# Web Programming Showdown
# Softwarepraktikum, SS 2004

**Matthias Neubauer**

 © 2004 Peter Thiemann, Matthias Neubauer

# 1 Crashkurs Webprogrammierung

## 1.1 Hintergrund

### 1.1.1 Das Internet

- globales Kommunikationssystem
- Verbindungen zwischen angeschlossenen Endgeräten

  **unicast** Rechner — Rechner *vs.*

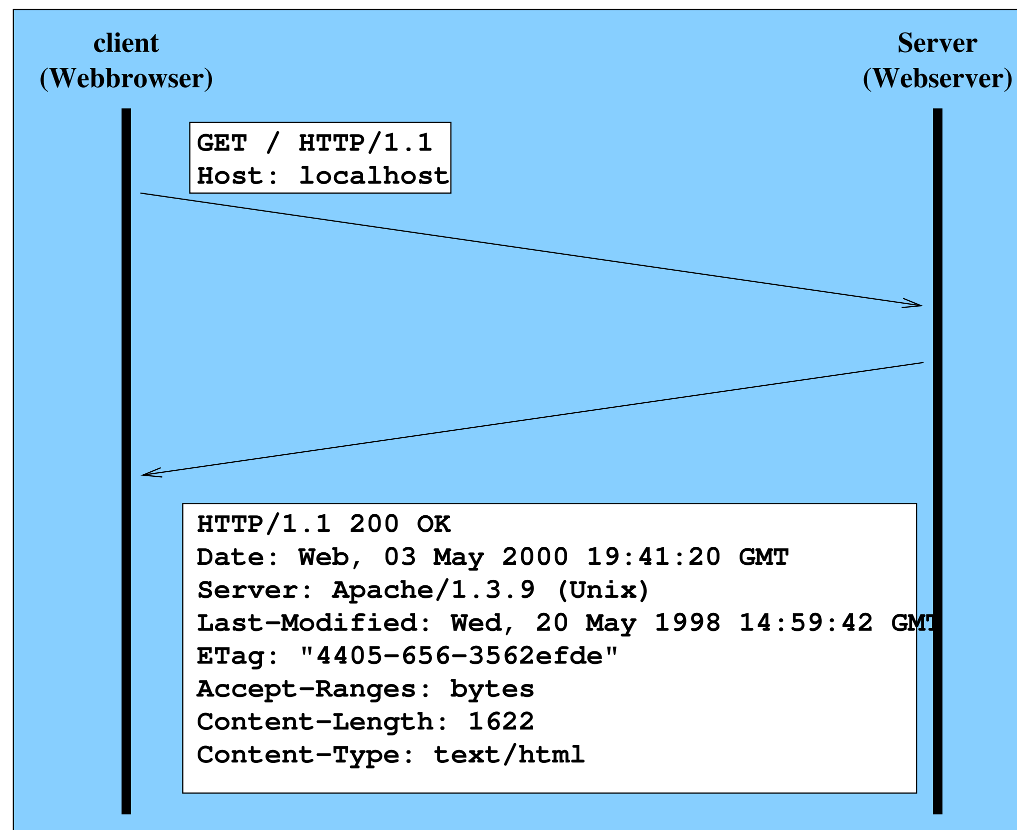
  **multicast** ein Rechner — viele Rechner
- einheitlicher Adressraum
  (Internet-Adressen, Domainnamen)

 © 2004 Peter Thiemann, Matthias Neubauer

## 1.1.2 Protokolle

**Protokoll** Spezifikation der Struktur einer Kommunikation

**Beispiel** HTTP (Hypertext Transfer Protocol)




 © 2004 Peter Thiemann, Matthias Neubauer

### 1.1.3 Dienste (Services)

- Protokolle auf höherer Abstraktionsebene
- meist aufbauend auf TCP/IP (Low-level Internet-Protokoll für strombasierte Verbindungen)
- meist Client-Server Struktur
  selten: Peer-to-peer Protokolle z.B.
- Ein Rechner – mehrere Dienste, *Portnummern*
  Telnet, E-Mail, FTP, WWW, ...

 © 2004 Peter Thiemann, Matthias Neubauer

### 1.1.4 Beispiel eines Dienstes: WWW

Hypertext Transfer Protocol (HTTP 1.1) RFC 2616

- Erweiterbares Protokoll zur Übertragung und Manipulation von getypten Dokumenten
- Adressierung der Dokumente durch URIs (Uniform Resource Identifiers)

