Buzzword-Bingo als Webapplikation mit REST Interface

Schriftliche Ausarbeitung der praktischen Übung in Verteilte Systeme an der HS-Mannheim im SS2011

Tim Braner
Jochen Gutermann
Steffen Hennhöfer
Sven Schönung
Sebastian Tschirpke

Abstract. Die vorliegende Arbeit beschäftigt sich mit der Entwicklung des Spiels "Buzzword Bingo" als Client-Server-System über HTTP. Die auf einem Tomcat-Server laufende Java-Applikation stellt dabei sowohl einen AJAX-basierten Webclient als auch ein von anderen Clients nutzbares REST Interface zur Verfügung. Ein dieses Interface nutzender Java-Client wurde ansatzweise entwickelt.

1 Einführung

Buzzword Bingo ist eine humorvolle Abänderung des bekannten Bingo Spiels: Statt der Zahlen stehen Worte auf dem Bingofeld und statt der zufälligen Ziehung und Bekanntgabe der Zahlen, die angekreuzt werden sollen, stammen die Worte von einem (meist ahnungslosen) Vortragenden. Ansonsten ist alles gleich: Gewonnen hat, wer zuerst eine vollständige Reihe, Spalte oder Diagonale auf seinem Spielfeld angekreuzt hat.

Im folgenden wollen wir eine Implementierung des Buzzword Bingos als verteiltes System vorstellen, die es mehreren Spielern erlaubt, Buzzword Bingo Spiele zu erstellen und an ihnen teilzunehmen.

Gewählt haben wir für unsere Implementierung eine *Client-Server-Topologie*, wobei der Server sowohl über ein *Webinterface* als auch eine *REST* Schnittstelle angesprochen werden kann. In beiden Fällen ist das verwendete Protokoll also *HTTP*.

Eine *Client-Server Topologie* bietet sich deshalb an, weil sie besonders einfach ist. Die Kommunikation geht immer vom Client aus. Der Server wartet lediglich auf Anfragen und beantwortet diese.

Die Vorteile eines *Webinterfaces* sind offensichtlich: Der Zugriff auf den Server kann von überall erfolgen. Webbrowser sind heute allgegenwärtig, weshalb die Installation spezieller Client-Software üblicherweise entfällt. Jeder, der einen

Email-Dienst nutzt, weiß wie vorteilhaft es ist, sein Postfach von überall aus lesen zu können.

REST ist ein vergleichsweise junges Konzept. Aufgrund seiner Einfachheit gegenüber anderen Ansätzen wie z.B. SOAP hat es jedoch schnell an Popularität gewonnen.

Begründet liegt dies nicht zuletzt in der konsequenten Nutzung von *HTTP*, das als Grundlage des Web ein wohlerprobtes, gut unterstütztes und gleichzeitig vielseitiges Kommunikations-Protokoll darstellt.

Unsere Ausarbeitung beginnt mit einer kurzen Erörterung der Problemstellung, wobei wir vor allem auf einige spezifische Probleme, die aus der Entscheidung für eine HTTP-basierte Client-Server-Topologie erwachsen, eingehen wollen.

Danach wenden wir uns der Implementierung des Servers zu: Wir stellen das von uns definierte REST Interface vor und unsere Lösungen der zuvor benannten Probleme.

Schließlich wollen wir am Ende der Arbeit ein kurzes Fazit ziehen.

2 Problemstellung

Wir wollen zunächst eine typische(?) Userinteraktion mit dem System betrachten und dann einige konkrete Probleme, die sich daraus für uns ergeben, erörtern.

Fred Foobar meldet sich mit einem Namen am Server an. Fred lässt sich die Liste aller auf dem Server laufenden Spiele anzeigen. Fred tritt einem Spiel bei. Fred markiert ein Wort. Fred bekommt die Meldung, dass Alice das Spiel gewonnen hat. Fred kehrt zur Liste aller Spiele zurück. Fred erstellt ein neues Spiel. Fred sieht, dass Alice dem Spiel beigetreten ist. Fred markiert ein Wort. Alice' Computer geht auf einmal aufgrund eines Stromausfalls aus. Fred sieht, dass Alice das Spiel verlassen hat.

