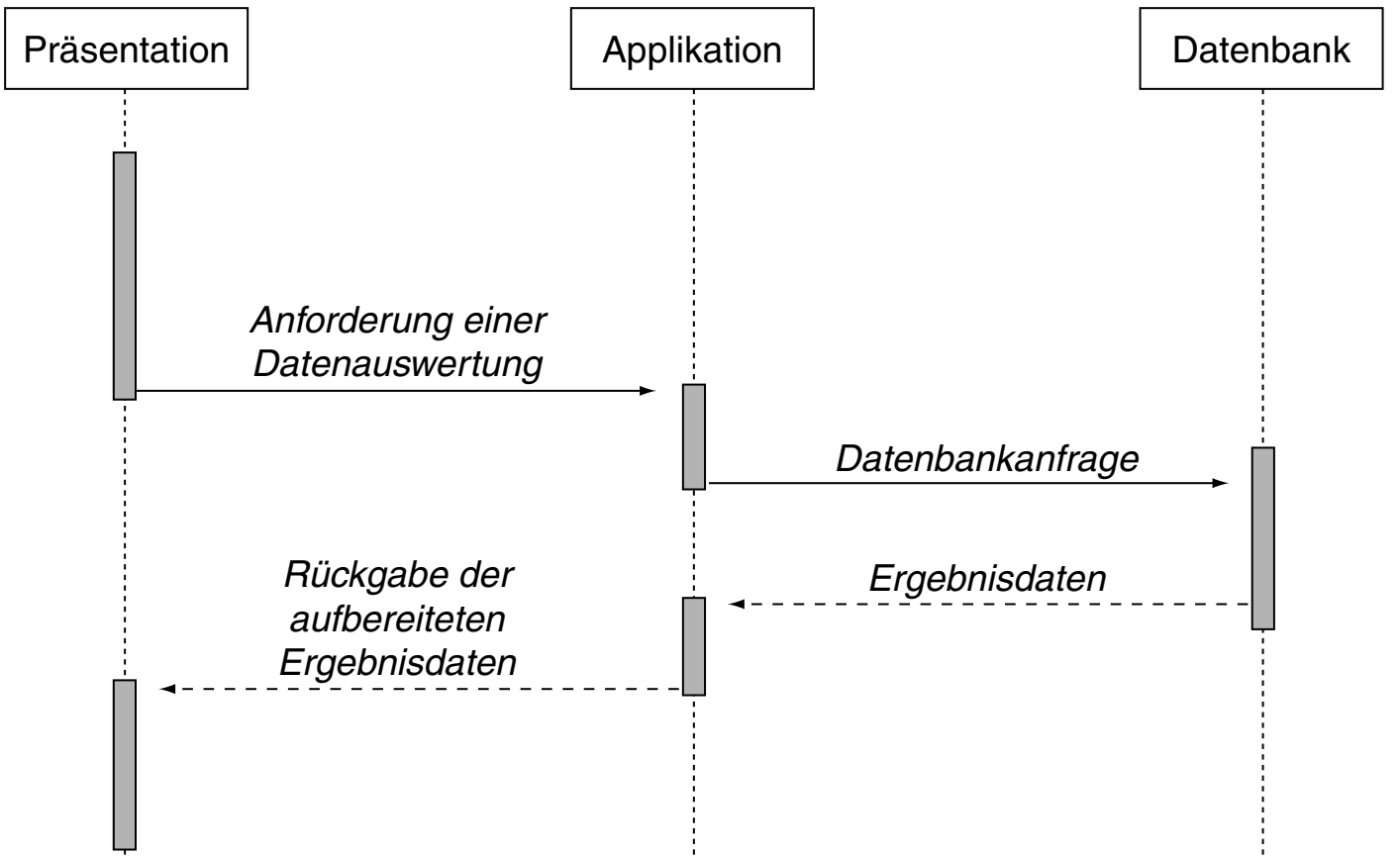
Kapitel WT:VI

VI. Architekturen und Middleware

- Client-Server-Architekturen
- Ajax
- REST
- Remote Procedure Call Systems
- Distributed Object Systems
- Web-Services mit SOAP
- Message-Oriented Middleware