**Server:** Apache, CERN, NCSA, MS IIS, Jigsaw, ...

**Clients:** `mozilla`, `netscape`, `msie`, `amaya`, `wget`, ...

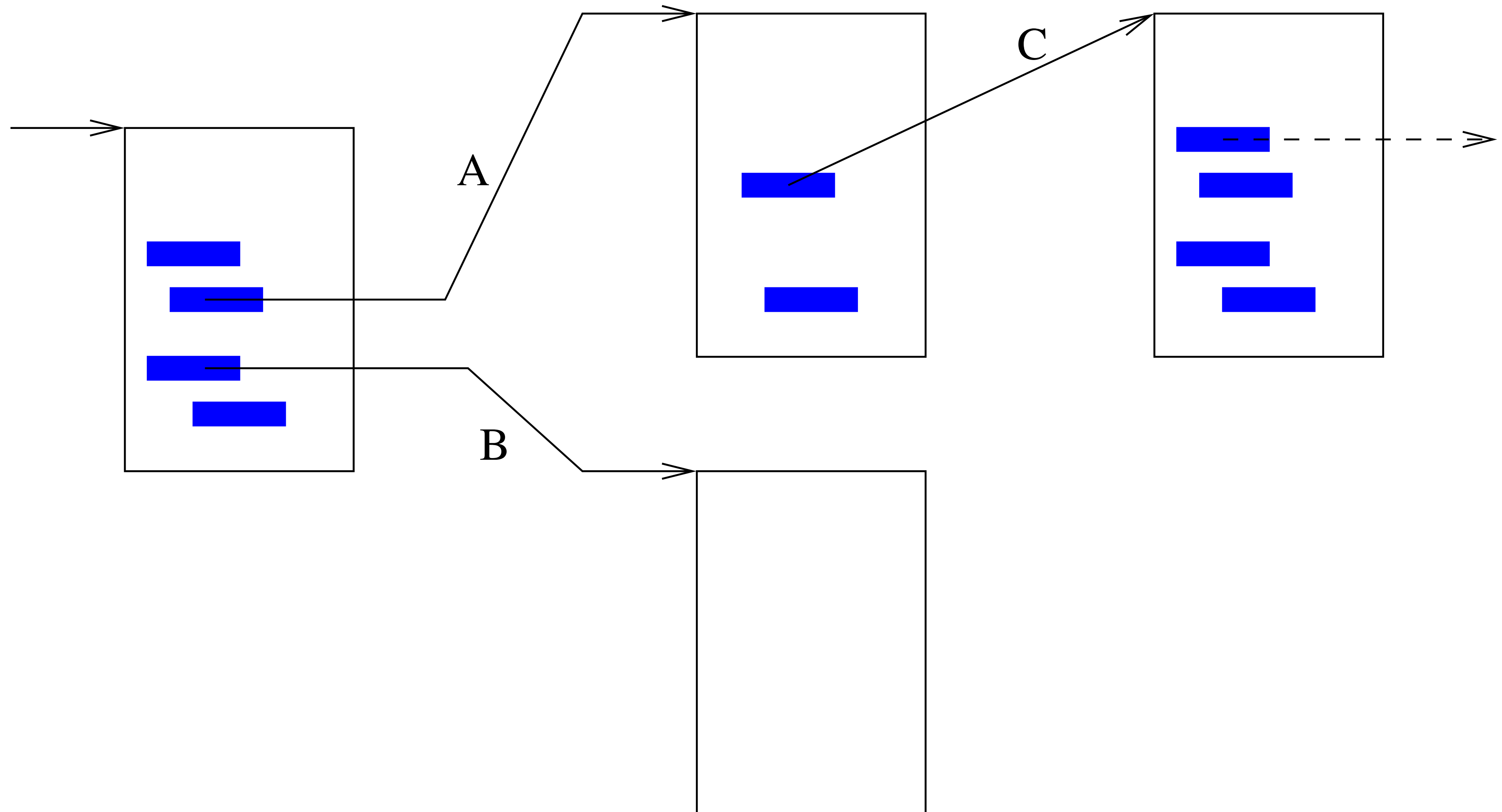
 © 2004 Peter Thiemann, Matthias Neubauer

### 1.1.5 Webservices

- Protokolle auf Anwendungsebene
- meist aufbauend auf HTTP
- Client-Server Struktur
- interaktive Anwendungen mit Webbrowser als GUI

 © 2004 Peter Thiemann, Matthias Neubauer

# Webscripting




 © 2004 Peter Thiemann, Matthias Neubauer

## 1.2 XML

- eXtensible Markup Language
- abgeleitet von SGML
- definierbare logische Struktur:
  markierte, sortierte Bäume
- mächtige Hyperlinkmöglichkeiten (XLink, XPointer)
- Transformationssprachen (XSL)
- basiert auf Unicode
- Anwendungen: WWW, Datenaustausch, E-Commerce
- spez. Versionen: XHTML, WAP/WML, VoiceXML, . . .

 © 2004 Peter Thiemann, Matthias Neubauer

# Beispiel: Wohlgeformtes XML Dokument

```
<?xml version="1.0"?>
<ELTERN>
<KIND>
Hier kommt der Inhalt.
</KIND>
<LEER />
</ELTERN>
```

**XML Deklaration:** `<?xml version="1.0"?>`

**Elemente:** `ELTERN`, `KIND`, `LEER`

**Tags:** `<ELTERN>`, `<KIND>` und `</KIND>`, `</ELTERN>`

**Leeres Element:** `<LEER />`

 © 2004 Peter Thiemann, Matthias Neubauer

# Beispiel: Gültiges XML Dokument

- muß die logische Struktur definieren

→ DTD (Document Type Definition)

```
<?xml version="1.0"?>
<!DOCTYPE ELTERN [
  <!ELEMENT ELTERN (KIND*)>
  <!ELEMENT KIND (MARKE?,NAME+)>
  <!ELEMENT MARKE EMPTY>
  <!ELEMENT NAME (NACHNAME+,VORNAME+)*>
  <!ELEMENT NACHNAME (#PCDATA)>
  <!ELEMENT VORNAME (#PCDATA)>
  <!ATTLIST MARKE
          NUMMER ID #REQUIRED
          GELISTET CDATA #FIXED "ja"
          TYP (natürlich|adoptiert) "natürlich">
  <!ENTITY STATEMENT "Wohlgeformtes XML">
]>
```


© 2004 Peter Thiemann, Matthias Neubauer

```
<ELTERN>
 &STATEMENT;
 <KIND>
  <MARKE NUMMER="1" GELISTET="ja" TYP="natürlich" />
  <NAME>
   <NACHNAME>Flavius</NACHNAME>
   <VORNAME>Secundus</VORNAME>
  </NAME>
 </KIND>
</ELTERN>
```