Daraus ergeben sich unter anderem die folgenden Anforderungen für unser System:

2.1 Authentifikation

Spiele werden von einem Spieler eröffnet. Dieser Spieler ist der "Verwalter" des Spiels und hat unter Umständen besondere Rechte, die anderen Spielern nicht zur Verfügung stehen sollten. Gleichzeitig hat jeder Spieler sein eigenes Spielfeld, auf dem er Worte markiert und das natürlich nicht von anderen Spielern verändert werden kann. Wird die Funktionalität, ein Wort auf dem Spielfeld eines bestimmten Spielers zu markieren, aber über ein REST Interface zur Verfügung gestellt, kann diese prinzipiell von jedem Client auch genutzt werden. Um zu verhindern, dass ein Client das Spielfeld eines anderen Clients verändert, muss der Server sicher zwischen einem Spieler A und einem anderen Spieler B unterscheiden können. Es muss also - zumindest für bestimmte Aktionen - eine Möglichkeit

geben, sich am Server mittels eines geheimen Codes, z.B. eines usergewählten Passworts oder einer automatisch generierten ID, zu authentifizieren.

2.2 Notifikation der Clients

Jeder Client soll dem Benutzer eine Liste der aktuell an einem Spiel teilnehmenden Spieler zur Verfügung stellen. Tritt ein weiterer Spieler dem Spiel bei, so soll der Client dem Benutzer davon Rückmeldung geben. In einem Peer-to-peer Netzwerk ließe sich dies auf folgende Weise lösen: Jeder Peer agiert für ein von ihm gestartetes Spiel als Server. Stellt ein anderer Peer eine Verbindung zu ihm her um an einem Spiel teilzunehmen, benachrichtigt der Server-Peer jeden anderen an dem Spiel teilnehmenden Peer von dem Neuankömmling.

In der von uns gewählten Client-Server Kommunikation über HTTP ist ein solches Vorgehen jedoch nicht möglich: Der Server kann von sich aus keine Nachrichten an die einzelnen Clients schicken, sondern wartet immer nur auf Anfragen, die er bearbeitet und beantwortet. Der Client muss also auf anderem Wege an die Information kommen, welche Spieler gerade an einem Spiel teilnehmen.

2.3 Verbindungsabbruch eines Clients

Aus der Unfähigkeit des Servers, die Clients direkt zu kontaktieren, erwächst ein weiteres Problem: Woher weiß der Server, welche Clients überhaupt noch an einem Spiel teilnehmen? Die naive Lösung besteht darin, dass jeder Client den Server beim Verlassen eines Spiels benachrichtigt. Das ist sicher wünschenswert, jedoch nicht ausreichend. Was, wenn z.B. der Computer eines Spielers durch einen Stromausfall auf einmal aus geht? Oder wenn ein Spieler plötzlich keine Verbindung mehr zum Server herstellen kann, weil sein ISP die Leitung aufgrund nicht bezahlter Rechnungen gekappt hat? Oder das Betriebssystem, auf dem der Client läuft, auf einmal abstürzt? Der Server muss die Möglichkeit haben, solche Verbindungsabbrüche zu erkennen und den Client aus der Liste der teilnehmenden Spieler an einem Spiel zu entfernen.

Wir werden später in dieser Arbeit auf alle diese Punkte und die Lösungen, die wir gewählt haben, noch einmal zu sprechen kommen.

3 Server

Der Server soll in unserer Implementierung Informationen über die einzelnen Spiele zentral verwalten und den einzelnen Clients zur Verfügung stellen. Die Kommunikation soll dabei über ein REST Interface laufen.

3.1 REST

REST wurde im Jahre 2000 von Roy Fielding im Rahmen seiner Doktorarbeit entwickelt. Es handelt sich dabei nicht um einen formalen Standard, sondern eher um eine Design-Philsophie, die sich stark an den Ideen von HTTP orientiert, die dem World Wide Web zu einem solchen Erfolg verholfen haben.

Die zugrundeliegende Idee ist, dass der REST Server selbst stateless ist, das heißt, dass er keine Informationen über den Client speichert und jeder Client Request alle Informationen enthält, die zur Verarbeitung eines Requests nötig sind. Dies hat den Vorteil, dass der Server einfacht bleibt und die Komplexität in die Clients verlagert wird.

Um ein REST Interface zu definieren, müssen zunächst die Ressourcen identifiziert werden, auf die die Clients Zugriff haben sollen. In unserem Fall können wir drei verschiedene Ressourcen identfizieren: Spieler, Spiele und Spielfelder. Jeder Spieler kann dabei mehreren Spielen angehören, jedes Spiel mehrere Spieler enthalten und jedes Spielfeld gehört genau zu einer Kombination aus Spieler und Spiel.

