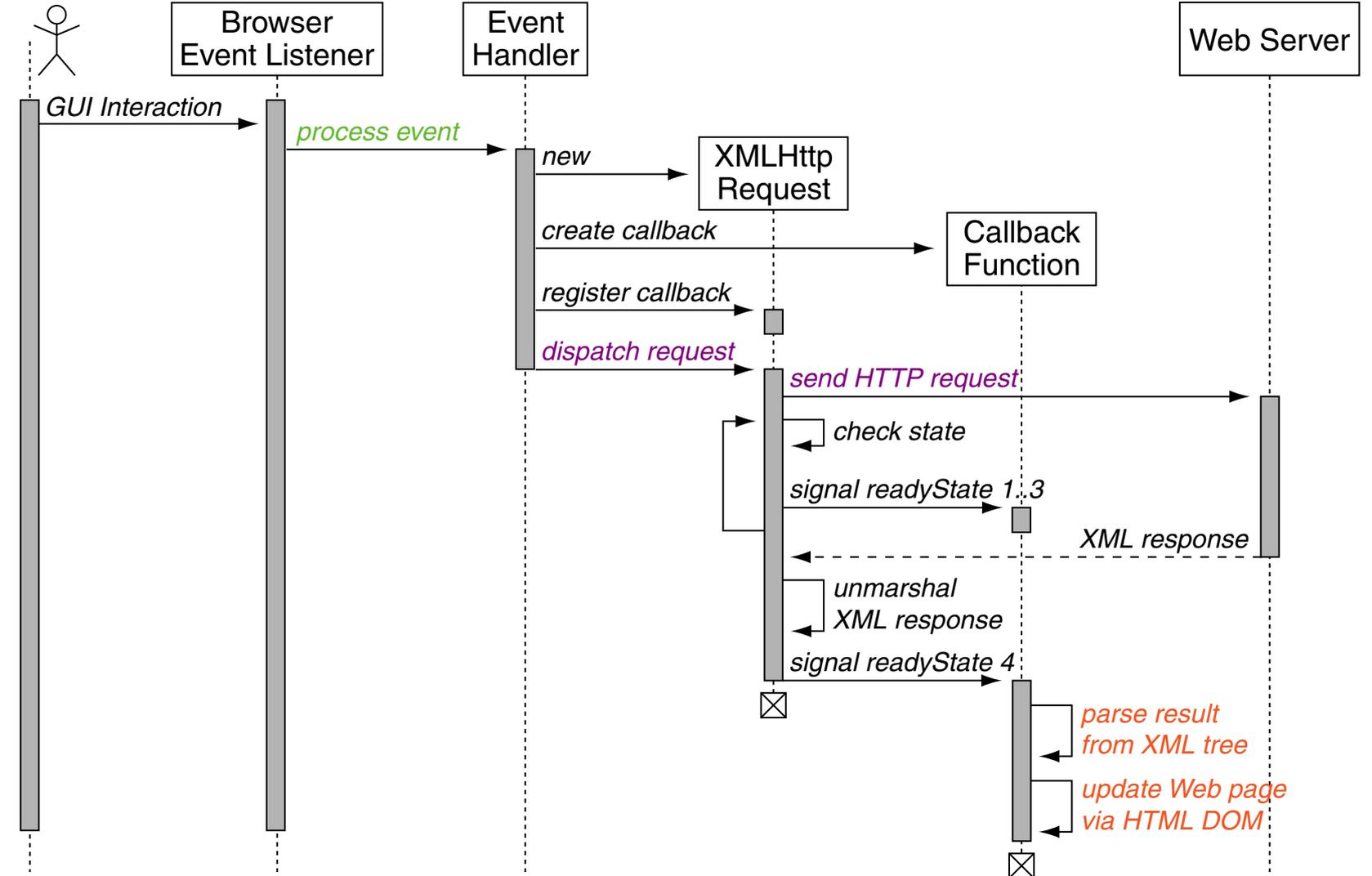
# Ajax

## Ablauf einer Ajax-Interaktion (Fortsetzung)



# Ajax

## Ablauf einer Ajax-Interaktion (Fortsetzung)



# Ajax

## Beispiel: Überwachung von Eingabefeld

Ajax Example - Mozilla Firefox

Preferred user name:

Ajax Example - Mozilla Firefox

Preferred user name:

[AJAX-Ausführung]

# Ajax

## Beispiel: Überwachung von Eingabefeld



Preferred user name:  
Ali ■



Preferred user name:  
Alice ■

[AJAX-Ausführung]

HTML-Code:

```
...  
<form action="">  
  Preferred user name:  
  <input name="username" type="text"  
    onkeyup="myEventHandler(  
      './check-username.php?q=' + this.value,  
      processUsernameResponse  
    )" />  
</form>
```



# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

### 1. Generischer JavaScript-Code für Event-Handler:

```
function myEventHandler(url, processResponseXML) {  
    var req = new XMLHttpRequest();  
    if(req) {  
        req.onreadystatechange = // Create and register callback.  
            createCallbackFunction(req, processResponseXML);  
        req.open("GET", url, true); // Dispatch request.  
        req.send(null); // Send HTTP request.  
    }  
}
```

# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

2a. Generischer JavaScript-Code der Callback-Funktion:

```
function createCallbackFunction(req, processResponseXML) {  
  return function() {  
    // Check whether readyState is complete.  
    if(req.readyState == 4) {  
      // Check whether server response is O.K.  
      if(req.status == 200) {  
        processResponseXML(req.responseXML);  
      } else { alert("HTTP error: " + req.status); }  
    }  
  }  
}
```

# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

### 2a. Generischer JavaScript-Code der Callback-Funktion:

```
function createCallbackFunction(req, processResponseXML) {
  return function() {
    // Check whether readyState is complete.
    if(req.readyState == 4) {
      // Check whether server response is O.K.
      if(req.status == 200) {
        processResponseXML(req.responseXML);
      } else { alert("HTTP error: " + req.status); }
    }
  }
}
```