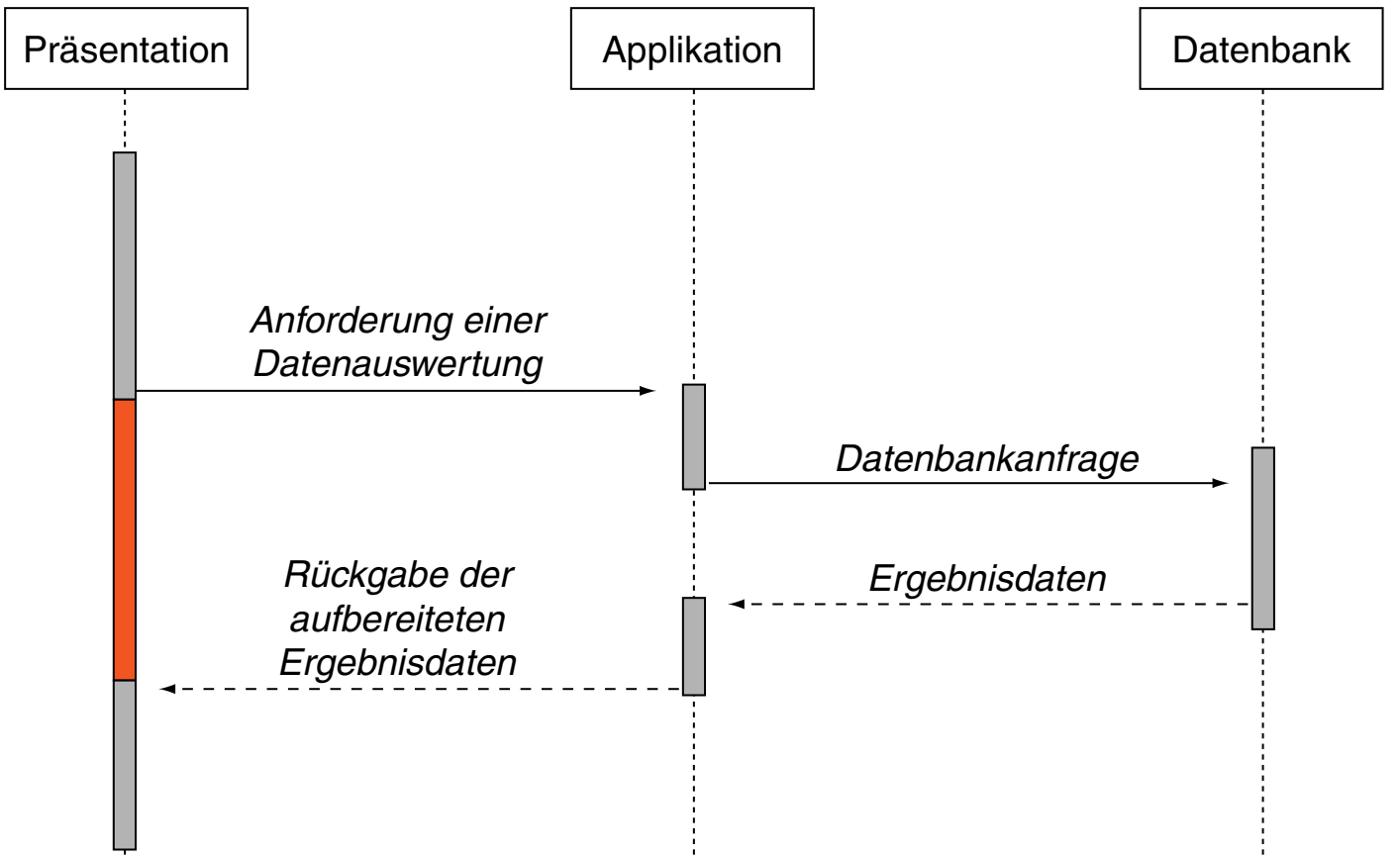
Client-Server-Architekturen

3-Tier Architektur: Sequenzdiagramm



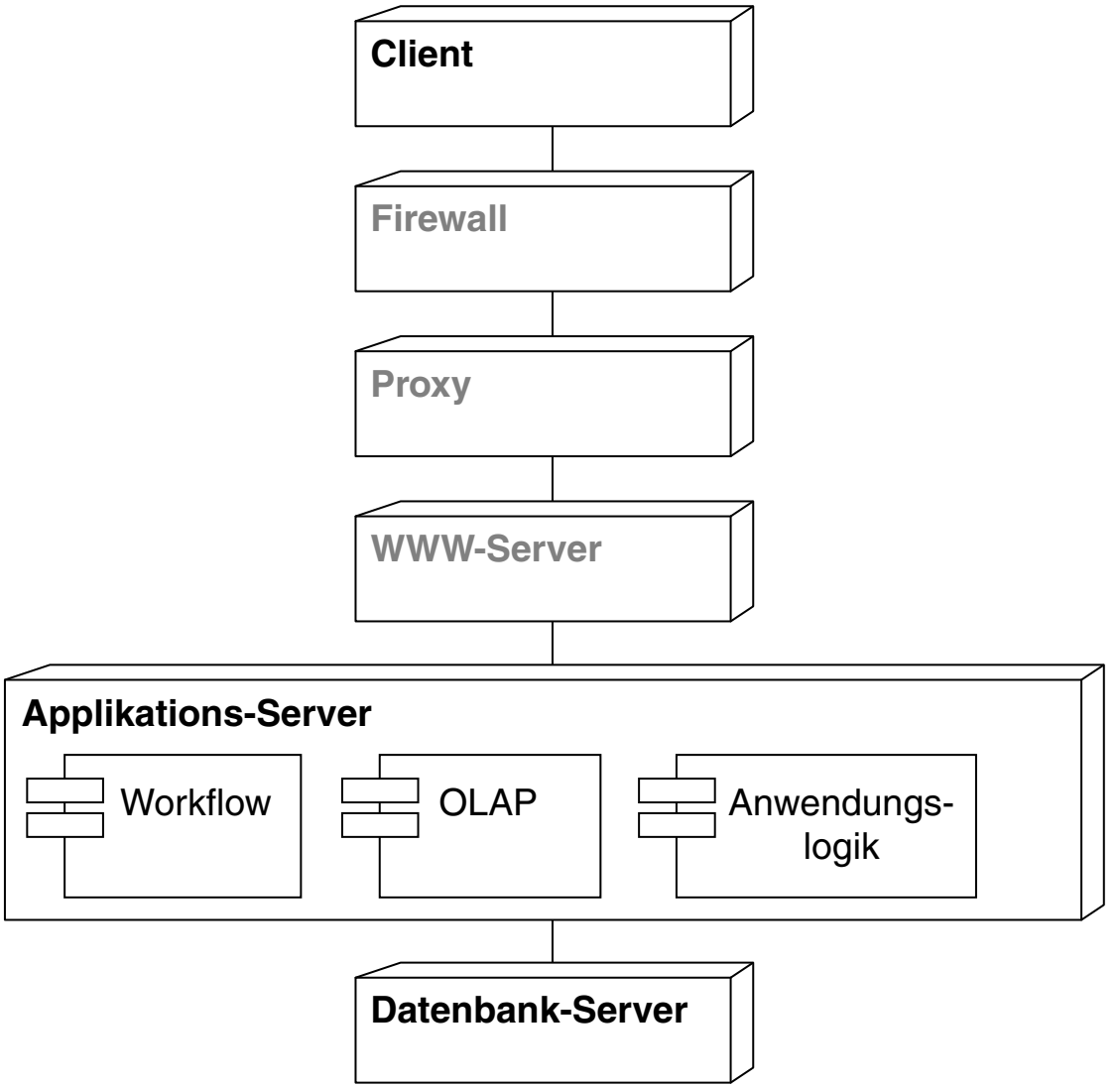
Client-Server-Architekturen

3-Tier Architektur: Sequenzdiagramm



Client-Server-Architekturen

3-Tier Architektur: Deployment-Diagramm



Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser
Portal	im Portal-Applikationsserver	Portlets im Browser

