Fachhochschule Köln - Campus Gummersbach

Web-basierte Anwendungen 2

Logbuch

ColourConnection

Jorge Henrique Ferreira Pereira

Sommersemester 2013

Fachsemester: 4

Inhaltsverzeichnis

1 2013-04-15 | Kickoff: Ideen- / Problemfindung 2
2 2013-04-22 | Konzeptioneller Meilensein - Kommunikationsabläufe und Interaktionen 2
3 2013-04-27 | Benutzerauthentifizierung bezogen auf ein RESTful Webservice 2
4 2013-04-29 | Ressourcen und URI Design 3
5 2013-05-04 | Ressourcen-Identifizierung, Ausarbeitung der Semantik und Umsetzung eines XML-Schema 4
6 2013-05-06 | Meilenstein 1 + 2 - XML Schema, Ressourcen und die Semantik der HTTP-Operationen 4
7 2013-05-11 | Umsetzung des RESTful Webservice 5
8 2013-05-13 | Meilenstein 3 - RESTful Webservice 6
9 2013-05-18 | Openfire lokal eingerichtet / Einarbeitung in die Smack-API 6
10 2013-04-26 | Benutzerauthentifizierung bezogen auf XMPP 6
11 2013-06-01 | XMPP und XML-Stanzas 7
12 2013-06-03 | Meilenstein 4 + 5 - Konzeption asynchrone Kommunikation + XMPP – Client 7
13 2013-06-17 | Fertigstellung des finalen Clients 8

1 2013-04-15 | Kickoff: Ideen- / Problemfindung

Es stellte sich heraus, dass es nicht so leicht war zu einer guten und interessanten Idee zu kommen. Viele Ideen mussten verworfen werden, weil sie schon von anderen Gruppen aufgegriffen wurden. Es sehr spät ergab sich die Überlegung, angelehnt an die Seite ColourLovers, eine kleine Plattform zu realisieren, wo Farben den Mittelpunkt darstellten. Alles soll sich um Farben drehen. Farben sollen geteilt und aufgegriffen werden, um neue Erzeugnisse zu kreieren. Dies soll zu einer großen Variation an Farbpaletten führen, die für eventuelle Medienerzeugnisse herangezogen werden können. Darunter fallen Farbschemen, an die sich orientiert werden kann, wenn z.B. eine gute Farbkombination für die Gestaltung einer Webseite benötigt wird. Es soll auch als Anlaufstelle dienen, um sich von frischen und dem Trend folgenden Farbkombinationen inspirieren zu lassen.

2 2013-04-22 | Konzeptioneller Meilensein - Kommunikationsabläufe und Interaktionen

Bevor es zur eigentlichen Umsetzung des Projekts kommt, sollte man sich im klaren sein, wie die Kommunikations- sowie Interaktionsabläufe aufgebaut sein sollten. Dies ist empfehlenswert, um schon im Vorfeld eventuell auftretende Probleme sowie Unstimmigkeiten identifizieren zu können. Es wurde sich dazu entschlossen kleine simple Szenarien zu verfassen, die die wichtigsten Kommunikationsabläufe verdeutlichen sollen. Darunter fielen die Schritte zum Erstellen eines Farbressource, dem Festlegen einer Lieblingsfarbe, aber auch der Vorgang des Abonnierens einer bestimmten Farbe. Speziell bei Letzterem kann parallel zueinander ein synchroner und asynchroner Kommunikationsablauf festgestellt werden. Solch ein Ablauf ist nicht nur beim Setzen einer Lieblingsfarbe relevant, sondern auch beim "Folgen" eines Benutzers und seiner Erzeugnisse. Der asynchrone Aspekt ist dadurch gegeben, dass ein Benutzer nicht immer wieder selbst überprüfen muss, ob ein anderer Benutzer, den er folgt, neue Farbpaletten erzeugt hat, sondern eine entsprechende Benachrichtigung erhält. Hier muss eins dieser Erzeugnisse nicht einmal eine der eigenen Lieblingsfarben verwenden. Dies kann somit als Möglichkeit dienen, neue interessante Farben zu finden, die dann evtl. als Lieblingsfarben festgelegt werden können. Sonst wäre man nur auf die zusätzlichen Farben einer Farbpalette beschränkt, die unter anderem eine Farbe verwendet zu der man angegeben hat, dass sie die Lieblingsfarbe von einem selbst ist.

3 2013-04-27 | Benutzerauthentifizierung bezogen auf ein RESTful Webservice

Als ein sehr interessantes Thema bezüglich RESTful Webservices empfinde ich die Möglichkeiten der Benutzerauthentifizierung. Bezogen auf das Projekt der Phase2, könnte es relevant sein, eine Benutzerauthentifizierungsschnittstelle einzuführen, um die Rechte eines Benutzers