### 2b. Spezifischer JavaScript-Code der Callback-Funktion zur Verarbeitung der XML-Antwortdatei:

```
function processUsernameResponse(xmltree) {
  var result = xmltree.documentElement.getElementsByTagName(
    'result')[0].firstChild.data;
  var message = document.getElementById('nameCheck');
  if(result == 1) {
    message.className = 'unavailable';
  } else{ message.className = 'available'; }
}
```

## Bemerkungen:

- ❑ Das Beispiel beschreibt ein wiederverwendbares Pattern für Ajax-Anwendungen: der generische JavaScript-Code kann unverändert in jedem HTML-Dokument zum Einsatz kommen. Zur Verarbeitung der XML-Antwort ist eine spezifische JavaScript-Funktion zu definieren. Diese Funktion zusammen mit der Server-Funktion bilden die Argumente von `myEventHandler()`, der beliebigen Events im HTML-Dokument zugeordnet werden kann.
- ❑ Die Art der Verwendung von `createCallbackFunction` hat Ähnlichkeit zum Template-Method-Pattern.
- ❑ Der Rumpf von `createCallbackFunction` definiert eine anonyme Callback-Funktion in Form eines Funktionsliterals und gibt diese Funktionsdefinition als Wert zurück.
- ❑ Das zweite Argument von `createCallbackFunction`, `processResponseXML`, enthält als Wert die Funktionsdefinition für `processUsernameResponse` mit dem JavaScript-Code zur Verarbeitung der XML-Antwortdatei.

# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [Ajax-Ausführung: [1](#), [2](#)]:

```
<?php
```

```
?>
```

```
<response>
```

```
  <result> <?php
```

```
?> </result>
```

```
</response>
```

# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [Ajax-Ausführung: [1](#), [2](#)]:

```
<?php
header('Content-Type: text/xml');
echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';
```

```
?>
<response>
  <result> <?php                                ?> </result>
</response>
```

# Ajax

## Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [Ajax-Ausführung: [1](#), [2](#)]:

```
<?php
header('Content-Type: text/xml');
echo '<?xml version="1.0" encoding="UTF-8" standalone="yes"?>';

function nameInUse($q) {
    if (isset($q)){
        switch(strtolower($q)) {
            case 'alice' :
                return '1';
                break;
            case 'bob' :
                return '1';
            ...
            default:
                return '0';
        }
    }else{ return '0'; }
}
?>

<response>
    <result> <?php echo nameInUse($_GET['q']) ?> </result>
</response>
```

# Ajax

## Quellen zum Nachlernen und Nachschlagen im Web

- ❑ Google. *Google Web Toolkit GWT*.  
[code.google.com/p/webtoolkit](http://code.google.com/p/webtoolkit)
- ❑ Horn. *Technische Kurzdokumentationen*.  
[www.torsten-horn.de/techdocs](http://www.torsten-horn.de/techdocs)
- ❑ McCarthy. *Ajax for Java developers: Build dynamic Java applications*.  
[www.ibm.com/developerworks/java/library/j-ajax1](http://www.ibm.com/developerworks/java/library/j-ajax1)
- ❑ McLellan. *Very Dynamic Web Interfaces*.  
[www.xml.com/pub/a/2005/02/09/xml-http-request.html](http://www.xml.com/pub/a/2005/02/09/xml-http-request.html)
- ❑ W3 Schools. *Ajax Tutorial*.  
[www.w3schools.com/ajax](http://www.w3schools.com/ajax)

## VI. Architekturen und Middleware-Technologien

- ❑ Client-Server-Architekturen
- ❑ Ajax
- ❑ REST
- ❑ RPC, XML-RPC, Java RMI, DCOM
- ❑ **Web-Services**
- ❑ CORBA
- ❑ Message-oriented-Middleware MOM
- ❑ Enterprise Application Integration EAI

# Web-Services

Web-Services ermöglichen die Abwicklung von Dienstleistungen und Geschäften über das Internet.

# Web-Services

Web-Services ermöglichen die Abwicklung von Dienstleistungen und Geschäften über das Internet.

WSFL

Service-Choreographie

UDDI

Service-Entdeckung

Service-Veröffentlichung

WSDL

Service-Beschreibung

SOAP

XML-basierter Nachrichtenaustausch

HTTP, SMTP, ...