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser
Portal	im Portal-Applikationsserver	Portlets im Browser
Rich Thin Client	sowohl im Client als auch im Applikationsserver	Java-AWT-GUI in speziellem Java Applet im Webbrowser

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser
Portal	im Portal-Applikationsserver	Portlets im Browser
Rich Thin Client	sowohl im Client als auch im Applikationsserver	Java-AWT-GUI in speziellem Java Applet im Webbrowser
Rich Fat Client	clientseitiges Framework (plattformabhängig)	z.B. Java-SWT, JFace-GUI

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser
Portal	im Portal-Applikationsserver	Portlets im Browser
Rich Thin Client	sowohl im Client als auch im Applikationsserver	Java-AWT-GUI in speziellem Java Applet im Webbrowser
Rich Fat Client	clientseitiges Framework (plattformabhängig)	z.B. Java-SWT, JFace-GUI
Managed Client	Client-Java-Middleware mit Client-Applikationsserver für Softwareverteilung, etc.	clientseitiges Framework

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client	im Applikationsserver, z.B. JSP + JavaBeans mit Tomcat, J2EE-EnterpriseJavaBeans	HTML + JavaScript im Webbrowser
Portal	im Portal-Applikationsserver	Portlets im Browser
Rich Thin Client	sowohl im Client als auch im Applikationsserver	Java-AWT-GUI in speziellem Java Applet im Webbrowser
Rich Fat Client	clientseitiges Framework (plattformabhängig)	z.B. Java-SWT, JFace-GUI
Managed Client	Client-Java-Middleware mit Client-Applikationsserver für Softwareverteilung, etc.	clientseitiges Framework
OS-dependent Fat Client	überwiegend im Client: Desktop-Applikation	Windows- oder Linux-GUI

Client-Server-Architekturen [torsten-horn.de]

Architekturmuster	Applikationslogik	Präsentationsschicht
Thin Client		
Portal		
Rich Thin Client		
Rich Fat Client		
Managed Client		
OS-dependent Fat Client		

Client-Server-Architekturen

Implementierung von Architekturmustern

- Ajax.
  - dynamisches (genauer: asynchrones) Web für Thin Clients
- REST.
  - Repräsentation und Manipulation von Ressourcen im Internet
- Remote Procedure Call Systems.
  - RPC, XML-RPC, Java RMI
- Distributed Object Systems.
  - SOAP, DCOM, CORBA
- Message-Oriented Middleware.
  - Broker-basierte 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 [Sequenzdiagramm]

Ajax = Asynchronous JavaScript and XML

Ajax

Einführung [Sequenzdiagramm]

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
- (graphische) Bedienelemente mit Feedback

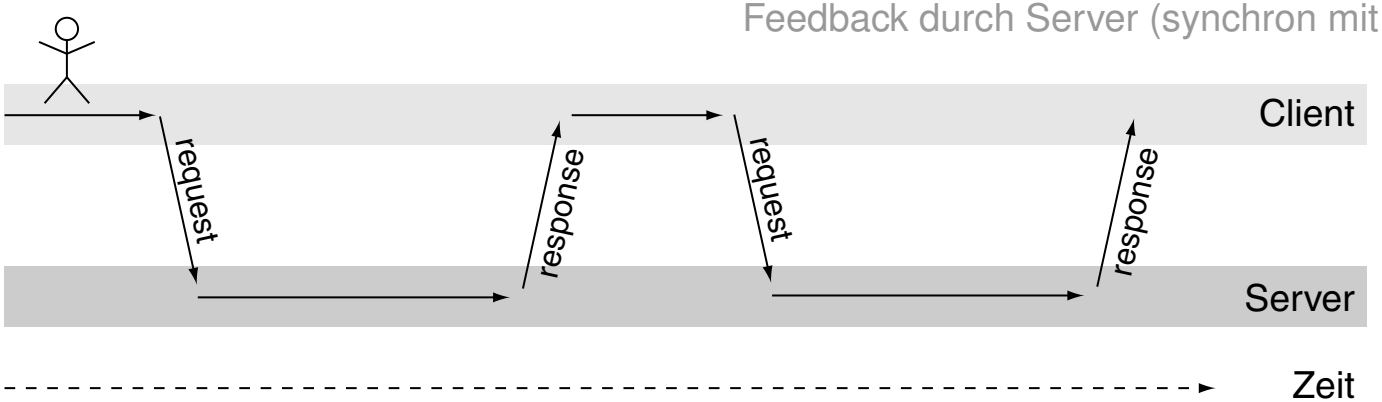
Bemerkungen:

- Die Kernidee von Ajax besteht darin, einen HTTP-Request *nebenläufig* (= non-blocking) auszuführen und das Ergebnis des Requests in den DOM-Seitenbaum des Browsers einzufügen.
- Teilweise wird Ajax als Client-Side-Technologie bezeichnet. [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 [Sequenzdiagramm]

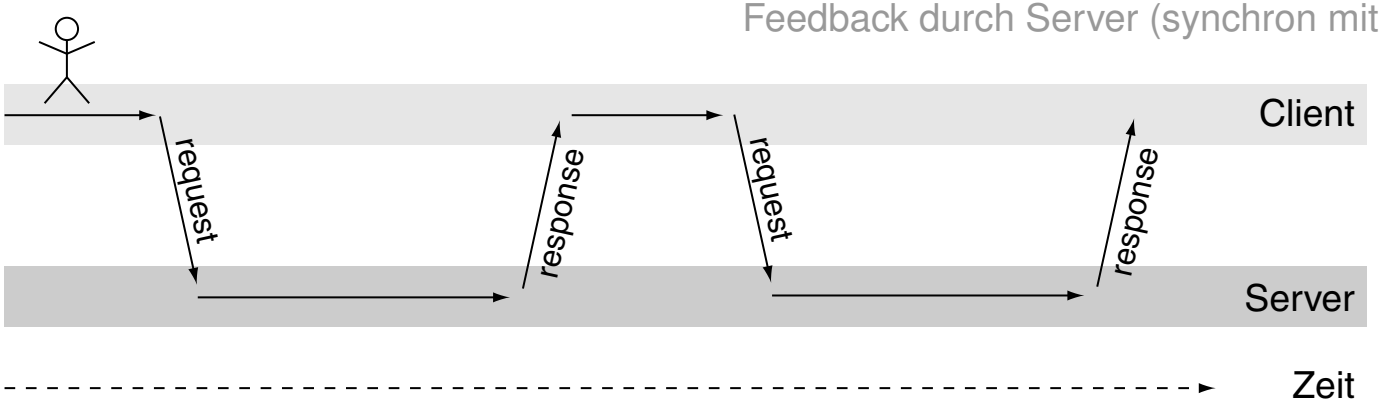
Feedback durch Server (synchron mit Antwort)