Jede Ressource wird nun einer URL zugeordnet. So werden in unserem Fall Spieler über die URL $SERVER/buzzwordbingo/rest/players/{name} angesprochen. Auf diesen Ressourcen kann ein Client nun bestimmte Operationen durchführen. Diese Operationen werden durch das HTTP Protokoll definiert.

HTTP Anfragen haben generell die folgende Form:

```
POST /rest/players/ HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/buzzwordbingo.player+xml
```

POST im obigen Beispiel ist die HTTP Methode, die auf die Ressource `/rest/players/` angewand werden soll. `Host` und `Content-Type` sind HTTP Header, die zusätzliche Informationen über den Request mitliefern.

HTTP bietet verschiedene Methoden an von den die vier wichtigsten die folgenden sind:

- `GET` (Informationen über eine Ressource abholen)
- `POST` (Eine Ressource hinzufügen)
- `PUT` (Eine Ressource ändern)
- `DELETE` (Eine Ressource löschen)

Ressourcen müssen natürlich auf irgendeine Art und Weise respräsentiert werden. Übliche Arten der Repräsentation sind XML und JSON. Wir wollen im folgenden XML verwenden.

Hier beispielhaft das XML für die Liste aller Spieler, die bei einem GET Request auf /rest/players zurückgeliefert würde (Beispiel-XML für alle Ressourcen findet sich in Anhang A):

```
<players from="100" to="199">
  <link rel="self" href="/rest/players/?from=100&to=199" />
  <link rel="prev" href="/rest/players/?from=0&to=99" />
  <link rel="next" href="/rest/players/?from=200&to=299" />
  <player>
    <link ref="self" href="/rest/players/foo/" />
    <name>foo</name>
    <loginTime>1301662134113</loginTime>
  </player>
  [...]
</players>
```

Besondere Erwähnung sollte im obigen Beispiel die Verwendung der link Elemente finden. Eines der wichtigsten Design-Prinzipien eines REST-Interfaces ist HATEOAS: Hypertext As The Engine Of Application State. Dies bedeutet ganz einfach, dass ein Client ein REST Interface ebenso nutzen können sollte, wie ein Mensch das Web benutzt. Dieser kennt auch nicht die URL jeder Ressource auf die er zugreifen möchte, auswendig, sondern üblicherweise eine Addresse (z.B. http://reddit.com/) von der aus er sich über Links selbst seinen Weg zu anderen Ressourcen suchen kann (z.B. http://reddit.com/r/programming/). Der Vorteil für die REST-Clients liegt darin, dass sie nur einen Entry Point kennen müssen, d.h. insbesondere sind sie unabhängig von der konkreten URLs an denen die jeweiligen Ressourcen an denen sie interessiert sind, liegen (und die sich ändern können), was einer Entkoppelung des Clients vom Server dient.

Nach diesen eher allgemeinen Eläuterungen, im folgdenden nun eine komplette Übersicht über das von uns implementierte REST Interface:

Liste aller Ressourcen

URL	`/rest/`
Method	`GET`
HTTP Auth	Nein
Status Codes	`200 OK` Erfolgreich
Returns	`application/buzzwordbingo.resources+xml`

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`

Beim Erstellen eines Spieler werden vom Client alle notwendigen Informationen (Name und Passwort) übertragen. Mit der Übertragung findet eine Überprüfung statt, ob bereits ein Spieler mit dem gleichen Namen existiert, da festgelegt ist, dass der Spielername 'unique' (einzigartig) ist. Gibt es keinen Konflikt mit anderen Spieler wird ein Spielerobjekt erstellt und der Liste der verbundenen Spieler, hinzugefügt. Ist der Spieler erfolgreich hinzugefügt worden, sendet der Server den passenden Statuscode zurück an den Client. Tritt bei der Überprüfung ein Namenskonflikt auf, bricht der Server die Interaktion ab und meldet dem Client den entsprechenden Statuscode.

Liste aller Spieler

URL	`/rest/players/`	
Querystring	`sort=[name|login]`	Sortierung festlegen
	`order=[asc|desc]`	Reihenfolge festlegen
	`from=`i	Startindex
	`to=`j	Endindex
	`game=`id	Filtern nach Spiel
Method	`GET`	
HTTP Auth	Nein	
Status Codes	`200 OK` Erfolgreich	
Returns	`application/buzzwordbingo.players+xml`	

Mit dem HTTP-Request an den Server, fügt der Server sämtliche Spieler, welche mit dem Server verbunden sind, in ein XML-Dokument ein und überträgt dieses an den Client. Hierbei können diverse optionale Parameter übergeben werden. So kann z.B. eine Sortierung oder ein Filter festgelegt werden oder bestimmte Indizes direkt angesprochen werden. Ist das XML-Dokument mit den passenden Ergebnissen übertragen worden, sendet der Server den entsprechenden Statuscode an den Client.