Netzwerk

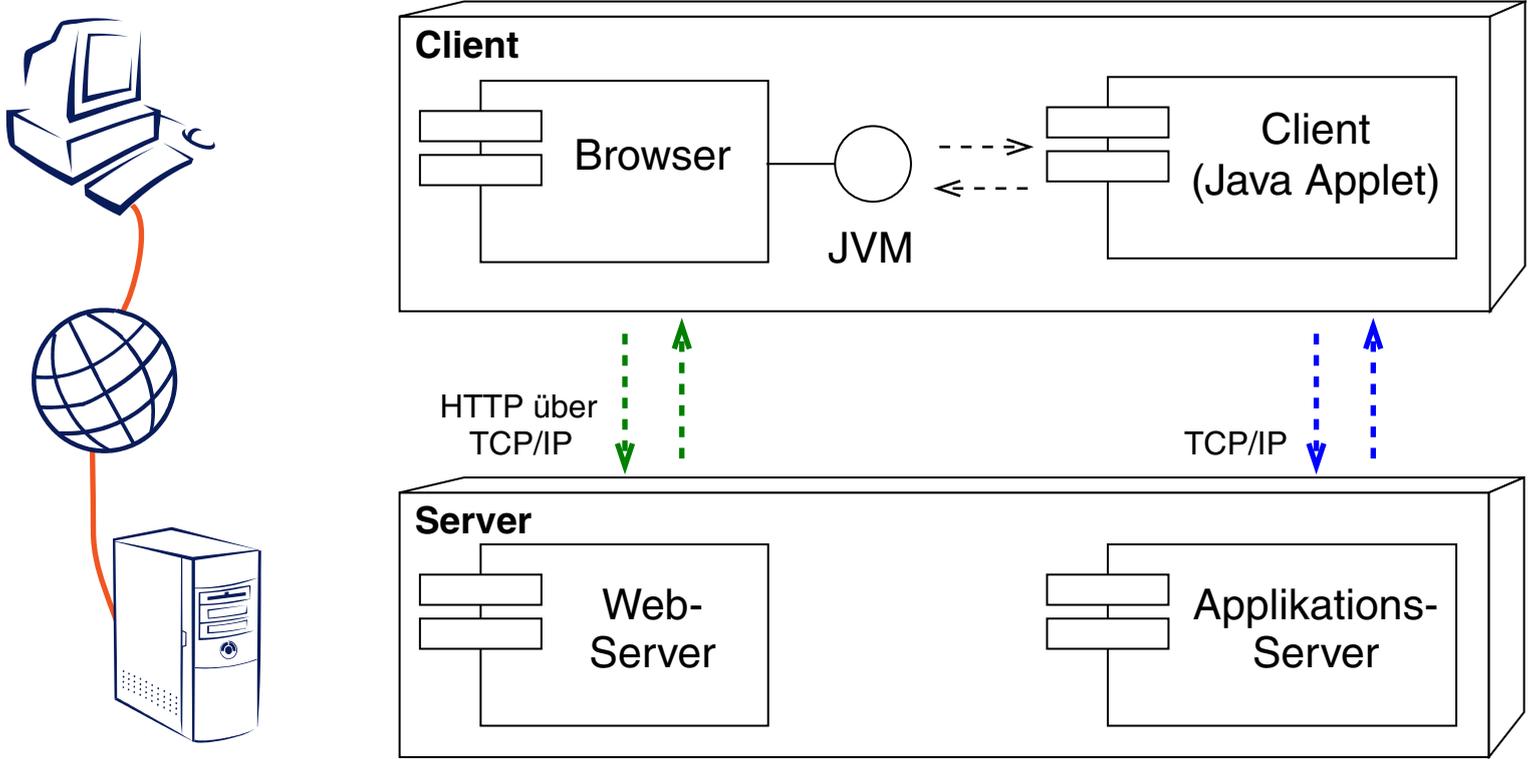
Web-Service-Definition aus Protokollsicht [Kilgore 2002]:

Web-Service = HTTP + XML + SOAP + WSDL (+ UDDI + WSFL)

# Web-Services

## Rich Client Anbindung

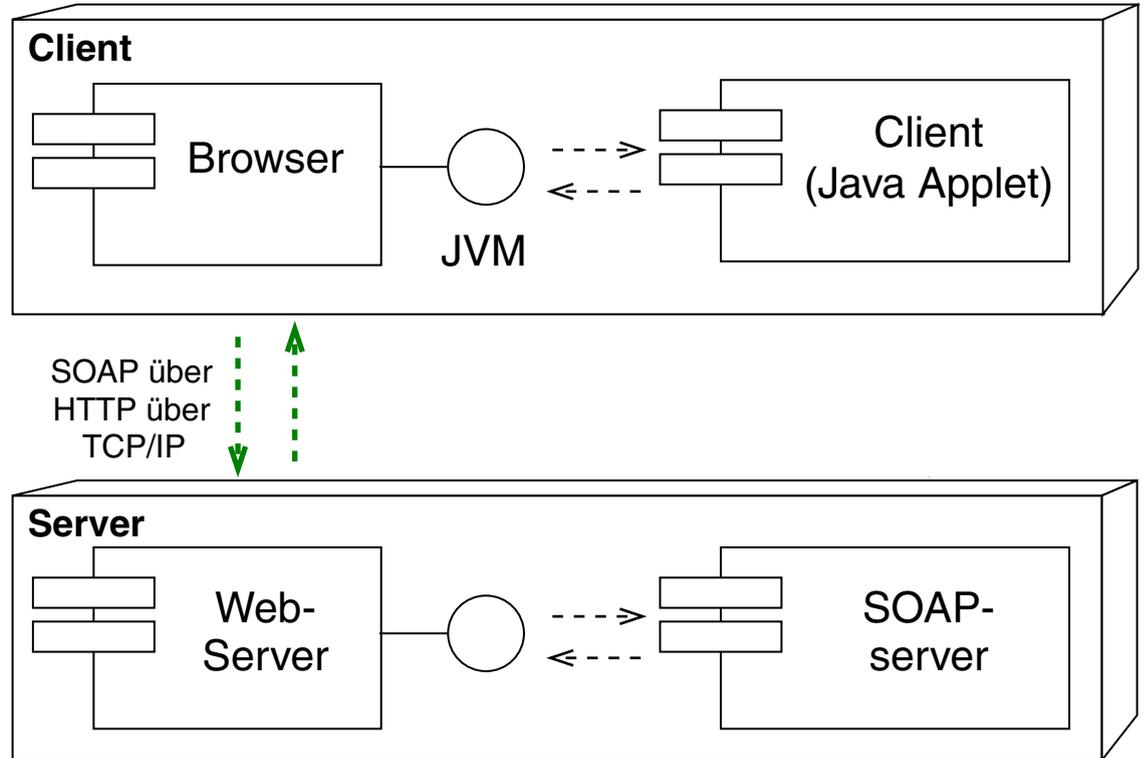
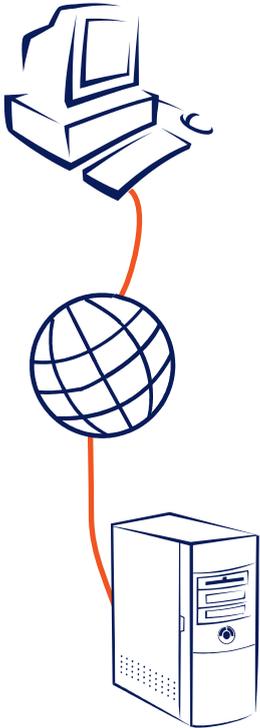
Verwendung von eigener TCP/IP-Verbindung und proprietärem Protokoll:



# Web-Services

## Rich Client Anbindung (Fortsetzung)

Verwendung von SOAP via Standard-HTTP-Protokoll zum Web-Server:

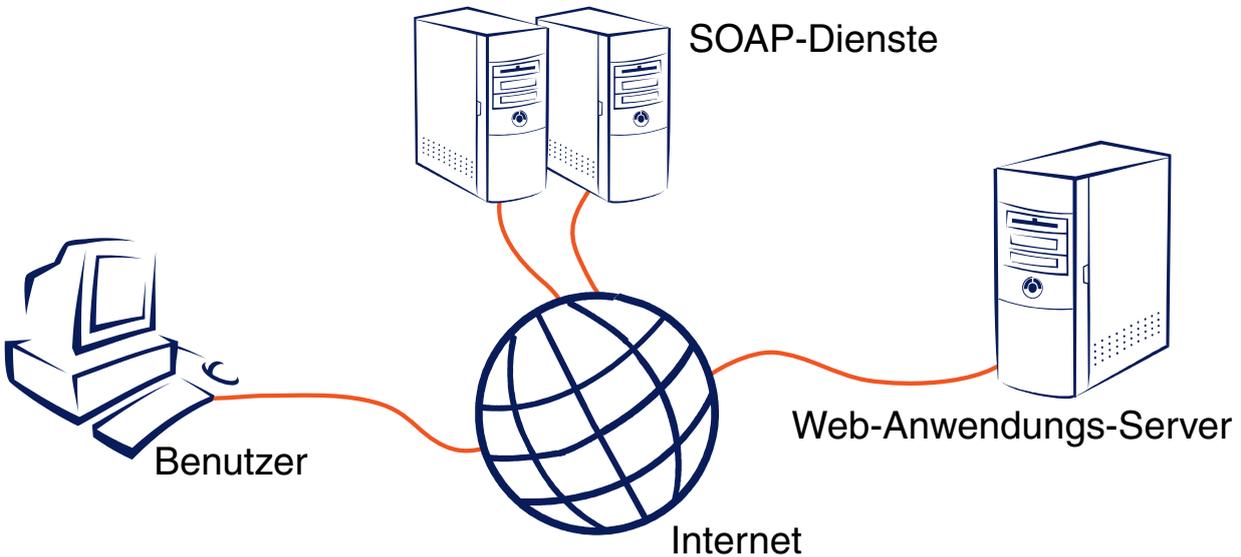


## Bemerkungen:

- ❑ Wenn man Web-Services entwickelt, muss man von der klassischen Client-Server-Sicht umdenken; insbesondere muss man sich bei SOAP zwei Server vorstellen. [[Lohrer 2003](#)]

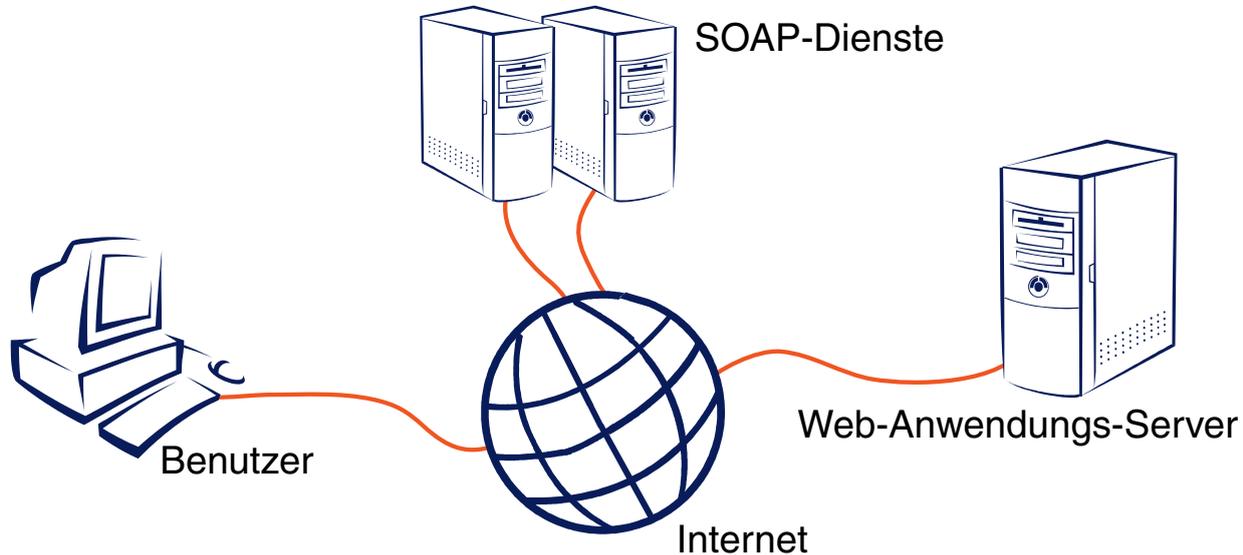
# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument



# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument



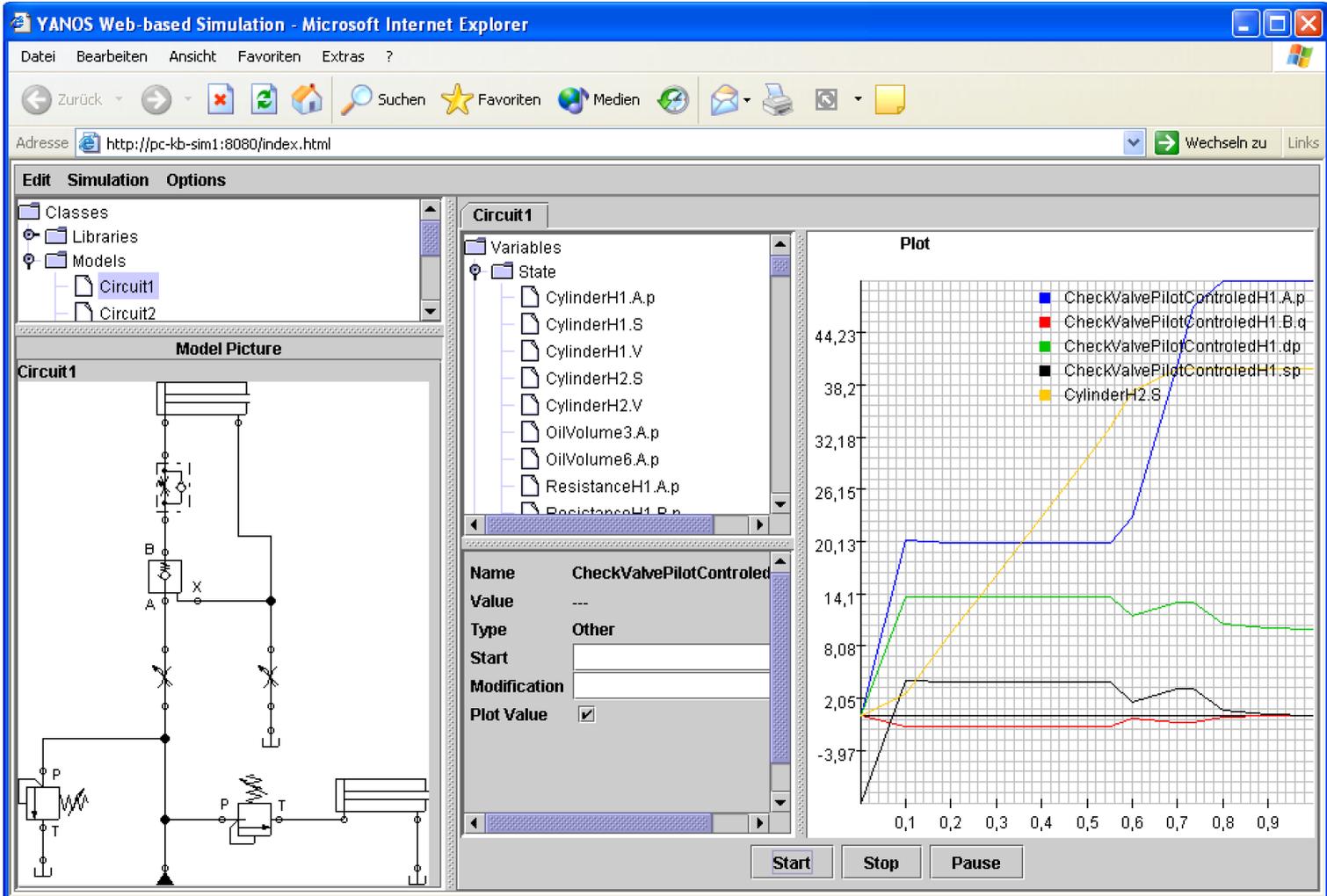
### Der Web-Anwendungs-Server

- ❑ bereitet mittels SOAP-Dienst A das Modell zur Simulation auf,
- ❑ lässt über SOAP-Dienst B die Simulation durchführen,
- ❑ lässt über SOAP-Dienst C die Simulationsergebnisse aufbereiten,

und präsentiert das Ergebnis dem Anwender.

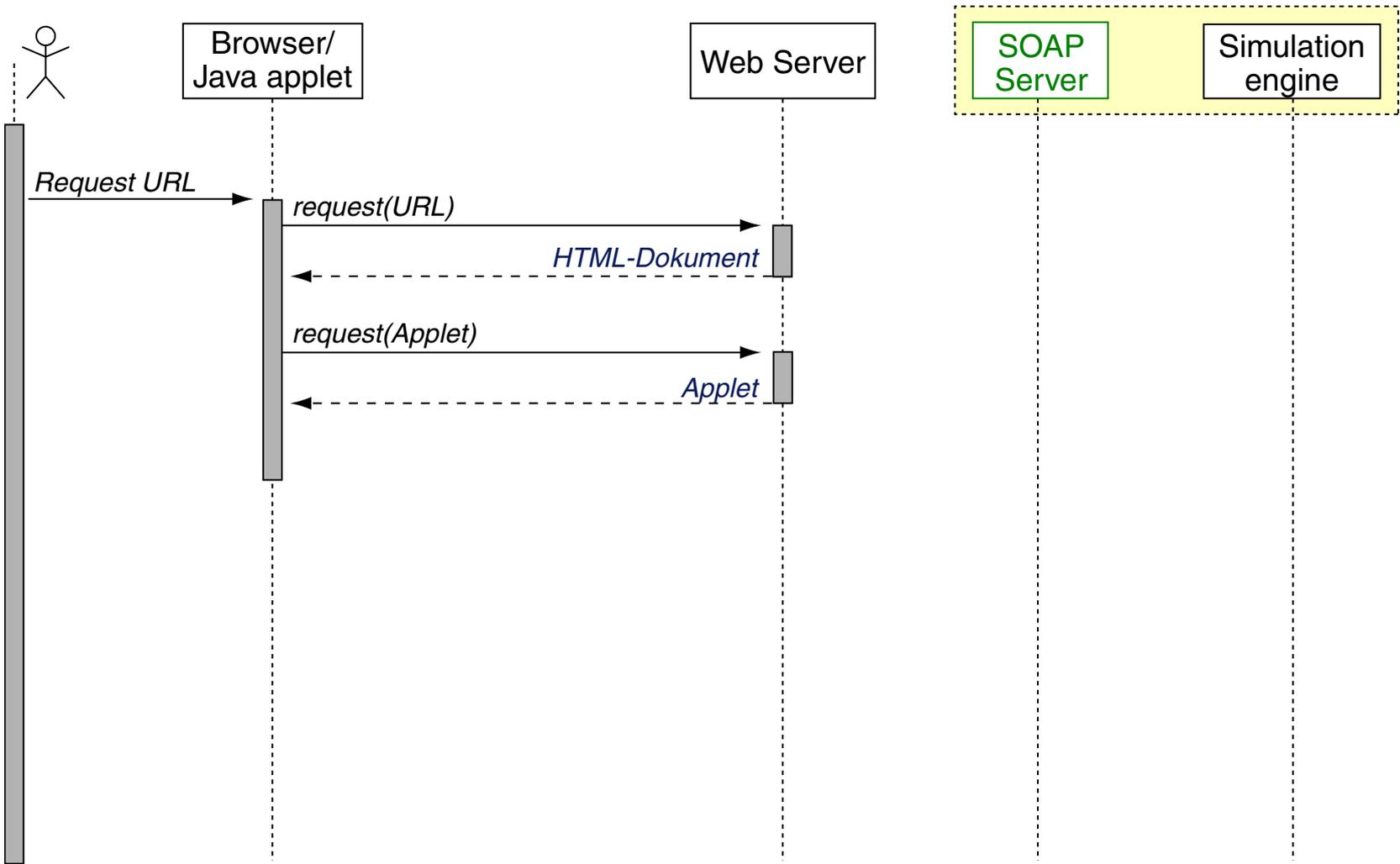
# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument (Fortsetzung)