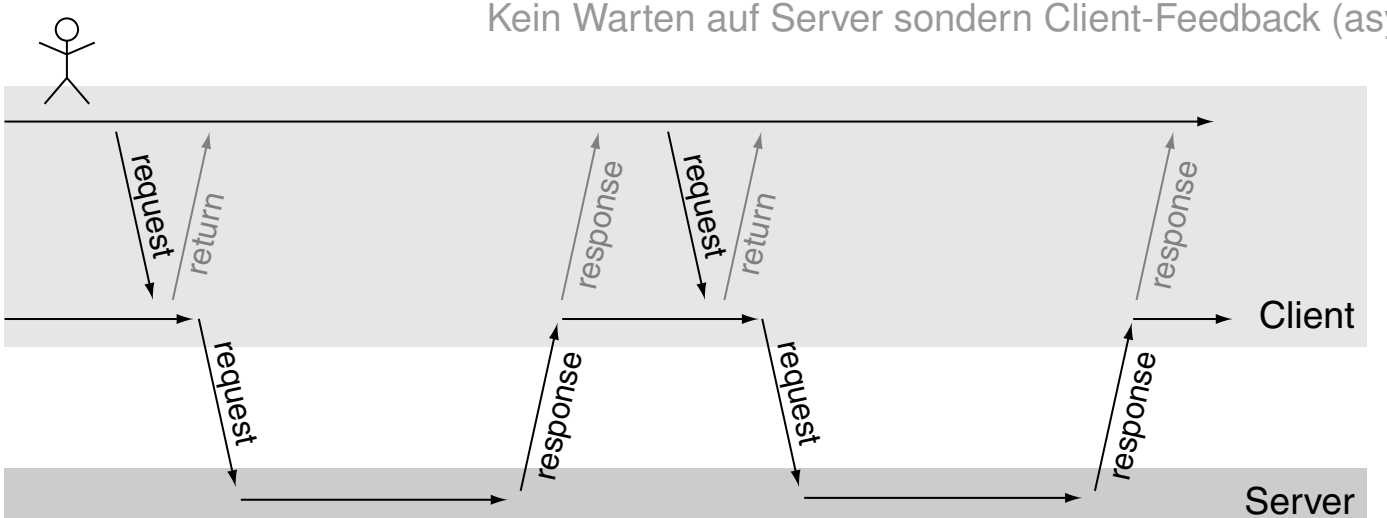
Ajax [Garret 2005]

Einführung [Sequenzdiagramm]

Feedback durch Server (synchron mit Antwort)



Kein Warten auf Server sondern Client-Feedback (asynchron)



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
   - 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 mittels `readyState`-Events vom `XMLHttpRequest`-Objekt aufgerufen (“call back”)
   - 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

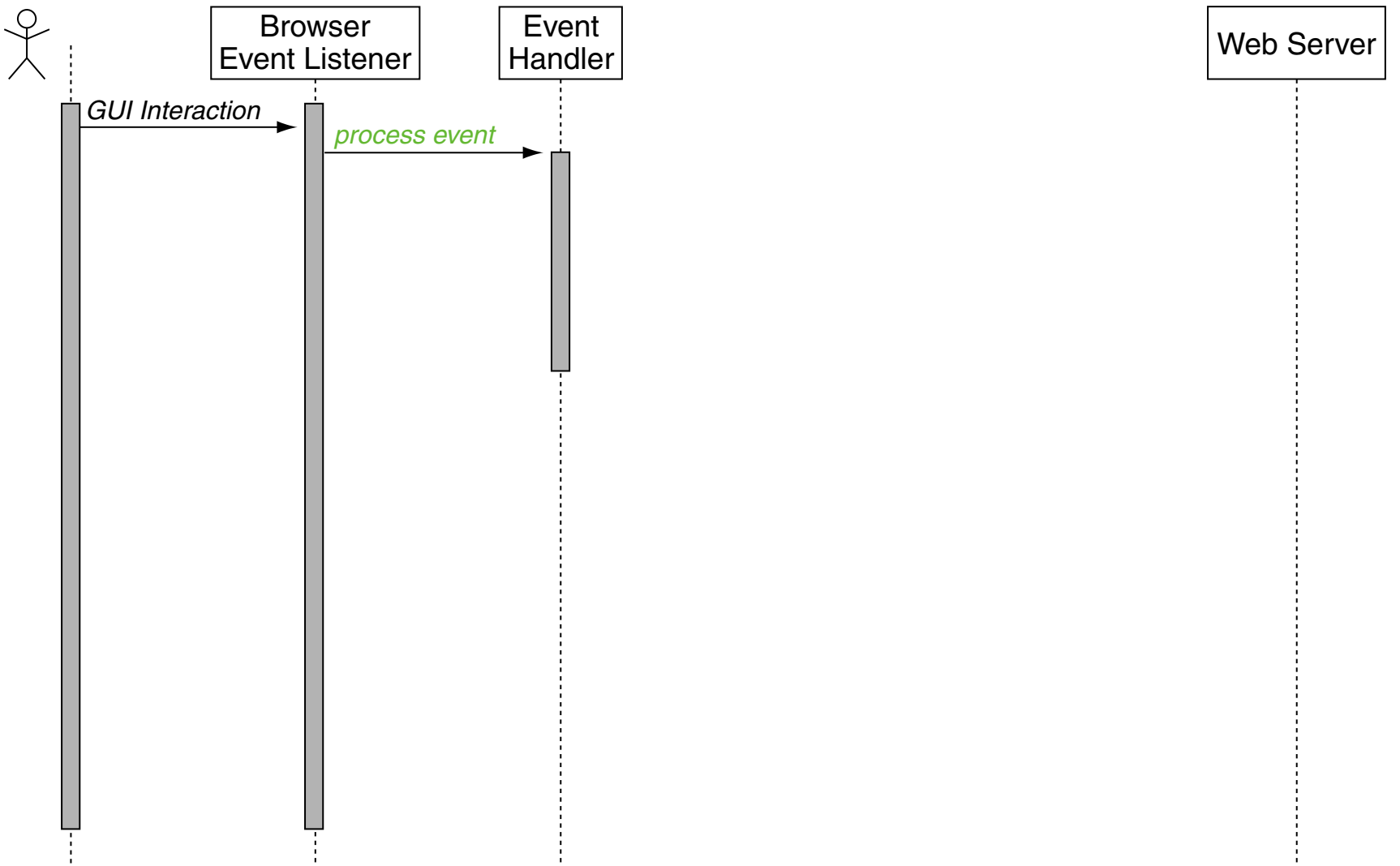
1. Event-Handler.
   - realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
   - wird bei entsprechender Anwenderaktion aufgerufen
   - instanziiert – bei jedem Aufruf – ein `XMLHttpRequest`-Objekt
   - meldet eine Callback-Funktion im `XMLHttpRequest`-Objekt an
   - ruft die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