Informationen zu einem Spieler

URL	`/rest/players/`*name*`/`
Method	`GET`
HTTP Auth	Nein
Status Codes	`200 OK` Erfolgreich `404 NOT FOUND` Spieler existiert nicht
Returns	`application/buzzwordbingo.player+xml`

Mit der Anfrage zu einem bestimmten Spieler an den Server, übergibt der Client dem Server einen bestimmten Spielernamen. Da der Spielername als 'einzigartig' definiert ist, kann dem Namen genau ein Spieler zugeordnet werden, falls ein Spieler mit entsprechendem Namen mit dem Server verbunden ist. Ist ein solcher Spieler vorhanden, fügt der Server alle Informationen des Spielers in ein XML-Dokument ein und überträgt dieses zum anfragenden Client. Ist die Übertragung dieses XML-Dokumentes abgeschlossen oder wird kein passender Spieler gefunden, sendet der Server den entsprechenden Statuscode an den Client.

Entfernen eines Spielers

URL	`/rest/players/`*name*`/`
Method	`DELETE`
HTTP Auth	Ja
Status Codes	`204 NO CONTENT` Spieler entfernt `403 FORBIDDEN` Keine Berechtigung

Will der Client einen bestimmten Spieler auf dem Server löschen, was zB. beim Trennen der Verbindung geschieht, muss der Client zusätzlich zum Namen des zu löschenden Spielers auch dessen codiertes Passwort übertragen. Dies stellt sicher, dass fremde Clients keine Spieler vom Server trennen kann. Beim Löschen des Spielers wird zunächst überprüft ob ein Spieler mit passendem Namen vorhanden ist. Ist der entsprechende Spieler identifiziert worden, gleicht der Server das übertragene Passwort des Clients mit dem eingetragenen Passwort des Spielers ab. Stimmen diese überein wird der Spieler vom Server getrennt. Hierbei wird der Spieler aus allen aktiven Spielen entfernt und alle seine Spielfelder werden gelöscht. Anschließend wird er aus der Liste der eingeloggten Spieler entfernt und seine Session zum Server wird beendet. Ist dieser Vorgang abgeschlossen oder hat das übertragene und das eingetragene Passwort nicht übereingestimmt, sendet der Server den entsprechenden Statuscode an den Client.

Erstellen eines Spiels

URL	`/rest/games/`
Method	`POST`
HTTP Auth	Ja

Body	`<game>`
	`  <title>Titel des zu erstellenden Spiels</title>`
	`  <size>3</size>`
	`  <description>Eine kurze Beschreibung des Spiels (optional)</description>`
	`  <buzzwords>`
	`  <buzzword>Ajax</buzzword>`
	`  <buzzword>Java</buzzword>`
	`  <buzzword>REST</buzzword>`
	`  <buzzword>XML</buzzword>`
	`  <buzzword>Tomcat</buzzword>`
	`  <buzzword>Spring MVC</buzzword>`
	`  <buzzword>HATEOAS</buzzword>`
	`  [...]`
	`  </buzzwords>`
	`</game>`

Status Codes	`201 CREATED` Spiel erstellt `403 FORBIDDEN` Keine Berechtigung für diesen Spieler
Header	`Location`

Beim Erstellen eines Spiels sendet der Client ein XML-Dokument an den Server welches wie oben beschrieben, formatiert ist. Es muss alle notwendigen Informationen zum erstellen des Spiels enthalten. Das Erstellen eines Spiels erfordert ebenfalls, dass sich der Client über einen verbundenen Spieler authentifiziert. Mit dem Erhalt der XML-formatierten Informationen filtert der Server die Daten aus dem Dokument und erstellt ein entsprechendes Spiel. Der authentifizierte Client wird dann automatisch als 'Owner' des Spiels eingetragen. Ist das Erstellen des Spiels erfolgt oder konnte sich der Client nicht authentifizieren, sendet der Server den entsprechenden Statuscode an den Client.