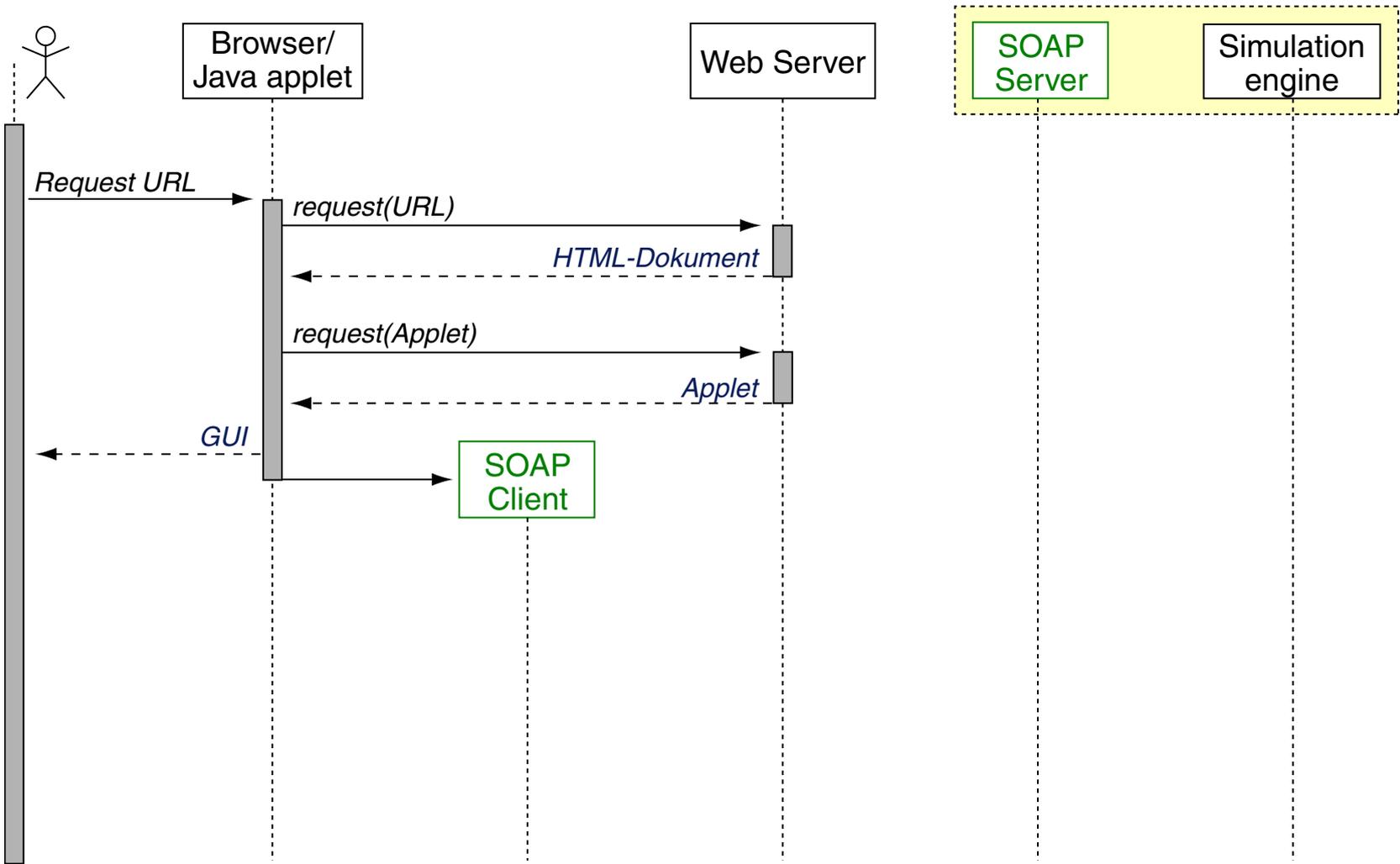
# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument (Fortsetzung)