 © 2004 Peter Thiemann, Matthias Neubauer

### 1.2.1 Inhalte eines XML Dokuments

- Elemente (benannte Baumknoten)
- Attribute (Namen/Wert-Paare an Baumknoten)
- Referenzen
  - *character reference* <, >
  - *entity reference* &STATEMENT;
  - *parameter-entity reference* %ISOLat2; (nur in DTD)
- <!-- Kommentare -->
- Verarbeitungsanweisungen (*processing instructions*) <?*name daten*?> werden an Anwendung überreicht

```
<?xml:stylesheet type="text/css2" href="style.css"?>
```

- CDATA (*character data*)

```
<![CDATA[ \emph{uninterpretierte Daten} ]]>
```

- Vorspann
  - <?xml version="1.0" encoding="UTF-8"?>
  - Spezifikation einer internen/externen DTD


© 2004 Peter Thiemann, Matthias Neubauer

## 1.3 Hypertext Transfer Protocol (HTTP)

Aus der Definition von HTTP/1.1 (RFC 2616):

> The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers and distributed object management systems, through extension of its request methods, error codes and headers. A feature of HTTP is the typing and negotiation of data representation, allowing systems to be built independently of the data being transferred.

 © 2004 Peter Thiemann, Matthias Neubauer

# Beispiel einer HTTP Kommunikation

**Aufbau der Verbindung zum WWW-Server**

```
[hanauma] 107 > telnet localhost www
```

TCP/IP Verbindung zum Rechner *localhost* an den Port *www* (80)

**Antwort von Telnet**

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
```

**Anfrage (Request) an den WWW-Server**

```
1|GET / HTTP/1.1
2|Host: localhost
3|
```

Request besteht nur aus Kopf (Header), der Rumpf (Body) ist leer

 © 2004 Peter Thiemann, Matthias Neubauer

**Kopf (Header) der Antwort**

```
1|HTTP/1.1 200 OK
2|Date: Wed, 03 May 2000 19:41:20 GMT
3|Server: Apache/1.3.9 (Unix)
4|Last-Modified: Wed, 20 May 1998 14:59:42 GMT
5|ETag: "4405-656-3562efde"
6|Accept-Ranges: bytes
7|Content-Length: 1622
8|Content-Type: text/html
9|
```

Leerzeile signalisiert das Ende des Headers

**Rumpf (Body) der Antwort** in diesem Fall ein HTML-Dokument

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<HTML>
 <HEAD>
  <TITLE>Test Page for Apache Installation on Web Site</TITLE>
 </HEAD>
<!-- ... -->
</HTML>
```

 © 2004 Peter Thiemann, Matthias Neubauer

# Format einer Anfrage

$$\begin{aligned}
\langle \mathit{Request} \rangle \quad ::= \quad & \langle \mathit{Request\text{-}Line} \rangle \\
& \Big( \big( \langle \mathit{general\text{-}header} \rangle \mid \langle \mathit{request\text{-}header} \rangle \mid \langle \mathit{entity\text{-}header} \rangle \big) \langle \mathit{CRLF} \rangle \Big)^{*} \\
& \langle \mathit{CRLF} \rangle \\
& [\langle \mathit{message\text{-}body} \rangle]
\end{aligned}$$

Jede Zeile wird durch $\langle \mathit{CRLF} \rangle$, CR (ASCII-Kode 13) gefolgt von LF (ASCII-Kode 10), abgeschlossen.

Vgl. Methodenaufruf

- erste Zeile ($\langle \mathit{Request\text{-}Line} \rangle$):
  Name der Methode und vorgeschriebene Parameter
- Headerzeilen: weitere (optionale) Parameter, durch Schl selworte identifiziert
- Rumpf: optionaler Inhalt der Anfrage (Parameter)

 © 2004 Peter Thiemann, Matthias Neubauer

# Erste Zeile einer Anfrage

$\langle \mathit{Request\text{-}Line} \rangle \quad ::= \quad \langle \mathit{Method} \rangle␣\langle \mathit{Request\text{-}URI} \rangle␣\langle \mathit{HTTP\text{-}Version} \rangle\langle \mathit{CRLF} \rangle$

| | | | | |
|---|---|---|---|---|
| ⟨*Method*⟩ | ::= | `GET` | * | Anfordern eines Dokuments |
| | | \| `HEAD` | * | Anfordern der Header eines Dokuments |
| | | \| `POST` | | Senden einer Anfrage |
| | | \| `PUT` | | Ablegen eines Dokuments |
| | | \| `DELETE` | | L chen eines Dokuments |
| | | \| `TRACE` | | Anfordern der empfangenen Anfrage |
| | | \| `OPTIONS` | | |

* erforderliche Methoden

$$
\begin{array}{lcl}
\langle \mathit{Request\text{-}URI} \rangle & ::= & \langle \mathit{abs_path} \rangle \mid \langle \mathit{absoluteURI} \rangle \mid \ldots \\
\langle \mathit{HTTP\text{-}Version} \rangle & ::= & \texttt{HTTP/1.1} \\
\langle \mathit{abs_path} \rangle & ::= & \texttt{/}\ [\langle \mathit{path} \rangle]\ [\texttt{;}\langle \mathit{params} \rangle]\ [\texttt{?}\langle \mathit{query} \rangle] \\
\langle \mathit{params} \rangle & ::= & \ldots \\
\langle \mathit{query} \rangle & ::= & \ldots
\end{array}
$$

 © 2004 Peter Thiemann, Matthias Neubauer

# Beispiel (Minimalanfrage)

```
1|GET / HTTP/1.1
2|Host: www.informatik.uni-freiburg.de
3|
```

 © 2004 Peter Thiemann, Matthias Neubauer

# Format einer Header-Zeile

$$\langle \mathit{header} \rangle \quad ::= \quad \langle \mathit{key\text{-}token} \rangle : \langle \mathit{value} \rangle$$

Beispiel

$$\langle \mathit{Host} \rangle \quad ::= \quad \texttt{Host:}\ \langle \mathit{host} \rangle\ [\ \texttt{:}\ \langle \mathit{port} \rangle\ ]$$