2. Callback-Funktion.
   - realisiert als JavaScript-Funktion im Client, typischerweise im HTML-Dokument
   - wird mittels `readyState`-Events vom `XMLHttpRequest`-Objekt aufgerufen (“call back”)
   - 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 mit Wurzel `<response>` als Rückgabewert

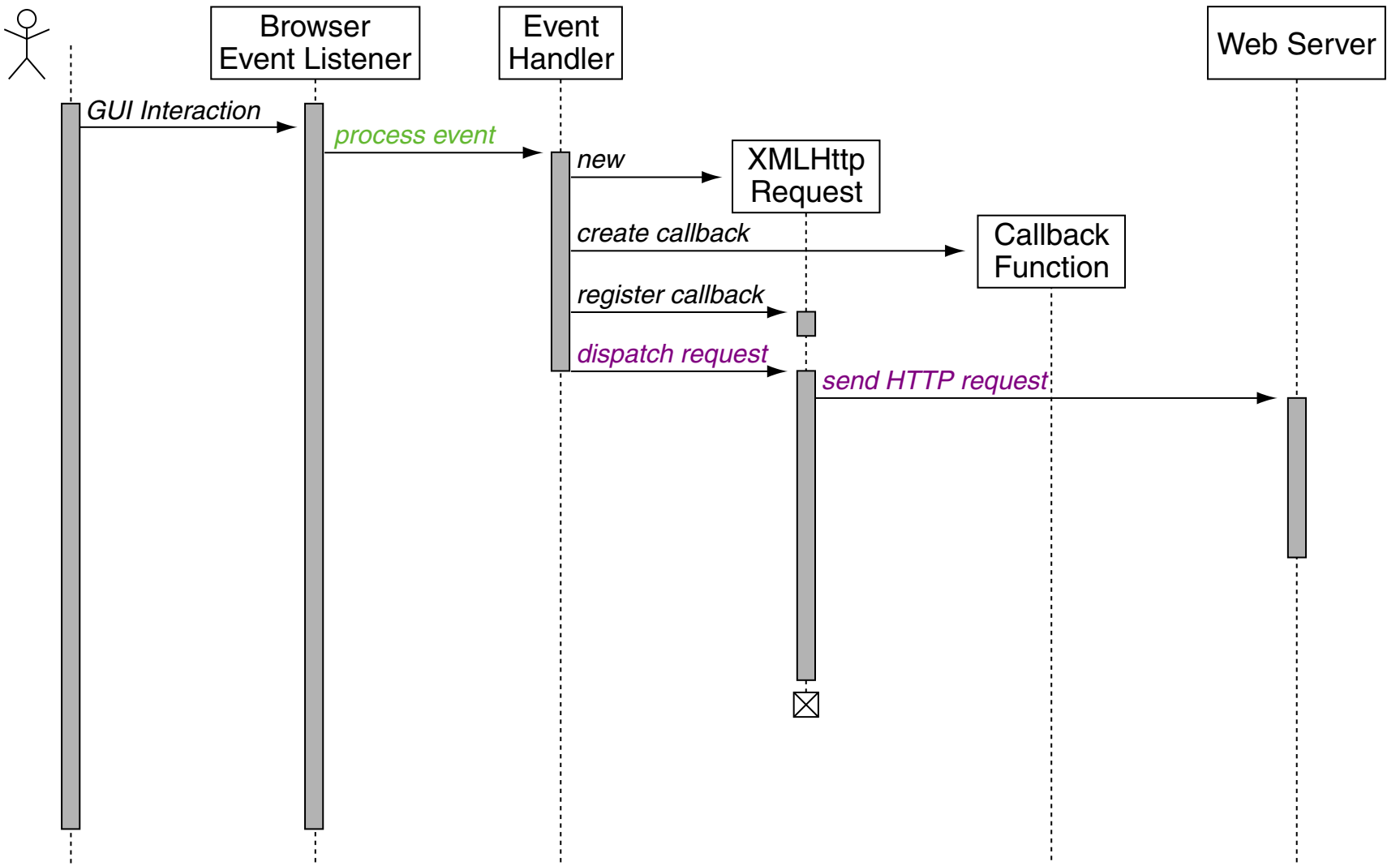
Ajax

Ablauf einer Ajax-Interaktion



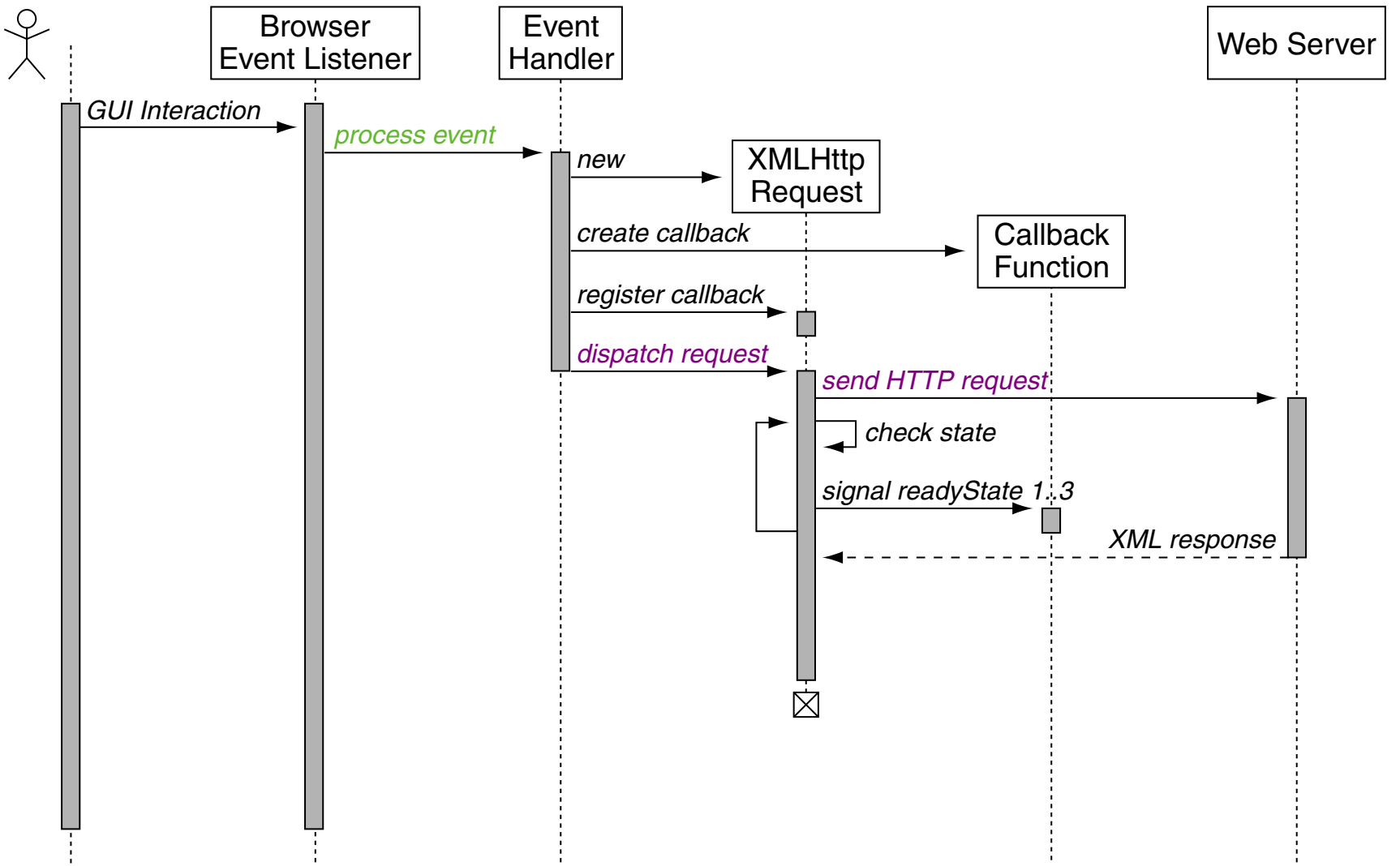
Ajax

Ablauf einer Ajax-Interaktion (Fortsetzung)