Liste aller Spiele

URL	`/rest/games/`	
Querystring	`sort=[title|creation|size]`	Sortierung festlegen
	`order=[asc|desc]`	Reihenfolge festlegen
	`from=`*`i`*	Startindex
	`to=`*`j`*	Endindex
	`player=`*`id`*	Filtern nach Spieler
Method	`GET`	
HTTP Auth	Nein	
Status Codes	`200 OK` Erfolgreich	
Returns	`application/buzzwordbingo.games+xml`	

Ähnlich der Abfrage aller Spieler, kann der Client alle aktuell laufenden Spiele vom Server abfragen. Nach dem Senden der Anfrage überträgt der Server sämtliche gelisteten Spiele in ein XML-Dokument und überträgt dieses zum anfragenden Client. Es werden ebenfalls weitere optionale Parameter unterstützt, welche eine Eingrenzung und/oder Sortierung der Ergebnisse zur Folge haben. Wurde das XML-Dokument erfolgreich übertragen, sendet der Server den entsprechenden Statuscode an den Client.

Informationen zu einem Spiel

URL	`/rest/players/`*`id`*`/`
Method	`GET`
HTTP Auth	Nein
Status Codes	`200 OK` Erfolgreich `404 NOT FOUND` Spiel existiert nicht
Returns	`application/buzzwordbingo.game+xml`

Mit der Anfrage zu einem bestimmten Spiel an den Server, übergibt der Client dem Server eine bestimmte Spiel-ID. Da der Spiel-ID als 'einzigartig' definiert ist, kann der ID genau ein Spiel zugeordnet werden, falls ein Spiel mit entsprechender ID auf dem Server aktiv ist. Ist ein solches Spiel vorhanden, fügt der Server alle Informationen des Spiels in ein XML-Dokument ein und überträgt dieses zum anfragenden Client. Ist die Übertragung dieses XML-Dokumentes abgeschlossen oder wird kein passendes Spiel gefunden, sendet der Server den entsprechenden Statuscode an den Client.

Hinzufügen eines Spielers zu einem Spiel

URL	`/rest/games/`*`id`*`/players/`
Method	`POST`
HTTP Auth	Ja

Body	`<player>`
	`  <name>foo</name>`
	`</player>`

Status Codes	`201 CREATED` Spieler beigetreten `404 NOT FOUND` Das Spiel existiert nicht `403 FORBIDDEN` Keine Berechtigung für den Spieler `409 CONFLICT` Spieler ist dem Spiel bereits beigetreten

Um einen Spieler einem bereits vorhandenen Spiel hizuzufügen muss der Client dem Server zuzüglich zur Authentifizierung, auch Informationen zum betroffenen Spiel und dem Spieler, welcher das Spiel beitreten möchte, übertragen. Hierbei ist zu beachten, dass die Informationen des betroffenen Spiels über angesprochene URL aufgelöst, und nur Spielerinformationen in einem XML-Dkoment übertragen werden. Hat der Server die Informationen erhalten, wird geprüft

ob das angesprochene Spiel existiert und ob der Spieler diesem nicht bereits beigetreten ist. Ist das Spiel vorhanden und der Spieler dem Spiel nicht bereits beigetreten, fügt der Server den Spieler dem Spiel hinzu und legt hierfür ein individuelles Spielbrett an. Ist dieser Vorgang abgeschlossen, wurde das Spiel nicht gefunden, hatte der Client keine Berechtigung oder war der Spieler bereits dem Spiel beigetreten, sendet der Server den entsprechenden Statuscode an den Client.

Entfernen eines Spielers aus einem Spiel

URL	`/rest/games/`*id*`/players/`*name*	
Method	`DELETE`	
HTTP Auth	Ja	
Status Codes	`204 NO CONTENT`	Spieler aus Spiel entfernt
	`404 NOT FOUND`	Das Spiel existiert nicht/Spieler nicht im Spiel
	`403 FORBIDDEN`	Keine Berechtigung für den Spieler

Zum Entfernen eines verbundenen Spielers aus einem einem vorhandenen Spiel muss sich der Client sowohl beim Server authentifizieren, als auch Informationen über das betroffene Spiel und den Spielernamen übermitteln. Hierbei gilt zu beachten, dass sowohl Spiel als auch Spieler über die angesprochene URL aufgelöst werden. Ist das Spiel vorhanden und befindet sich der gewählte Spieler in diesem Spiel, entfernt der Server den Spieler aus dem Spiel, dessen individuelles Spielfeld wird jedoch weiterhin gespeichert, damit der Spieler dem Spiel evtl. zu einem späteren Zeitpunkt wieder beitreten kann. Ist dieser Vorgang abgeschlossen, das Spiel existiert nicht, ein solcher Spieler befindet sich nicht im Spiel oder hat der Client keine Berechtigungen für diesen Spieler, sendet der Server den entsprechenden Statuscode an den Client.

