Verteilte Systeme
Hochschule Mannheim

VSY Team 5

Tim Braner, Jochen Gutermann, Steffen Hennhöfer, Sven Schönung, Sebastian Tschirpke

8.4.2011

Gliederung

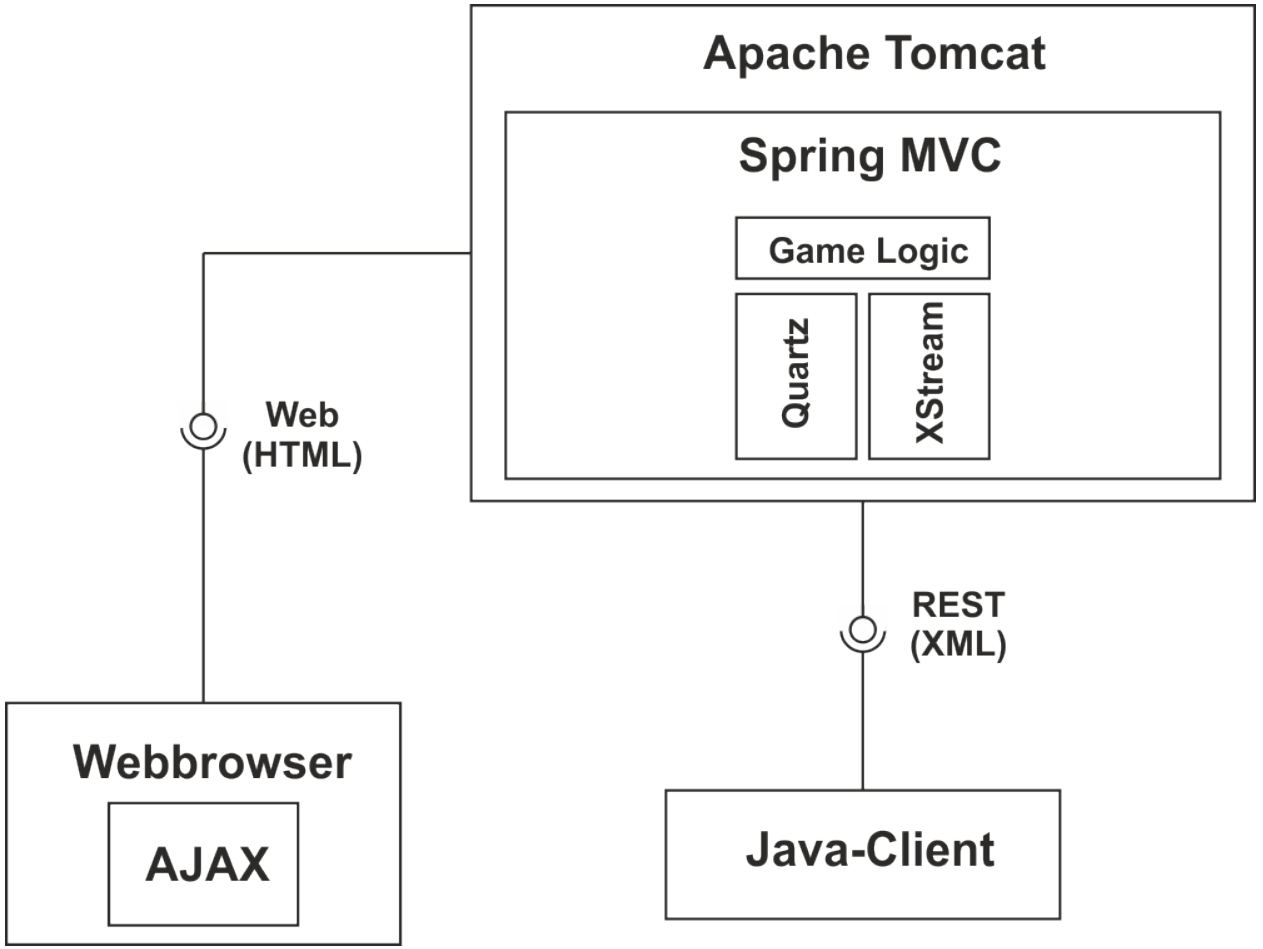
- Architektur
  - Vor- und Nachtteile
- Server
  - HTTP-Request
  - Rest-Interface
  - Konsequenzen und Probleme
- Clients

Architektur

Gliederung

- Architektur
  - Vor- und Nachtteile
- Server
  - HTTP-Request
  - Rest-Interface
  - Konsequenzen und Probleme
- Client
  - Vor- und Nachtteile

Architektur



Java

Vorteile

- Plattformunabhängig
- Automatische Speicherverwaltung
- Vielzahl an Open-Source libaries
- Einfache Entwicklung von Server Anwendung durch Servlet-API

Nachteile

- Nicht so performant wie C++
- keine Mehrfachvererbung

Client / Server

Warum Client Server?