 © 2004 Peter Thiemann, Matthias Neubauer

# Anfragespezifische Header

| | | | | |
|---|---|---|---|---|
| ⟨*request-header*⟩ | ::= | ⟨*Accept*⟩ | | media types |
| | | \| ⟨*Accept-Language*⟩ | | |
| | | \| ⟨*Host*⟩ | * | Hostname des Servers |
| | | \| ⟨*Referer*⟩ | | URL |
| | | \| ⟨*User-Agent*⟩ | | |
| | | ⋮ | | |

* in jeder direkten Anfrage erforderlich


© 2004 Peter Thiemann, Matthias Neubauer

# Allgemeine Header

K nen in Anfragen und Antworten benutzt werden, meist optional

| | | | | |
|---|---|---|---|---|
| ⟨*general-header*⟩ | ::= | ⟨*Connection*⟩ | + | persistent/one-shot |
| | | \| ⟨*Date*⟩ | +! | |
| | | \| ⟨*Transfer-Encoding*⟩ | + | |

| | |
|---|---|
| + | h fig in Antworten benutzt |
| +! | in jeder Antwort erforderlich |

# Entity Header

Information  er den ⟨*message-body*⟩, falls vorhanden:
Content -Encoding, -Language, -Length, -Location, . . .

 © 2004 Peter Thiemann, Matthias Neubauer

# Format einer Antwort

$$\langle \mathit{Response} \rangle ::= \langle \mathit{Status\text{-}Line} \rangle$$
$$\Big( \big( \langle \mathit{general\text{-}header} \rangle \mid \langle \mathit{response\text{-}header} \rangle \mid \langle \mathit{entity\text{-}header} \rangle \big) \langle \mathit{CRLF} \rangle \Big)^*$$
$$\langle \mathit{CRLF} \rangle$$
$$[\langle \mathit{message\text{-}body} \rangle]$$

**Beispiel**

```
1|HTTP/1.1 200 OK
2|Date: Wed, 03 May 2000 19:41:20 GMT
3|Server: Apache/1.3.9 (Unix)
4|Last-Modified: Wed, 20 May 1998 14:59:42 GMT
5|ETag: "4405-656-3562efde"
6|Accept-Ranges: bytes
7|Content-Length: 1622
8|Content-Type: text/html
9|
```

 © 2004 Peter Thiemann, Matthias Neubauer

# Statuszeile

| | | |
|---|---|---|
| ⟨*Status-Line*⟩ | ::= | ⟨*HTTP-Version*⟩␣⟨*Status-Code*⟩␣⟨*Reason-Phrase*⟩⟨*CRLF*⟩ |
| ⟨*HTTP-Version*⟩ | ::= | `HTTP/1.1` |
| ⟨*Status-Code*⟩ | ::= | ⟨*digit*⟩⟨*digit*⟩⟨*digit*⟩ |
| ⟨*Reason-Phrase*⟩ | ::= | Text ohne ⟨*CRLF*⟩ |

Interpretation des Status-Code

`1xx` Informational – Request received, continuing process

`2xx` Success – The action was successfully received, understood, and accepted

`3xx` Redirection – Further action must be taken in order to complete the request

`4xx` Client Error – The request contains bad syntax or cannot be fulfilled

`5xx` Server Error – The server failed to fulfill an apparently valid request

- Siehe RFC 1700 f  alle reservierten ⟨*Status-Code*⟩s und ⟨*Reason-Phrase*⟩s
- ⟨*Reason-Phrase*⟩s sind nur Empfehlungen, k nen von Server und Client ignoriert und/oder ge dert werden
- Im Fehlerfall enth t ⟨*message-body*⟩ oft weitere Erkl ung


© 2004 Peter Thiemann, Matthias Neubauer

# Inhalt der Nachricht

- beliebige Folge von Oktetts
- falls ⟨*Content-Type*⟩ nicht vorhanden
  - Client darf aufgrund der URI raten
  - falls erfolglos `application/octetstream`
- ⟨*Content-Encoding*⟩: `gzip`, `compress`, `deflate`
  beschreiben Kodierungseigenschaften des urspr glichen Objekts
- ⟨*Transfer-Encoding*⟩ definiert  ertragungskodierung:
  `chunked`, `gzip`, `compress`, `deflate`, `identity`

 © 2004 Peter Thiemann, Matthias Neubauer

# 1.4 Common Gateway Interface (CGI)

CGI (Common Gateway Interface) Skripte erlauben die dynamische Erzeugung von Dokumenten auf dem Server.

Typische Anwendung: CGI-Skripte verarbeiten Eingaben aus Formularen und erzeugen in Abhängigkeit von den Eingaben ein Antwortdokument.

Eigenschaften von CGI:

**Einfachheit**

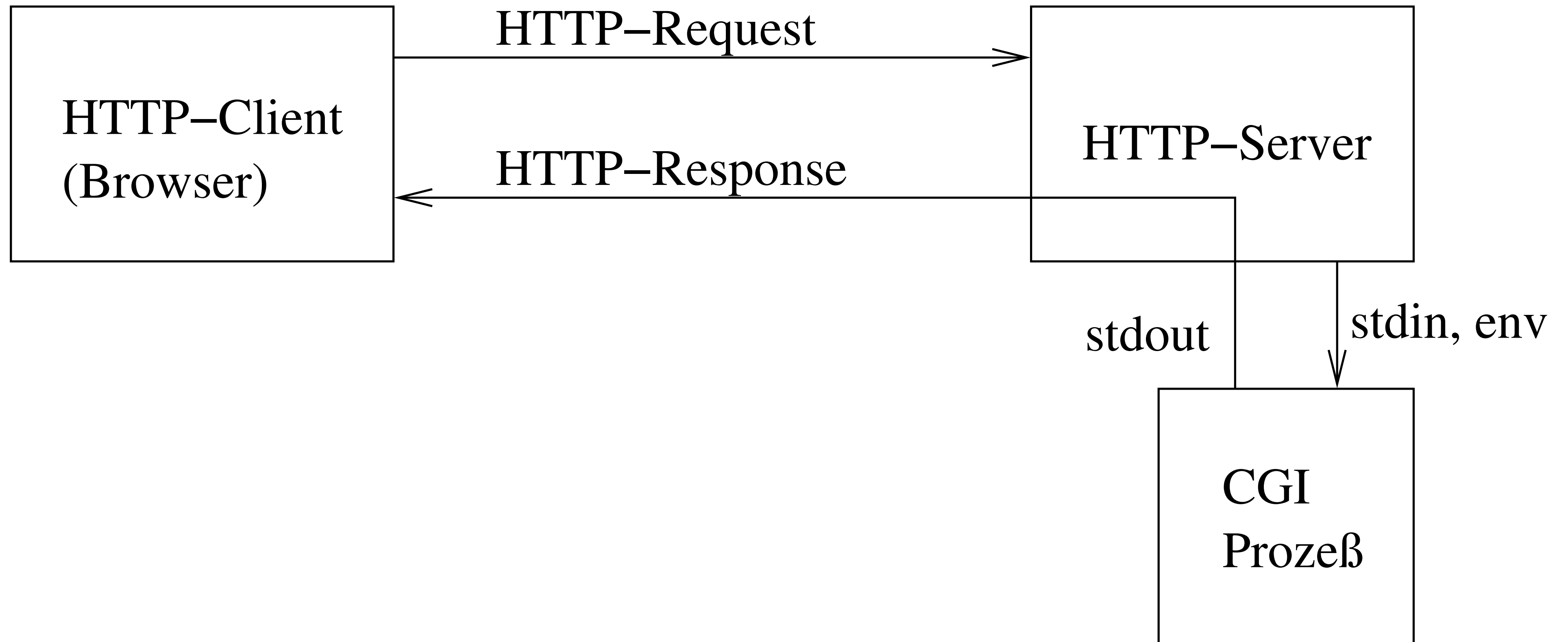
**Sprachunabhängigkeit**

**Sicherheit** durch separaten Prozess

**Offener Standard**

**Architekturunabhängigkeit**

 © 2004 Peter Thiemann, Matthias Neubauer




 © 2004 Peter Thiemann, Matthias Neubauer

# Ausführung eines CGI-Skripts

- Server erkennt CGI-Skripte an der URL
  - spezielles Element im Pfad (z.B. `cgi-bin`), dann nächstes Pfadelement
    = Name eines ausführbaren Programms in konfigurierbarem Verzeichnis
  - spezielle Extension am Dateinamen (z.B. `.cgi`)
    = Name eines ausführbaren Programms
- Server verarbeitet den Header des HTTP-Requests
- Legt Request-Information in Environment ab (Prozessumgebung)
- Generiert die Statuszeile und einige Response-Header (`Date`, `Server`, `Connection`)
- Schliesst den Headerteil der Response **NICHT** ab
- Startet das CGI-Programm mit
  - Standardausgabe ⇔ Versenden an Client/Browser
  - Standardeingabe ⇔ ggf. Lesen vom Client/Browser
  - Argumente ⇔ Pfadelemente **nach** dem Namen des CGI-Programms
  - Umgebung definiert weitere Parameter der Anfrage


© 2004 Peter Thiemann, Matthias Neubauer

# Pflichten eines CGI-Programms

- Interpretation der Parameter und der Anfrage
- Drucken weiterer Headerzeilen
  (`Content-Length`, `Content-Type`, `Content-Encoding`, ...)
- **Abschliessen des Headerteils der Response durch Leerzeile**
- Generieren des Inhaltes

**Sprachen zur CGI-Programmierung**

- Jede geeignet, die Standardeingabe und Umgebungsvariable lesen kann, sowie Standardausgabe schreiben kann
- Für Java ist ein *wrapper* Programm zum Lesen der Umgebungsvariablen erforderlich
- Manche Webserver beinhalten Interpreter für Skriptsprachen (perl, php, etc), um die Startzeit für einen externen Interpreter zu sparen
  Beispiel: Apache Module mod_perl, mod_php, mod_python, mod_ruby, ...

 © 2004 Peter Thiemann, Matthias Neubauer

### 1.4.1 Parameter für ein CGI-Programm

Die Einsendung eines XHTML Formulars liefert

$Feldname_1$=$Wert_1$
$Feldname_2$=$Wert_2$
...
$Feldname_k$=$Wert_k$

wobei Feldnamen wiederholt auftreten können.

Feldnamen und Werte werden vor Übertragung vom Browser kodiert

**Standardkodierung: URL Kodierung** `application/x-www-form-urlencoded`

- Buchstaben und Zahlen bleiben erhalten
- Leerzeichen werden durch + ersetzt
- Alle weiteren Zeichen werden durch %⟨*ASCII-code*⟩ ersetzt
  (in zweistelliger Hexadezimaldarstellung)

vgl.

```
public static String java.net.URLEncoder.encode(String s)
```

 © 2004 Peter Thiemann, Matthias Neubauer

### 1.4.2 Zugriffsmethoden

`GET` Kodierung der Anfrage in der URL durch Anhängen eines ⟨*Querystring*⟩ der Form `?`⟨*Feld-Wert-Liste*⟩ an die `action` URL

$$\langle \mathit{Feld\text{-}Wert\text{-}Liste} \rangle ::= \langle \mathit{kodierter\text{-}Feldname} \rangle\texttt{=}\langle \mathit{kodierter\text{-}Wert} \rangle \left(\texttt{\&}\langle \mathit{kodierter\text{-}Feldname} \rangle\texttt{=}\langle \mathit{kodierter\text{-}Wert} \rangle\right)^*$$

Der Webserver legt den ⟨*Querystring*⟩ in der Umgebungsvariable `QUERY_STRING` ab.