Liste aller Boards

URL	`/rest/boards/`	
Querystring	`from=`*i*	Startindex
	`to=`*j*	Endindex
	`player=`*name*	Filtern nach Spieler
	`game=`*id*	Filtern nach Spiel
Method	`GET`	
HTTP Auth	Nein	
Status Codes	`200 OK` Erfolgreich	
Returns	`application/buzzwordbingo.boards+xml`	

Ähnlich der Abfrage aller Spiele, kann der Client eine Liste aller Spielfelder vom Server anfordern. Hierbei iteriert der Server über alle aktiven Spiele und fügt alle Spielfelder in ein XML-Dokument, welches anschließend an den Client übertragen wird. Es werden zusätzliche optionale Parameter unterstützt, welche

eine Eingrenzung und/oder Sortierung der Ergebnisse zur Folge haben. Ist dieser Vorgang abgeschlossen, sendet der Server den entsprechenden Statuscode an den Client.

Informationen zu einem Board

URL	`/rest/boards/id/`
Method	`GET`
HTTP Auth	Nein
Status Codes	`200 OK` Erfolgreich `404 NOT FOUND` Board existiert nicht
Returns	`application/buzzwordbingo.board+xml`

Ändern eines Boards

URL	`/rest/boards/id/`
Method	`PUT`
HTTP Auth	Ja

Body	`<board>`
	`  <buzzwords>`
	`  <buzzword>`
	`  <row>0</row>`
	`  <col>0</col>`
	`  <text>foo</text>`
	`  <marked>true<marked>`
	`  </buzzword>`
	`  [...]`
	`  <buzzword>`
	`  <row>4</row>`
	`  <col>4</col>`
	`  <text>bar</text>`
	`  <marked>false</marked>`
	`  </buzzword>`
	`  </buzzwords>`
	`</board>`

Status Codes	`204 NO CONTENT` Board geändert `404 NOT FOUND` Das Board existiert nicht

4 Problemlösungen

Nachdem unser REST Interface nun definiert ist, wollen wir uns den in Abschnitt 2 angesprochenen Problemen zu wenden und hier kurz unsere Lösungen vorstellen.

4.1 Authentifikation

Um zu verhindern, dass ein Spieler auf Ressourcen eines anderen Spielers zugreifen kann, ist eine Authentifizierung vonnöten. Dies geschieht in unserer Implementierung durch ein Passwort, welches beim Erstellen eines Spielers dem entsprechenden POST-Request mitgegeben wird. Mithilfe von HTTP Basic Authentication sind alle möglichen Aktionen eines eingeloggten Spielers passwortgeschützt: Spieler ausloggen, Spiel anlegen, Spiel joinen, Spiel verlassen, Spielfeld ändern. Will ein Client eine dieser Aktionen ausführen, wird er auf seine Anfrage hin vom Server nach einem Passwort gefragt. Erst wenn er einen Request mit dem korrekten Passwort sendet, wird die Aktion ausgeführt.

Da bei HTTP Basic Authentication Benutzername und Passwort zwar codiert, jedoch nicht verschlüsselt werden, ist die Übertragung zuerst unsicher. Wählt der Benutzer nun ein eigenes Passwort, ohne dass es zusätzlich verschlüsselt wird, besteht die Gefahr, dass er beispielsweise sein Email-Passwort einer unverschlüsselten Verbindung preisgibt. Um dies zu verhindern, gibt es zwei Lösungswege. Die aufwändige Variante wäre, die gesamte Kommunikation via SSL zu verschlüsseln. Aufgrund der geringen Sicherheitsanforderungen an das Buzzword Bingo und der simplen Durchführbarkeit entschieden wir uns jedoch dafür, den Benutzer gar nicht erst zur Passworteingabe aufzufordern, sondern ihm im Client ein automatisch generiertes Passwort zuzuweisen. Das Passwort fungiert also praktisch als eine Art Session-ID.