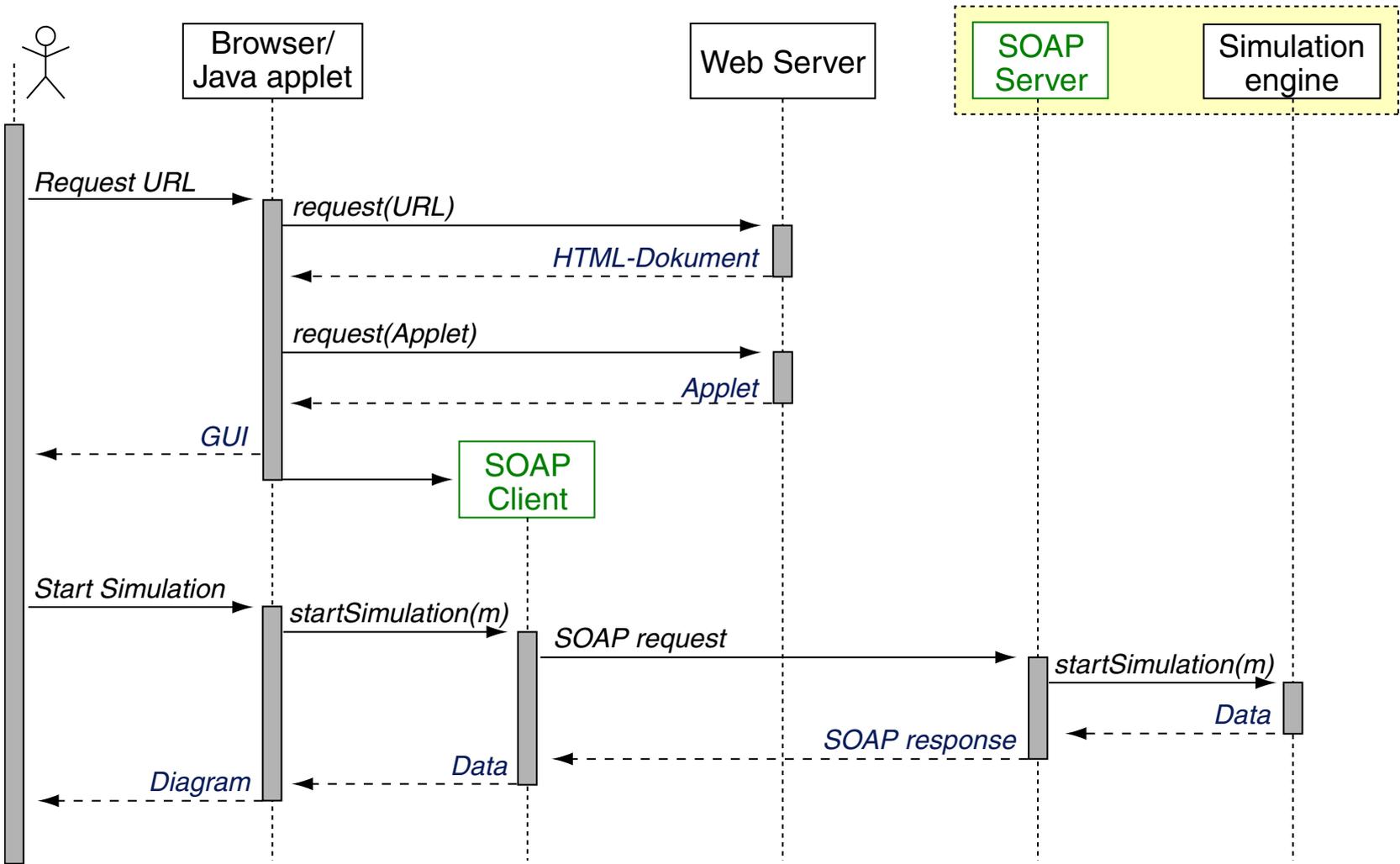
# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument (Fortsetzung)



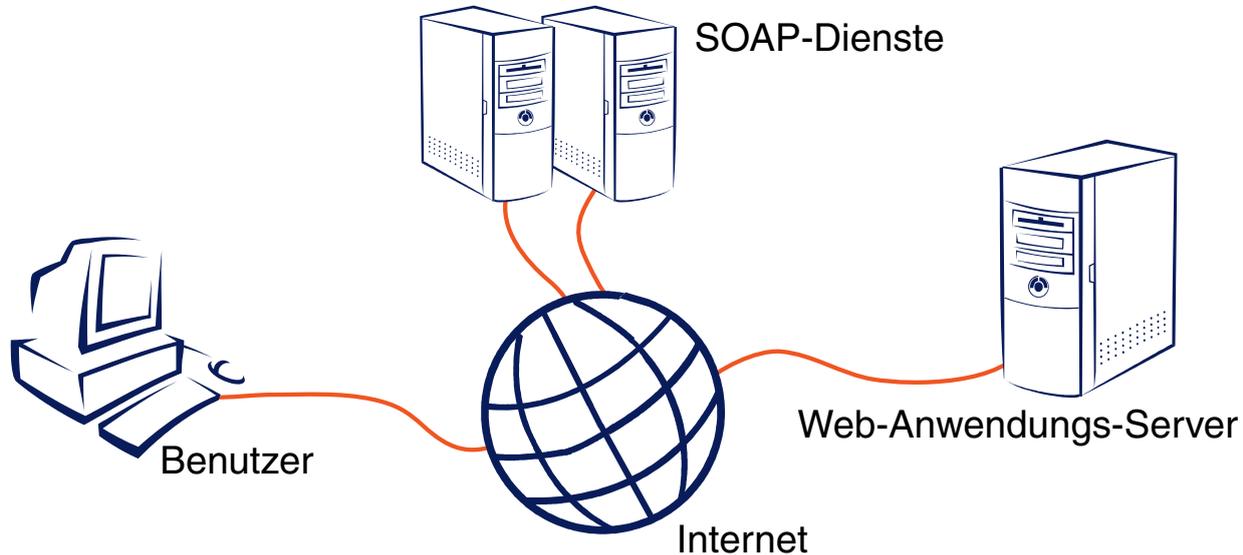
# Web-Services

## Szenario 1: Simulation integriert im CAD-Dokument (Fortsetzung)



# Web-Services

## Szenario 2: Übersetzung eines Textdokuments



### Der Web-Anwendungs-Server

- ❑ überprüft über einen SOAP-Dienst X die Kreditkartennummer,
- ❑ erfragt beim SOAP-Dienst Y aktuelle Währungsumrechnungskurse,
- ❑ lässt über SOAP-Dienst A den ASCII-Text extrahieren,
- ❑ fordert über SOAP-Dienst B eine Übersetzung des Textes an,
- ❑ lässt über SOAP-Dienst C ein neues Dokument erstellen,

und präsentiert das Ergebnis dem Anwender.

# Web-Services

## Automatisierungsaspekte

- ❑ SOAP ist ein **Mechanismus für entfernte Funktionsaufrufe**, codiert in XML.
- ❑ SOAP ist unabhängig vom Transportprotokoll, meistbenutzt ist HTTP.