`POST` verschickt die ⟨*Feld-Wert-Liste*⟩ im ⟨*message-body*⟩ der Anfrage. Der Webserver speichert die Länge (in Octets) des ⟨*message-body*⟩ in der Umgebungsvariable `CONTENT_LENGTH`. Das CGI-Programm muss **genau** so viele Octets lesen und interpretieren (nicht bis Dateiende lesen!)

Durch URL Kodierung werden

- unerlaubte Zeichen in URLs vermieden
- die Zeichen = und `&` in Feldnamen und Werten verwendbar

Rationale für Methoden: `GET` beobachtet, `POST` ändert ggf. Zustand des Servers

 © 2004 Peter Thiemann, Matthias Neubauer

## 1.5 PHP

- "Personal Home Page" Tools
- PHP/FI von Rasmus Lerdorf, 1995
  - "Menge von Perl-Skripte zum Managment eines Online-Resumes"
- PHP 3.0, 1998, erste Version, die heutigem PHP sehr ähnelt, Andi Gutmans and Zeev Suraski
- PHP 4, ZEND Engine, 2000

 © 2004 Peter Thiemann, Matthias Neubauer

# Das erste PHP-Skript: `date.php`

```
<html>
  <head><title>Date and Time</title></head>
  <body>
    <h1>Date and Time</h1>
    <p>
      <?php
        echo(date("d. F. Y; H:i"));
      ?>
    </p>
    <p><a href="/index.html">Back to index</a></p>
  </body>
</html>
```

 © 2004 Peter Thiemann, Matthias Neubauer

## 1.6 Perl

- "Practical Extraction and Report Language"
- Perl 1.000, 1987 von Larry Wall

  > Perl is a interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information. It's also a good language for many system management tasks. The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal).

- Perl 5.000, 1994, komplette Reimplementierung
- Apache-Modul `mod_perl` 1996

 © 2004 Peter Thiemann, Matthias Neubauer

# Das erste Perl-Skript: `perl.cgi`

```
#!/usr/bin/perl
require 5.005; use English; use POSIX;
use Getopt::Std; use Time::Local;

print "Content-type: text/html\n\n";
cat <<EOF
<HTML><head><title>Date and Time</title></head>
<body> <h1>Date and Time</h1>
EOF
/bin/date
cat <<EOF
<p>
<a href="/index.html">Back home</a>
</body></html>
EOF
```

 © 2004 Peter Thiemann, Matthias Neubauer

# 1.7 Haskell/WASH

- rein-funktionale Sprache
- statisches Typsystem
- nicht-strikte Auswertung

- Haskell 1.0, 1990
- Haskell 98 Report, 1999

- Web Authoring System Haskell (WASH)
  - Haskell-Libraries
  - Webprogrammierung
  - Email-Verarbeitung
  - von Peter Thiemann

 © 2004 Peter Thiemann, Matthias Neubauer

# Das erste WASH-Skript: `date.wash`

- `date.wash`

```
import CGI
import Time
main = do
  clkt <- getClockTime
  calt <- toCalendarTime clkt
  let str = calendarTimeToString calt
  run $ standardQuery "Date and Time" $ do
    text str
    <p><a href="/index.html">Back home</a></p>
```

- Kompilieren ergibt CGI-Programm `date.cgi`

 © 2004 Peter Thiemann, Matthias Neubauer

## 1.8 Java/JSP

- Sun Microsystems, `http://java.sun.com/`
- Einmal kompiliert, immer wieder und überall ausgeführt
  - `javac`
    Java (*Name*`.java`) → Java-Bytecode (*Name*`.class`)
    Bytecode ist plattformunabhängig
  - Java Virtual Machine (JVM)
    Verifikation des Java-Bytecode
    Interpreter des Java-Bytecode
    oder Just-in-time-Übersetzung (JIT) in Maschinencode
- Standardisierte Bibliotheken (packages)
  `java.lang`, `java.io`, `java.awt`, `java.net`, ...


© 2004 Peter Thiemann, Matthias Neubauer

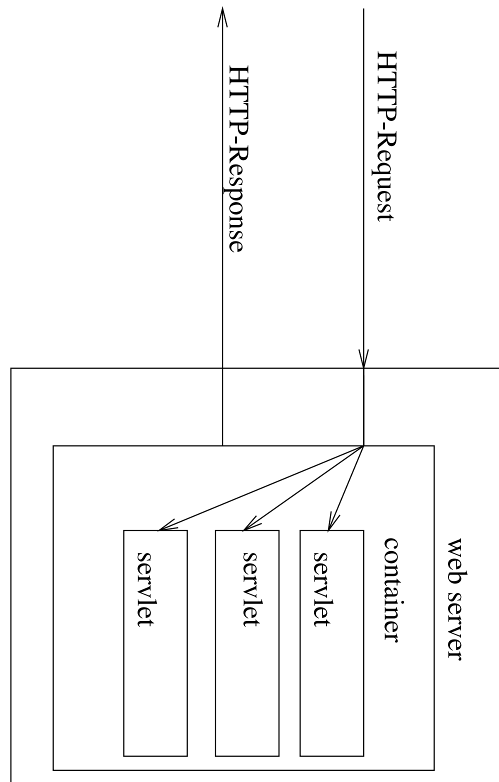
### 1.8.1 Java Servlet technology - “Servlets”

- From the [Java$^{TM}$ Servlet Specification, v2.4]

  > A servlet is a JavaTM technology-based Web component, managed by a container, that generates dynamic content. Like other Java technology-based components, servlets are platform-independent Java classes that are compiled to platform-neutral byte code that can be loaded dynamically into and run by a Java technology-enabled Web server. Containers, sometimes called servlet engines, are Web server extensions that provide servlet functionality. Servlets interact with Web clients via a request/response paradigm implemented by the servlet container.