4.2 Notifikation der Clients & Verbindungsabbruch eines Clients

Um beispielweise die aktuelle Spielerliste anzuzeigen, müssen die Clients ständig darüber informiert sein, wer gerade am Spiel teilnimmt. Dazu müssen sie erfahren, wenn ein Spieler ein Spiel betreten, verlassen oder bereits gewonnen hat. Dies geschieht durch regelmäßige Anfragen beim Server (Polls), die alle zwei Sekunden erfolgen. Der Server antwortet mit einer Liste aller an einem Spiel teilnehmenden Spieler. Dabei kann man jedoch nicht davon ausgehen, dass der Client den Server beim Verlassen des Spiels zuverlässig benachrichtigt. Wird die Verbindung unerwartet abgebrochen oder stürzt der Client ab, muss der Server ebenfalls davon in Kenntnis gesetzt werden, dass der Client nicht mehr anwesend ist. Daher überprüft der Server in regelmäßigen Zeitabständen von zwanzig Sekunden die Zeit des letzten Polls des jeweiligen Clients. Ist diese größer als 10 Sekunden, so hat der Spieler das Spiel verlassen und wird entsprechend aus der Spielerliste entfernt.

5 Clients

Es wurden von uns beispielhaft zwei Clients implementiert, die den vom Server angebotenen REST Service in Anspruch nehmen: 1. Ein Webclient, der im

Browser bedient werden kann. 2. Eine stand-alone Java-Applikation, die auf jedem Java fähigen System läuft.

5.1 Webclient

Für den Browserclient standen zwei Alternativen zur Auswahl: 1. Ein vollständig in AJAX implementierter Client, bei dem alle Kommunikation über das REST-Interface geht. 2. Eine Mischlösung, bei der HTML Seiten auf dem Server gerendert werden und lediglich das Nötigste über asynchrones JavaScript läuft.

Im ersten Fall würde lediglich zu Beginn eine hauptsächlich JavaScript beinhaltende Seite geladen. Alle weiteren Requests würden dann im Hintergrund an das REST-Interface des Servers gehen. Das zurückgegebene XML würde auf Client-Seite geparst und das im Browser dargestellte HTML direkt über den im Speicher liegenden DOM-Baum modifiziert. Dies hat den Vorteil, dass der Seitenaufbau auf Client-Seite lediglich einmal, nämlich zu Beginn, zu erfolgen hätte. Zudem wäre der Client vollständig unabhängig vom Server, der lediglich die Aufgabe hätte den Client einmalig zu übertragen.

Im zweiten Fall übernimmt der Server nicht nur das initiale Versenden des Clientcodes an den Browser, sondern stellt diesem auch von ihm selbst anhand der Daten produzierte HTML-Seiten zur Verfügung. Dies erhöht die Komplexität des Server, macht jedoch die Implementierung des clientseitigen Codes wesentlich einfacher, weil das Parsen des XMLs großenteils entfällt. Zudem ist die Implementierung auf Serverseite einfacher: Die Daten sind hier direkt verfügbar und müssen nicht erst wie auf Clientseite deserialisiert werden.

Beide Lösungen haben ihre Vor- und Nachteile. Die zweite Option schien uns jedoch ein für unsere Zwecken geeigneter Mittelweg zwischen Einfachheit und Funktionalität.

5.2 Java-Client

Anders als beim Webclient vollzieht sich beim Java-Client die vollständige Kommunikation über das REST-Interface des Servers.

6 Fazit

Prinzipiell lässt sich sagen, dass die von uns gewählte Client-Server Topologie mit HTTP eine recht gute Wahl für die gestellte Aufgabe war. Die Implementierung des Servers war relativ einfach, zum einen durch das Vorhandensein von spezifisch auf diese Technologie zugeschnittener Softwarelösungen (Tomcat und das verwendete Spring Framework), zum anderen aufgrund der Einfachheit des Protokolls selbst.

Einziger Nachteil ist die Ineffizienz der Kommunikation: Die große Mehrzahl der Anfragen des Clients ist unnötig und dient nur dazu herauszufinden ob sich am gegenwärtigen Zustand des Spiels etwas verändert hat. Eine Lösung bei der die Clients über Veränderungen notifiziert werden, Kommunikation also nur dann stattfindet, wenn es auch wirklich nötig ist, wäre effizienter gewesen.

Etwas problemantisch war die Wahl von XML zur Datenrepräsentation. Eine einfacheres Datenformat, wie z.B. JSON, hätte den Anforderungen wohl ebenfalls genügt und zudem den Vorteil eines geringen Datenvolumns bei der Übertragung gehabt.