Vorteile

- Zentrale Regelung des Systems
  - Konsistenz der Daten
- Einfach
- Hohe Performance

Nachteile

- Ausfallgefahr

Webinterface

Warum Webinterface?

Vorteile

- Keine eigene Software notwendig
- Vorkenntnisse nicht erforderlich

Nachteile

- Browserunabhängigkeit Aufwendig

REST-Interface

Warum REST-Interface?

Vorteile

- Simpel, einfache Entwicklung
- Nutzt vorhandenes HTTP-Protokoll
- erlaubt die Nutzung anderer Clients
- Skalierbarkeit der Dienste

Nachteile

- keine Standardisierung
- Schwierige Abbildung der Anwendungsfunktionalität auf Ressourcen

XML

Warum XML?

Vorteile

- Für Menschen verständlich/lesbar
- Eigene Datenstruktur definierbar
- Standardlibaries vorhanden z.B. zum Parsen

Nachteile

- Komplizierter als andere Formate wie z.B. Json

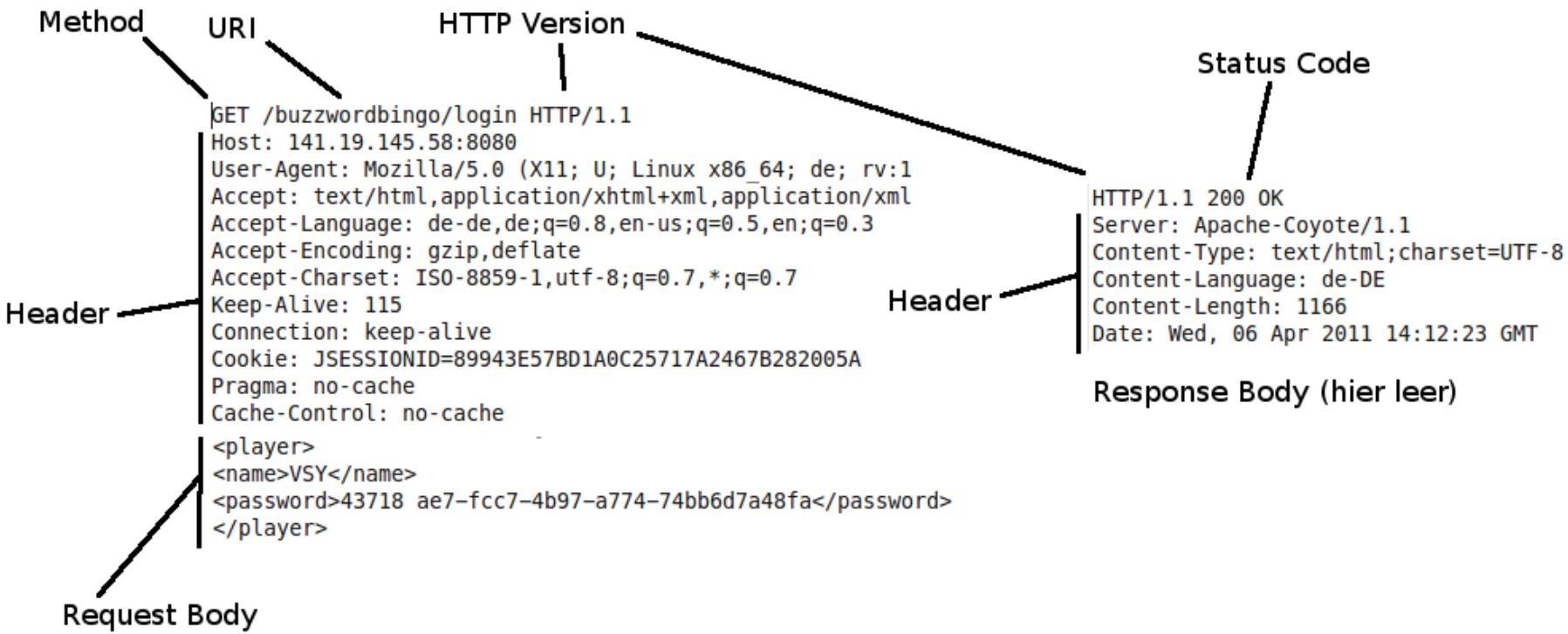
Server

REST Interface

- Wurde im Jahre 2000 von Roy Fielding definiert
- Mehr "Philosophie", kein formaler Standart
- basiert stark auf den Ideen von HTTP