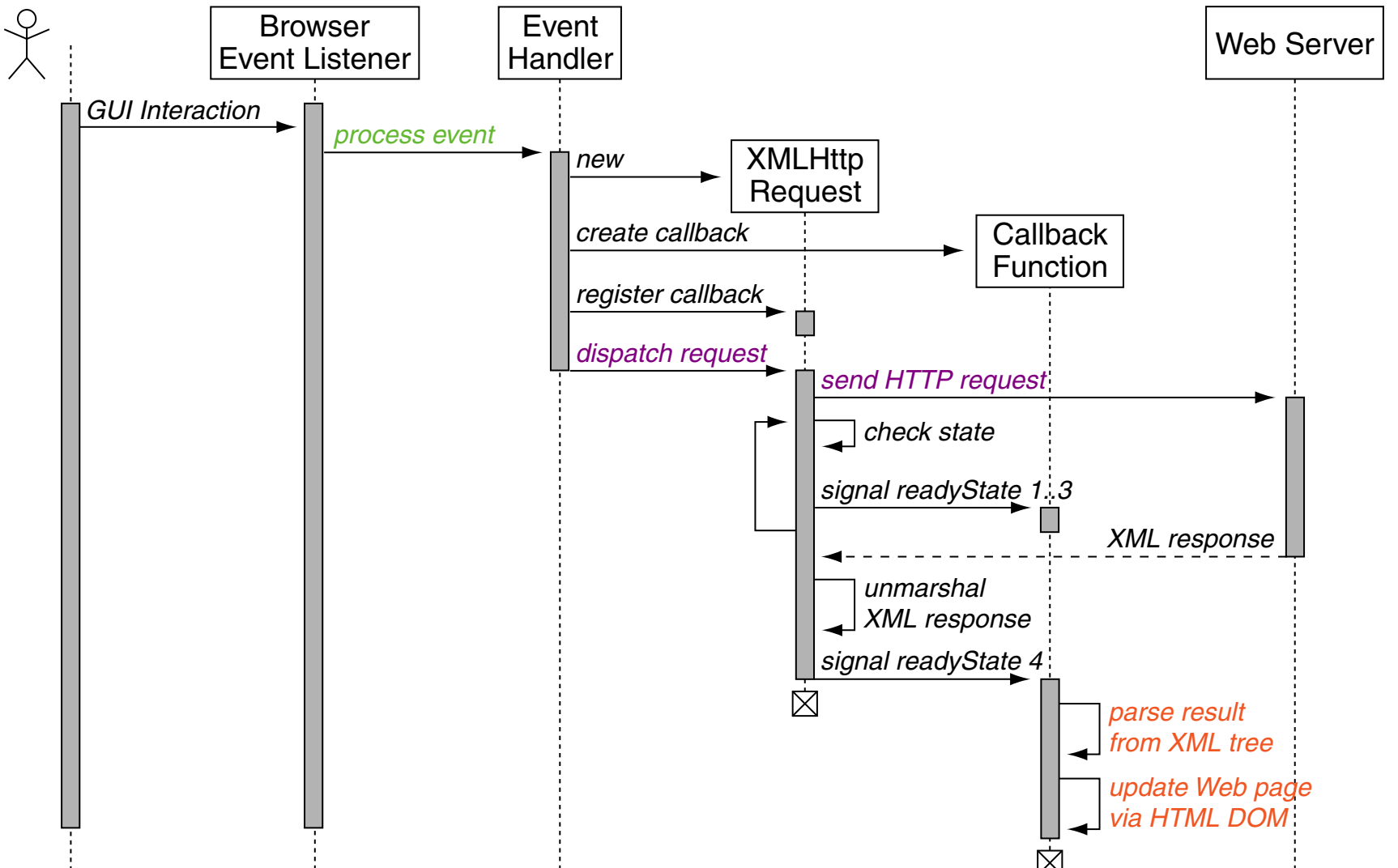
Ajax

Ablauf einer Ajax-Interaktion (Fortsetzung)



Ajax

Ablauf einer Ajax-Interaktion (Fortsetzung)



Ajax

Beispiel: Überwachung von Eingabefeld

[AJAX-Ausführung]

Ajax

Beispiel: Überwachung von Eingabefeld

[AJAX-Ausführung]

HTML-Code:

```
<style type="text/css">
  span.available{background-color: green;}
  span.unavailable{background-color: red;}
</style>
...
<form action="">
  Preferred user name:
  <input name="username" type="text"
    onkeyup="genericEventHandler(
       './check-username.php?q=' + this.value, // URL of server function.
       processUsernameResponse // *Name* of function to process response.
    )" />
  <span class="available" id="nameCheck">     </span>
</form>
```

Ajax

Beispiel: Überwachung von Eingabefeld

[AJAX-Ausführung]

HTML-Code:

```
<style type="text/css">
  span.available{background-color: green;}
  span.unavailable{background-color: red;}
</style>
...
<form action="">
  Preferred user name:
  <input name="username" type="text"
    onkeyup="genericEventHandler(
       './check-username.php?q=' + this.value, // URL of server function.
       processUsernameResponse // *Name* of function to process response.
    )" />
  <span class="available" id="nameCheck">     </span>
</form>
```

Ajax

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

1. Generischer JavaScript-Code für Event-Handler:

```
function genericEventHandler(url, processResponseXML) {
  let req = new XMLHttpRequest();
  if(req) {
    req.onreadystatechange =      // Register callback function.
      function() {                // Create anonymous callback function.
        if(req.readyState == 4) { // Check whether readyState is complete.
          if(req.status == 200) { // Check whether server response is O.K.
            processResponseXML(req.responseXML);
          } else { alert("HTTP error: " + req.status); }
        }
      };
    req.open("GET", url, true);   // Dispatch request.
    req.send(null);               // Send HTTP request.
  }
}
```

Ajax

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

1. Generischer JavaScript-Code für Event-Handler:

```
function genericEventHandler(url, processResponseXML) {
  let req = new XMLHttpRequest();
  if(req) {
    req.onreadystatechange =      // Register callback function.

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

    req.open("GET", url, true);   // Dispatch request.
    req.send(null);               // Send HTTP request.
  }
}
```

Ajax

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

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

```