 © 2004 Peter Thiemann, Matthias Neubauer

# Servlets




 © 2004 Peter Thiemann, Matthias Neubauer

# Servlet Container

- Standort
  - im Server-Prozeß
  - in anderem Prozeß auf Server-Maschine
  - auf anderer Maschine
- Aufgaben
  - Dekodierung von Formulardaten
  - Verbindungsparameter
  - Zustandsverwaltung

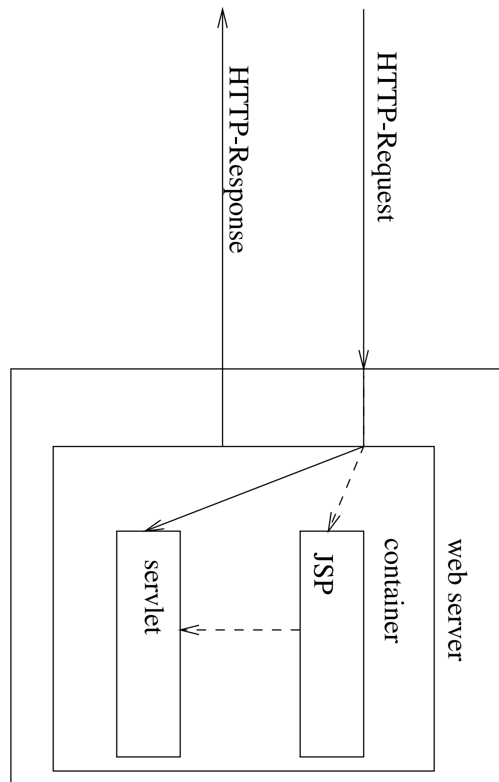
 © 2004 Peter Thiemann, Matthias Neubauer

### 1.8.2 JavaServer Pages (JSP)

- Eigenschaften
  - Flexibles Werkzeug zur Erzeugung von Web-Sites mit dynamischem Inhalt
  - Basiert auf Java (*Write Once, Run Anywhere*)
  - Sämtliche Java-Tools und Packages
  - Wiederverwendung: Template-Seiten
  - Skripting
  - Frontend für Anwendungen mit JavaBeans etc
  - Einfache Erzeugung von XML
- Grundidee: auf dem Server werden textuelle Beschreibungen abgelegt, wie Anfragen zu Antworten verarbeitet werden
- Vorteil: aktiver Inhalt ohne Eingriff der Server-Administration

 © 2004 Peter Thiemann, Matthias Neubauer

# JSP




 © 2004 Peter Thiemann, Matthias Neubauer

# Inhalt einer JSP

- Standarddirektiven
- Standardaktionen
- Deklarationen, Skripte und Ausdrücke der Skriptsprache
  (Java ist erforderlich, andere können unterstützt werden)
- Mechanismus für Tag-Erweiterung

 © 2004 Peter Thiemann, Matthias Neubauer

# Lebenszyklus einer JSP

- Client: Anfrage an eine JSP
- Server: Weiterleiten an JSP Container
- Container: Übersetzung JSP $\rightarrow$ Servlet (falls noch nicht geschehen)
- Container: Laden des Servlets (falls noch nicht geschehen)
- Container: Initialisierung der Servlets; `jspInit ()` (falls noch nicht geschehen)
- Container: Weiterleitung der Anfrage
- Servlet: Ausführung der Anfrage
- Servlet: Generierung der Antwort
- Container, Server: Weiterleitung der Antwort
- Container: Entfernen des Servlets; `jspDestroy ()`

 © 2004 Peter Thiemann, Matthias Neubauer

# Direktiven

- Syntax

  `<%@` $\langle$*Direktive*$\rangle$ `(`$\langle$*Attribut*$\rangle$`="`$\langle$*Wert*$\rangle$`")`$^*$ `%>`

- Abkürzung für XML-Element

  `<jsp:directive.`$\langle$*Direktive*$\rangle$ `(`$\langle$*Attribut*$\rangle$`="`$\langle$*Wert*$\rangle$`")`$^*$ `/>`

- $\langle$*Direktive*$\rangle$ ::= page
  - mehrfach erlaubt, aber jedes Attribut nur einmal definierbar
  - Beispiel

    ```
    <%@ page info="meine erste JSP" %>
    <%@ page buffer="none" isThreadSafe="yes" errorPage="/fehler.jsp" %>
    ```

  - Attribute
    - `language="`$\langle$*scriptingLanguage*$\rangle$`"`
      $\langle$*scriptingLanguage*$\rangle$ = `java`
    - `extends="`$\langle$*fullyQualifiedClass*$\rangle$`"`
    - `import="`$\langle$*importList*$\rangle$`"`
    - `session="`$\langle$*boolean*$\rangle$`"`
    - `isThreadSafe="`$\langle$*yes-or-no*$\rangle$`"`
    - `buffer="`$\langle$*bufferSpec*$\rangle$`"`
      Größe oder `none`
    - `errorPage="`$\langle$*URL*$\rangle$`"`
      bei uncaught exception
    - `contentType="`$\langle$*contentType*$\rangle$`"`

 © 2004 Peter Thiemann, Matthias Neubauer

# Direktiven

- ⟨*Direktive*⟩ ::= `include`
  - Statisches Einfügen während der Übersetzung in Servlet
  - Beispiel