REST Interface

Hypertext Transfer Protokol



REST Interface

Methoden operieren auf Ressourcen

Ressourcen

- Identifiziert durch URIs

Methoden

- GET (Eine Ressource abfragen)
- POST (Eine Ressource hinzufügen)
- PUT (Eine Ressource ersetzen)
- DELETE (Eine Ressource löschen)
- ...

REST Interface

Identifikation der Ressourcen
In unserem Fall:

- Spieler: `http://localhost:8080/buzzwordbingo/rest/players`
- Spiele: `http://localhost:8080/buzzwordbingo/rest/games`
- Boards: `http://localhost:8080/buzzwordbingo/rest/boards`

REST Interface

Repräsentation der Ressourcen: XML

```
GET /buzzwordbingo/rest/players HTTP/1.1
Host: 127.0.0.1:8080
...
```

```
<players from="0" to="99">
 <link rel="prev" href="/rest/players/?from=0&to=99" />
 <link rel="next" href="/rest/players/?from=100&to=199" />
 <player>
 <link rel="self" href="/rest/players/foo/" />
 <name>foo</name>
 <loginTime>1301662134113</loginTime>
 </player>
 <player>
 <link rel="self" href="/rest/players/bar/" />
 <name>bar</name>
 <loginTime>1301662133984</loginTime>
 </player>

</players>
```

REST Interface

HATEOAS

- Hypertext As The Engine Of Application State
- Clients sollen das REST Interface navigieren können wie ein Mensch das Web navigiert
- Entkopplung des Clients vom Server
- Nur eine URL braucht bekannt zu sein: Entry Point

REST Interface

Entry Point

In unserem Fall: http://localhost:8080/buzzwordbingo/rest/

```
<resources>
<link rel="self" href="/rest/" />
<games>
<link rel="self" href="/rest/games/" />
</games>
<players>
 <link rel="self" href="/rest/players/" />
 </players>
 <boards>
 <link rel="self" href="/rest/boards/" />
 </boards>

</resources>
```

REST Interface

REST API Definition

3.2.2 Erstellen eines Spielers

URL	`/rest/players/`
Method	`POST`
HTTP Auth	Nein

Body	`<player>`
	`  <name>foobar</name>`
	`  <password>43718ae7-fcc7-4b97-a774-74bb6d7a48fa</password>`
	`</player>`

Status Codes	`201 CREATED` Spieler erstellt `409 CONFLICT` Der Spieler existiert bereits
Header	`Location`

Problem: Wie kann verhindert werden dass ein Spieler auf Ressourcen eines anderen Spielers zugreift?
Antwort: Authentifizierung durch Passwort

Konsequenzen & Probleme

Passwort wird beim erstellen eines Spielers mitgegeben:
POST /rest/players
<player> <name>foo</name> <password>pass</password> </player>

Konsequenzen & Probleme

Geschützte Aktionen:

`POST /rest/games/` (Spiel erstellen)

`DELETE /rest/players/{name}` (Spieler ausloggen)

`POST /rest/players/{name}/games/{id}` (Spiel joinen)

`DELETE /rest/players/{name}/games/{id}` (Spiel verlassen)

`PUT /rest/boards/{id}` (Spielfeld ändern)