function processUsernameResponse(xmltree) {
  let result = xmltree.documentElement.getElementsByTagName(
    'result')[0].firstChild.data;
  let message = document.getElementById('nameCheck');

  if(result == 1) { // Update the DOM regarding the database result.
    message.className = 'unavailable';
  } else {
    message.className = 'available'; }

}
```

Bemerkungen:

- Das Beispiel beschreibt ein wiederverwendbares Pattern für Ajax-Anwendungen: der generische JavaScript-Code (1.) kann unverändert in jedem HTML-Dokument zum Einsatz kommen. Zur Verarbeitung der XML-Antwort ist eine spezifische JavaScript-Funktion (2.) zu definieren. Diese Funktion zusammen mit der Server-Funktion (3.) bilden die Argumente von `genericEventHandler()`, der beliebige Events im HTML-Dokument zugeordnet werden kann.
- Der Aufruf von `function(){...}` in `genericEventHandler()` definiert eine anonyme Callback-Funktion in Form eines Funktionsliterals.
  Programmiersprachentechnische Besonderheit: Bei der anonymen Callback-Funktion handelt es sich um eine Closure. [Wikipedia]

Ajax

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [PHP-Ausführung] :

```
<?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

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [PHP-Ausführung] :

```
<?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

Beispiel: Überwachung von Eingabefeld (Fortsetzung)

3. Server-Funktion `check-username.php` generiert XML-Antwortdatei [PHP-Ausführung] :

```
<?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>
```

REST

Einführung

REST = Representational State Transfer

REST

Einführung

REST = Representational State Transfer

Im Web-Kontext:

- vorhandene Web-Technologie für neue Web-Service-APIs nutzen

➜ URLs definieren Ort und Namen der Ressourcen eines Web-Service

➜ die (Namen der) Request-Methoden des HTTP-Protokolls bezeichnen (passend ihrer Semantik) die Funktionen eines Web-Service

Historie

1995 Ursprung ist das von Roy Fielding entworfene HTTP Object Model

2000 Dissertation Roy Fielding: REST-Architekturstil bzw. *RESTful Application*

2014 steigende Aufmerksamkeit und Akzeptanz in der Web-Community

REST

Einführung (Fortsetzung)

Geforderte Eigenschaften einer RESTful Application:

1. Client-Server-Architektur (Server stellt Dienst für Clients bereit)
2. Zustandslosigkeit (jede REST-Nachricht enthält alle notwendigen Informationen)
3. Ausnutzung von HTTP Caching
4. einheitliche Schnittstelle (siehe Web-Kontext: Namen der HTTP-Request-Methoden)
5. Systeme sind mehrschichtig (Vereinfachung der Architektur)
6. Code on Demand (Client kann Code zur lokalen Ausführung erhalten)