    ```
    <%@ include file="⟨relativeURL⟩" %>
    ```

- ⟨*Direktive*⟩ ::= `taglib`
  - Def. neuer Tags zu Objekten und Methoden aus ⟨*libraryURI*⟩

    ```
    <%@ taglib uri="⟨libraryURI⟩" prefix="⟨tagPrefix⟩" %>
    ```

  - Beispiel

    ```
    <%@ taglib uri="lib.cream.com" prefix="cream" %>
    ...
    <cream:whip>
    ...
    </cream:whip>
    ```

 © 2004 Peter Thiemann, Matthias Neubauer

# Skript-Elemente

- Deklaration

```
<%! ⟨Deklarationen der Skript-Sprache⟩ %>

<jsp:declaration>
  <![CDATA[ ⟨Deklaration der Skript-Sprache⟩ ]]>
</jsp:declaration>
```

- Skripte (*scriptlet*)

```
<% ⟨Fragmente von Anweisungen der Skript-Sprache⟩ %>

<jsp:scriptlet>
  ⟨Fragmente Anweisungen der Skript-Sprache⟩
</jsp:scriptlet>
```

- Ausdrücke

```
<%= ⟨Ausdruck der Skript-Sprache⟩ %>

<jsp:expression>
  ⟨Ausdruck der Skript-Sprache⟩
</jsp:expression>
```

 © 2004 Peter Thiemann, Matthias Neubauer

## Die erste JSP: `date.jsp`

```
<html>
  <!-- Zugriff auf einen Kalender -->
  <jsp:useBean id="clock" class="calendar.jspCalendar" />
  <ul>
    <li>Tag: <%=clock.getDayOfMonth () %>
    <%-- Methode für Monat fehlt! --%>
    <li>Jahr: <%=clock.getYear () %>
  </ul>
</html>
```


© 2004 Peter Thiemann, Matthias Neubauer

# 2 Subversion

- Version Control System

- verwaltet Daten, die sich über die Zeit ändern
- zentrales Repositorium, merkt sich alle Versionen
- Zugriff auf alte Versionen und “Geschichte” der Daten

- Zugriff über ein Netzwerk von unterschiedlichen Orten
- Gemeinsamer Zugriff mehrerer Benutzer

 © 2004 Peter Thiemann, Matthias Neubauer

## 2.1 Das Repositorium

- Hält Informationen in Form eines Dateisystembaumes
- Mehrere Clients lesen oder bearbeiten das Repositorium
- Repostitorium merkt sich alle Änderungen

- “Was ist die neuste Version?”
- “Wie sah das Repositorium letzten Mittwoch aus?”
- “Wer hat zuletzt eine bestimmte Datei geändert? Wie?”

 © 2004 Peter Thiemann, Matthias Neubauer

## 2.2 Problem des gemeinsamen Editierens

- Zwei Projektteilnehmer editieren die gleiche Datei zur selben Zeit
- Abspeichern der modifizierten Dateien nacheinander

$\rightarrow$ erste Version geht verloren

 © 2004 Peter Thiemann, Matthias Neubauer

## 2.3 Subversion: "Copy-Modify-Merge"

- Jeder bekommt eigene, lokale Kopie
- Am Ende: lokale Kopien werden in das zentrale Repositorium "gemergt"
- Bei überlappenden Änderungen: lokale Konfliktbehandlung durch Benutzer

 © 2004 Peter Thiemann, Matthias Neubauer

## 2.4 Wichtige Subversion-Befehle - 1 -

- Hilfe zu Subversion-Befehlen

```
> svn help
> svn help <Befehl>
```

- Auschecken eines Moduls aus einem Repositorium

```
> svn checkout <URL> <PATH>
```

- Status der lokalen Kopie

```
> svn status
```

- Update der lokalen Kopie auf den neusten Stand

```
> svn update
```


© 2004 Peter Thiemann, Matthias Neubauer

## 2.5 Wichtige Subversion-Befehle -2-

- (Lokales) Hinzufügen einer Datei/eines Verzeichnisses

```
> svn add <Datei>
```

- (Lokales) Umbenennen einer Datei/eines Verzeichnisses

```
> svn move <Datei> <Datei>
```

- (Lokales) Kopieren einer Datei/eines Verzeichnisses

```
> svn copy <Datei> <Datei>
```

- Commit der lokalen Änderungen in das Repositorium

```
> svn commit -m <Text> <Datei>
```

- Geschichte einer Datei

```
> svn log <Datei>
```


© 2004 Peter Thiemann, Matthias Neubauer

# 3 Entwickeln im Computer-Pool

- Checkout des Team-Moduls

  ```
  > svn checkout
  https://abacus.informatik.uni-freiburg.de/svn/proglang/sopra/teamXX
  ```

- Gemeinsames Entwickeln im Verzeichnis “`trunk`”
- Testen/Veröffentlichen von (statischen) HTML-Dateien:
  Commit in das Verzeichnis “`apache-docs`”
- Testen/Veröffentlichen von CGI-Programmen:
  Commit in das Verzeichnis “`apache-cgi`”
- Testen/Veröffentlichen von Java/JSPs:
  Commit in das Verzeichnis “`tomcat`”

 © 2004 Peter Thiemann, Matthias Neubauer

- Betrachten einer (statischen) XHTML-Datei:

  `http://pl.informatik.uni-freiburg.de/sopra/teamXX/yyy.html`

- Ausführen eines CGI-Programms:

  `http://pl.informatik.uni-freiburg.de/cgi/sopra/teamXX/yyy.cgi`

- Ausführen einer JSP:

  `http://abacus.informatik.uni-freiburg.de:8080/sopra/teamXX/yyy.jsp`


© 2004 Peter Thiemann, Matthias Neubauer