Konsequenzen & Probleme

HTTP Basic Authentication

- Client schickt Anfrage an Server
- Server meldet dass ein Passwort nötig ist
- Client sendet Request mit Passwort

```
DELETE /rest/players/foo HTTP/1.1
Host: 127.0.0.1:8080
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Konsequenzen & Probleme

Vorsicht

Übertragung praktisch unverschlüsselt

- Kein userdefiniertes Passwort verwenden (z.B. UUID)
- Gesamte Kommunikation über SSL laufen lassen

Konsequenzen & Probleme

Konsequenz: Clients müssen Server regelmäßig anfragen um Veränderungen mitzubekommen

- um zu erfahren ob ein Spieler bereits gewonnen hat
- um zu erfahren ob ein Spieler ein Spiel betreten oder verlassen hat

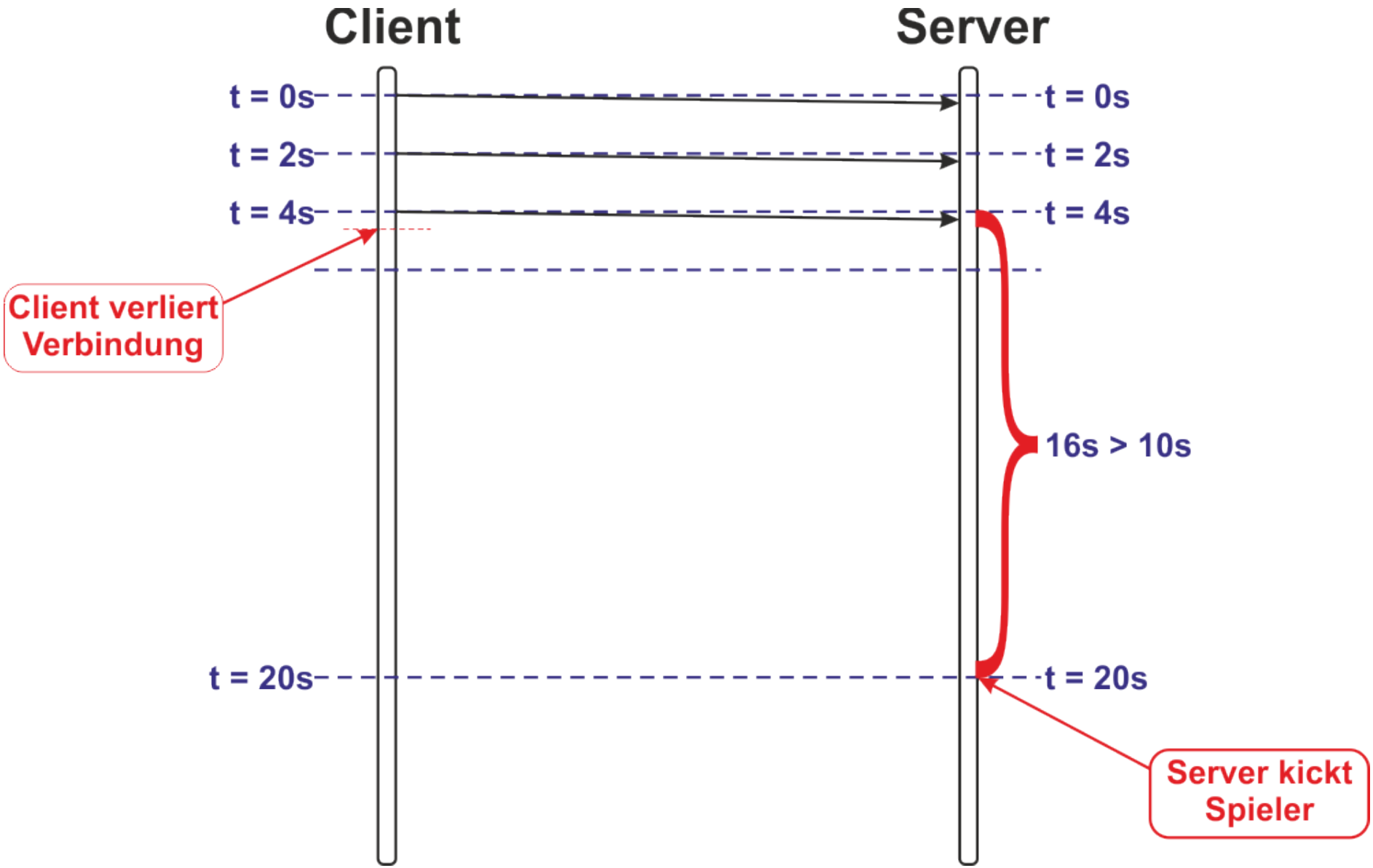
Konsequenzen & Probleme

Problem: Wie erfährt der Server ob ein Client überhaupt noch anwesend ist?

- Verbindung gekappt
- Client abgestürzt
- ...

Antwort: Anhand z.B. der Polls für die Spielerliste!

Konsequenzen & Probleme



Clients

Clients

Zwei Clients

- Browser-Client (AJAX)
- Java-Client

Clients

Webclient

- Oberfläche: HTML/CSS
- Asynchrones Javascript (mit jQuery)
  - Pollen der Spielerliste
  - Pollen des möglichen Gewinners
- Cookies zum Sessionmanagement

Clients

Javaclient

- Oberfläche: Swing
- Kommunikation über REST-Template
- XML zum Verarbeiten der Informationen