REST

Einführung (Fortsetzung)

Geforderte Eigenschaften einer RESTful Application:

1. Client-Server-Architektur (Server stellt Dienst für Clients bereit)
2. Zustandslosigkeit (jede REST-Nachricht enthält alle notwendigen Informationen)
3. Ausnutzung von HTTP Caching
4. einheitliche Schnittstelle (siehe Web-Kontext: Namen der HTTP-Request-Methoden)
5. Systeme sind mehrschichtig (Vereinfachung der Architektur)
6. Code on Demand (Client kann Code zur lokalen Ausführung erhalten)

Anwendung:

- Maschine-zu-Maschine-Kommunikation
- langlebige und selbstdokumentierende Web-Services

Bemerkungen: [Wikipedia]

- Für die Umsetzung des REST-Paradigmas wird ein zustandsloses Client-Server-Protokoll verwendet. Als Anwendungsschicht-Protokolle werden hauptsächlich HTTP und HTTPS eingesetzt.
- Wird über HTTP zugegriffen, so gibt die verwendete HTTP-Methode, darunter GET, POST, PUT und DELETE, an, welche Operation des Dienstes gewünscht ist.
- Die Methoden GET, HEAD, PUT und DELETE müssen laut HTTP-Spezifikation idempotent sein, was in diesem Zusammenhang bedeutet, dass das mehrfache Absenden der gleichen Anforderung sich nicht anders auswirkt als ein einzelner Aufruf.
- REST ist ein Programmierparadigma, das mit verschiedenen Mechanismen implementiert werden kann. Eine Besonderheit ist die Verwendung passender Namen von HTTP-Methoden in Zusammenhang mit der Semantik der auszuführenden Aktion.

Bemerkungen: (Fortsetzung)

- Für ein tieferes Verständnis der Verwendung von HTTP-Request-Methoden und deren intendierte Semantik sei auf die aktuelle RFC verwiesen [RFC 7231] :

  4.3.1 GET is the primary mechanism of information retrieval [. . .].

  4.3.3 The POST method requests that the target resource process [. . .] the request according to the resource's own specific semantics. For example, [. . .] providing a block of data, [. . .] creating a new resource.

  4.3.4 The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload.

  4.3.5 For example, a resource that was previously created using a PUT request, [. . .] might allow a corresponding DELETE request to undo those actions.

REST

Einheitlichkeit der Schnittstelle

Abbildung der Funktionen (einer API) eines Web-Services auf „klassische“ Operationen bzw. Methodennamen:

Web-Service		CRUD	SQL	Java List	**HTTP**
Funktion 1		create	insert	add	POST
Funktion 2	➜	read	select	get	GET
Funktion 3		update	update	set	PUT
Funktion 4		delete	delete	remove	DELETE
...		...	...	...	...

Vergleiche die intendierte Semantik von HTTP-Request-Methoden. [RFC 7231]

REST

Einheitlichkeit der Schnittstelle

Abbildung der Funktionen (einer API) eines Web-Services auf „klassische“ Operationen bzw. Methodennamen:

Web-Service		CRUD	SQL	Java List	HTTP
Funktion 1		create	insert	add	POST
Funktion 2	➜	read	select	get	GET
Funktion 3		update	update	set	PUT
Funktion 4		delete	delete	remove	DELETE
...		...	...	...	...

Vergleiche die intendierte Semantik von HTTP-Request-Methoden. [RFC 7231]

Aufgabe:

1. Person in Datenbank finden.
2. Geburtstag auf einen bestimmten Wert setzen.

REST

Aufgabe: Datenbank mit Personen [personen.xml]

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="personen.xsl" ?>
<personen>
  <person>
    <name>
      <vorname>Alan</vorname>
      <nachname>Turing</nachname>
    </name>
    <geburtstag>23. Juni 1912</geburtstag>
    <beruf>Mathematiker</beruf>
    <beruf>Informatiker</beruf>
  </person>

  <person>
    <name>
      <vorname>Judea</vorname>
      <nachname>Pearl</nachname>
    </name>
    <geburtstag>unknown</geburtstag>
    <beruf>Informatiker</beruf>
  </person>
</personen>
```

REST

Aufgabe: Lösung als *non*-RESTful Web-Service

Aufrufe mit HTTP-GET + Parameter für Funktionen und Daten:

- Zurücksetzen auf Ausgangssituation

  `https://server/parameters/personen?action=reset`

- Anzeigen aller Personen

  `https://server/parameters/personen?action=get`

- Anzeigen der Person an Index 1

  `https://server/parameters/personen?person=1&action=get`

- Setzen des Geburtstages der Person an Index 1

  `https://server/parameters/personen?person=1&action=set&geburtstag=10.10.1949`

REST

Aufgabe: Lösung als RESTful Web-Service

Aufrufe mit HTTP-GET/PUT + Representational State + Ressourcen-URL:

- Vorbereitung: Kopieren der Dateien um Zustände zu senden

```
curl -X GET "https://server/personen.xml" > personen.xml
curl -X GET "https://server/pearl.xml" > pearl.xml
```

- Zurücksetzen auf Ausgangssituation

```
curl -X PUT --data @personen.xml "https://server/rest/personen"
```

- Anzeigen aller Personen

```
curl -X GET "https://server/rest/personen"
```

- Anzeigen der Person an Index 1

```
curl -X GET "https://server/rest/personen/1"
```

- Setzen des Geburtstages der Person an Index 1

```
curl -X PUT --data @pearl.xml "https://server/rest/personen/1"
```

REST

Aufgabe: Lösung als RESTful Web-Service (Fortsetzung)

[Web-Service-Ausführung]

Ajax

Quellen zum Nachlernen und Nachschlagen im Web

- Google. *Google Web Toolkit GWT.* code.google.com/p/webtoolkit
- Horn. *Technische Kurzdokumentationen.* www.torsten-horn.de/techdocs
- McCarthy. *Ajax for Java developers: Build dynamic Java applications.* 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
- W3 Schools. *Ajax Introduction.* www.w3schools.com/ajax/xml/ajax_intro.asp