A Medientypen

A.1 application/buzzwordbingo.resources+xml

```
<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>
```

A.2 application/buzzwordbingo.players+xml

```
<players from="100" to="199">
  <link rel="self" href="/rest/players/?from=100&to=199" />
  <link rel="prev" href="/rest/players/?from=0&to=99" />
  <link rel="next" href="/rest/players/?from=200&to=299" />
  <player>
    <link ref="self" href="/rest/players/foo/" />
    <name>foo</name>
    <loginTime>1301662134113</loginTime>
  </player>
  [...]
</players>
```

A.3 application/buzzwordbingo.player+xml

```
<player>
    <link rel="self" href="/rest/players/foo/" />
    <name>foo</name>
    <loginTime>1301662134113</loginTime>
    <games>
      <game>
        <link ref="self" href="/rest/games/Spiel_1/">
```

```
        <id>Spiel_1</id>
        <title>Spiel 1</title>
        <creationTime>1301662198434</creationTime>
      </game>
      [...]
    </games>
    <boards>
      <board>
        <link ref="self" href="/rest/boards/d102c080-5c5c-11e0-80e3-0800200c9a66/" />
      </board>
      [...]
    </boards>
</player>
```

A.4 application/buzzwordbingo.games+xml

```
<games from="100" to="199">
  <link rel="self" href="/rest/games/?from=100&to=199" />
  <link rel="prev" href="/rest/games/?from=0&to=99" />
  <link rel="next" href="/rest/games/?from=200&to=299" />
  <game>
    <link ref="self" href="/rest/games/Spiel_1/" />
    <id>Spiel_1</id>
    <title>Spiel 1</title>
    <creationTime>1301662198434</creationTime>
    <owner>
      <link ref="self" href="/rest/players/foo/" />
      <name>foo</name>
    </owner>
    <winner>
      <link rel="self" href="/rest/players/bar/" />
      <name>bar</name>
    </winner>
  </game>
</games>
```

A.5 application/buzzwordbingo.game+xml

```
<game>
  <link rel="self" href="/rest/games/Spiel_1">
  <id>Spiel_1</id>
  <title>Spiel 1</title>
  <size>5</size>
  <description>Kurze Beschreibung</description>
  <creationTime>1301662198434<creationTime>
  <owner>
    <link rel="self" href="/rest/players/foo/" />
    <name>foo</name>
  </owner>
  <winner>
    <link rel="self" href="/rest/players/bar/" />
    <name>bar</name>
  </winner>
  <buzzwords>
    <buzzword>Ajax</buzzword>
```

```
    <buzzword>Java</buzzword>
    <buzzword>REST</buzzword>
    <buzzword>XML</buzzword>
    <buzzword>Tomcat</buzzword>
    <buzzword>Spring MVC</buzzword>
    <buzzword>HATEOAS</buzzword>
    [...]
  </buzzwords>
  <players>
    <player>
      <link ref="self" href="/rest/players/bar/" />
      <name>bar</name>
    </player>
    <player>
      <link ref="self" href="/rest/players/quux/" />
      <name>quux</name>
    </player>
    [...]
  </players>
  <boards>
    <board>
      <link rel="self" href="/rest/players/d102c080-5c5c-11e0-80e3-0800200c9a66/" />
    </board>
    <board>
      <link rel="self" href="/rest/players/550e8400-e29b-41d4-a716-446655440000/" />
    </board>
    [...]
  </boards>
</game>
```

A.6 application/buzzwordbingo.boards+xml

```
<boards from="100" to="199">
  <link rel="self" href="/rest/boards/?from=100&to=199" />
  <link rel="prev" href="/rest/boards/?from=0&to=99" />
  <link rel="next" href="/rest/boards/?from=200&to=299" />
  <board>
    <link ref="self" href="/rest/boards/d102c080-5c5c-11e0-80e3-0800200c9a66/" />
    <game>
      <link rel="self" href="/rest/game/foo/" />
      <id>foo</id>
    </game>
    <player>
      <link ref="self" href="/rest/player/bar/" />
      <name>bar</name>
    </player>
  </board>
  [...]
</boards>
```

A.7 application/buzzwordbingo.board+xml

```
<board>
  <link rel="self" href="/rest/boards/d102c080-5c5c-11e0-80e3-0800200c9a66/" />
  <size>5</size>
```

```
  <game>
    <link rel="self" href="/rest/games/Spiel_1" />
  </game>
  <player>
    <link rel="self" href="/rest/players/foo/" />
  </player>
  <buzzwords>
    <buzzword>
      <row>0</row>
      <col>0</col>
      <text>XML</text>
      <marked>true</marked>
    </buzzword>
    [...]
    <buzzword>
      <row>4</row>
      <col>4</col>
      <text>Ajax</text>
      <marked>false</marked>
    </buzzword>
  </buzzwords>
</board>
```