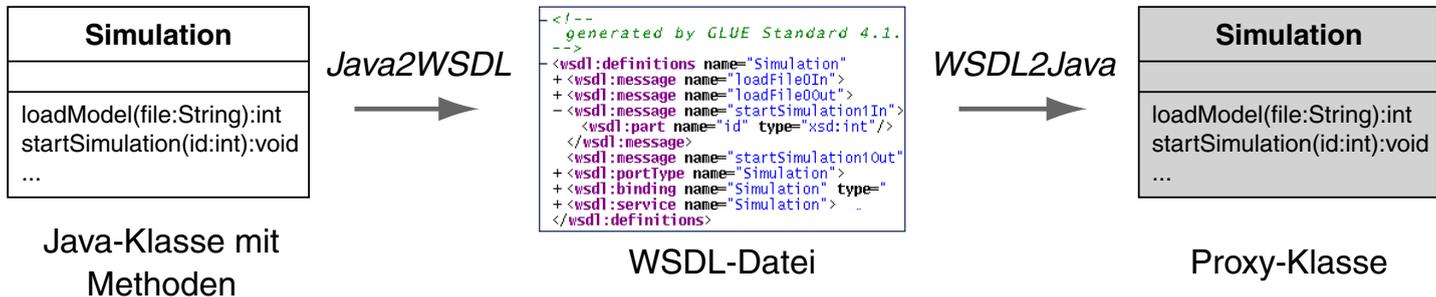
Vergleich mit anderen High-Level-Protokollen:

	RPC	RMI	DCOM	proprietares TCP/IP	CORBA	SOAP
plattformunabhängig		+		+	+	+
herstellerunabhängig	0				+	+
sprachunabhängig	0		+	+	+	+
Browser-integrierbar		+		+	+	+
Protokollgenerierung			0		0	+
öffentliche Interfaces					0	+
Firewall-verträglich						+

# Web-Services

## Automatisierungsaspekte (Fortsetzung)

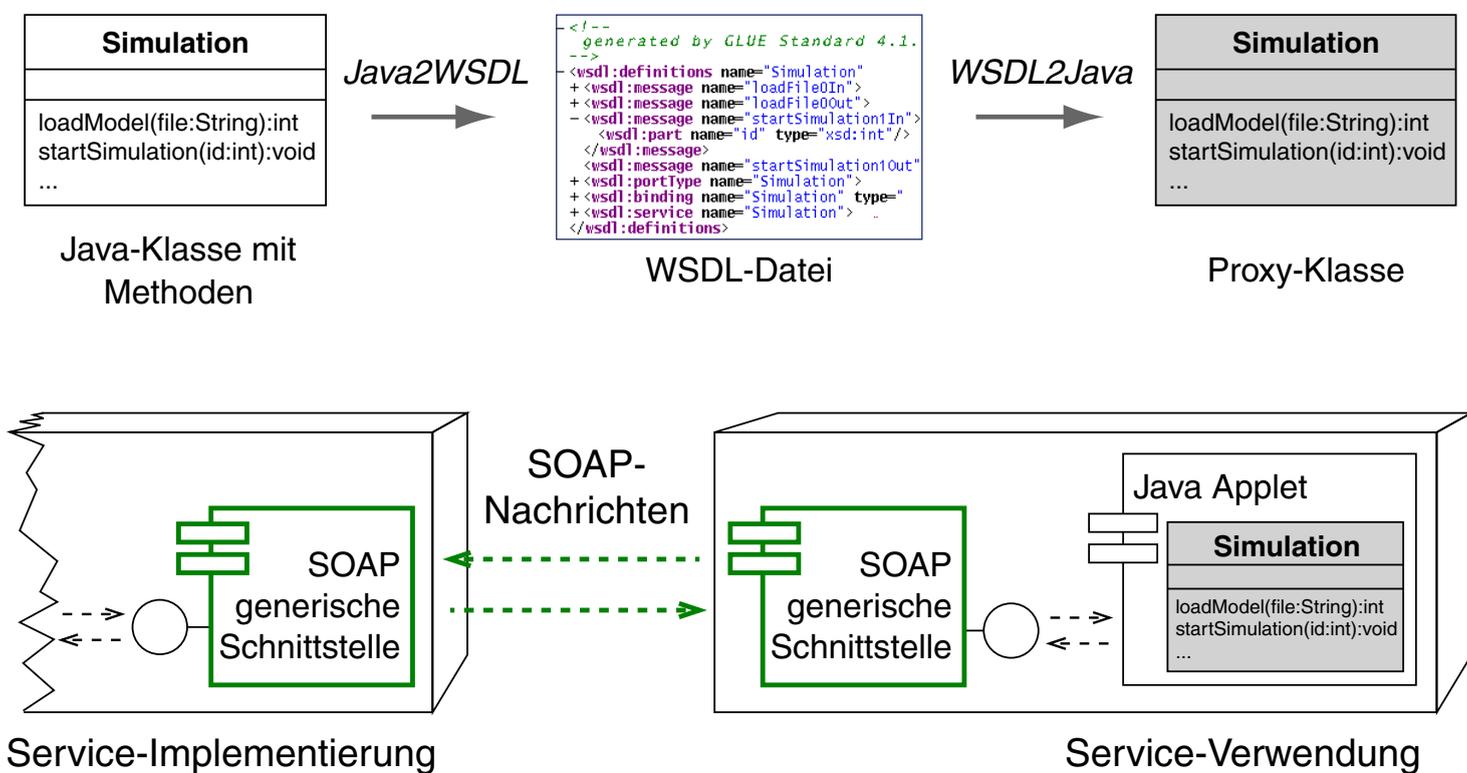
Bei der Anwendungsentwicklung für SOAP-Protokolle ist ein hoher Automatisierungsgrad möglich:



# Web-Services

## Automatisierungsaspekte (Fortsetzung)

Bei der Anwendungsentwicklung für SOAP-Protokolle ist ein hoher Automatisierungsgrad möglich:



# Web-Services

## Automatisierungsaspekte (Fortsetzung)

Vollautomatische Kommunikation zwischen Applikationen via Internet (Stichwort: „Semantic Web“) auf Basis von UDDI: Universal Description, Discovery, and Integration (of web services).