konkret zu bestimmen. Obwohl solche Authentifizierungs-Mechanismen in den Vorlesungen noch nicht behandelt wurden, war das Interesse so groß, dass ich nach solchen auf eigene Faust recherchiert habe. Generell wird immer gesagt, dass Kommunikationsabläufe die einen Benutzer authentifizieren, über SSL/TLS (Secure Socket Layer bzw. Transport Layer Security) laufen sollten. Somit würde man hier direkt auf HTTPS setzen. Dies erlaubt es sensible Daten (Passwörter), die normalerweise im quasi Klartext übertragen werden, geschützt zu übertragen. Der Einsatz von SSL/TLS würde das Risiko extrem minimieren, Opfer einer `Man-In-The-Middle`-Attacke zu werden. Als Beispiel für eine Authentifizierung per Klartext (angenommen, dass SSL/TLS nicht eingesetzt wird), könnte das "Basic Authentication"-Verfahren genannt werden, welches direkter Bestandteil des `Hyptertext Transfer Protocol` ist. Hier werden Benutzername und Passwort mittels eines `Authorization-Headers` im Klartext übertragen. Das Mitlesen dieser sensiblen Daten ist sehr leicht möglich. Mit dem Einsatz von SSL/TLS kann aber eine sichere Übertragung durch eine schützende Kapselung der Daten gewährleistet werden. Als Alternative zum "Basic Authentication"-Verfahren, bietet sich das "Digest Access Authenticaon"-Verfahren an. Hier werden nie wirklich sensible Daten verschickt, es wird anstatt dessen ein Hashcode versendet, welcher zuvor mittels einer Hashfunktion, angewandt auf die sensiblen Daten sowie weiteren vom Server gegebenen Daten, errechnet wurde. Die Gegenstelle muss demnach den Hashcode dann nur noch mit einem selbst berechneten Hashcode auf Übereinstimmung prüfen. Bevor ich mich für eines dieser Verfahren entscheide, und anhand dieser Entscheidung das Projekt für Phase 2 weiter ausarbeite, werde ich warten, bis das Thema Authentifizierung in einer Vorlesung behandelt wird und auch bekannt gegeben wird, welches Verfahren letztendlich verwendet werden soll. Evtl. können wir auch frei entscheiden, auf welches Verfahren wir schlussendlich setzen.

4 2013-04-29 | Ressourcen und URI Design

Ein wichtiger Punkt, welcher in Phase 2 näher betrachtet wird und sicherlich auch ausschlaggebend für die Qualität einer API, ist das Design der URI als Identifizierungsmechanismus für Ressourcen. Nach einigen Recherchen bezüglich dem Entwickeln einer guten und sauberen Semantik, kristallisiert sich eine klare Faustregel, an die man sich idealerweise richten sollte. Nomen sind gut und Verben sind schlecht. Konkret bedeutet dies, dass Ressourcen über Nomen adressiert werden. Sie repräsentieren ja eine Entität bzw. eine Liste von Entitäten. Als Verben setzen man nur die HTTP-Verben `GET`, `PUT`, `POST` und `DELETE` ein, die für grundlegende Operationen völlig ausreichen. Für ColourConnection würde eine Beispiel-URI evtl. wie folgt aussehen:

```
/user/42
/color/333333
/user/followers
/user/followers/42
/colorpalettes
/colorpalette/815
/user/favourite/colours
```

Wie zu sehen ist, werden in keinster Weise Verben verwendet, sondern explizit nur Nomen. Ein `GET /user/42` würde man grob mit "hole mir die Ressource vom Benutzer mit der ID 42" übersetzen. Obwohl `PUT` und `POST` eine ähnliche Operation suggerieren, unterscheiden sie sich doch in einem kleinen Detail. Der `PUT`-Verb wird verwendet, wenn man eine neue Ressource erstellen bzw. eine bereits existierende Ressource aktualisieren will. Der `POST`-Verb hingegen setzt man ein, wenn man eine untergeordnete Ressource erstellen will. Evtl. ein neues Element einer Liste.
Als Beispiel:
`POST /colorpalettes` würde eine untergeordnete Ressource erstellen. In diesem Fall eine Ressource für eine neue Farbpalette. Ob ich die Erstellung einer neuen Ressource wirklich so gestalten werde, muss ich noch entscheiden, denn es würde sich auch folgender Requestaufruf anbieten: `POST /colorpalette`. Hier sei auf das fehlende "s" hingewiesen. Dies würde sich meiner Meinung sogar eher anbieten, da `/colorpalettes` nur auf ein `GET`-Request reagieren sollte um eine Liste zurückzugeben und `POST /colorpalette` schon mit der zuvor aufgegriffenen Semantik von `POST` klarer ausdrückt, dass man eine untergeordnete Ressource erstellen will. Einer der Enität colourpalette untergeordnete Ressource.

5 2013-05-04 | Ressourcen-Identifizierung, Ausarbeitung der Semantik und Umsetzung eines XML-Schema

Für die Identifizierung der Ressourcen, habe ich mir erst mal klar gemacht, was im System alles eine grundlegende Entität repräsentiert. Nach einer kurzen Zeit stand fest, dass es drei grundlegende Entitäten gibt. Ein Benutzer (`user`), eine Farbe (`colour`) und eine Farbpalette (`colourpalette`). Von diesen drei Basis-Entitäten bzw. -Ressourcen, leiten sich dann leicht abweichende/erweiterte Ressourcen ab. Alle anderen Ressourcen stehen zuletzt nur im Zusammenhang mit den Basis-Ressourcen. Diese Abhängigkeiten und Abwandlungen, basierend auf die Basis-Ressourcen, ermöglichen die Einführung von einer Art Referenz. Referenzen können dazu genutzt werden um anzugeben, mit welchem anderen Element sie in Beziehung stehen. Sei es, dass es eine Spezialisierung einer Ressource ist, oder die Ressource nur im Kontext mit der referenzierten Ressource Sinn macht bzw. nur im Zusammenspiel mit dieser an Relevanz gewinnt. Diese Idee hat sich im Endeffekt auch komplett auf den Aufbau des XML-Schemas ausgewirkt. Elemente nutzen Referenzen, um andere Elemente im System zu referenzieren.
Auch die Ausarbeitung der Semantik der HTTP-Operationen hat von Identifizierung der Ressourcen profitiert. Aufbauend auf die zuvor identifizierten Basis-Ressourcen, konnte eine Hierarchie aufgebaut werden.

6 2013-05-06 | Meilenstein 1 + 2 - XML Schema, Ressourcen und die Semantik der HTTP-Operationen

Die Abnahme des ersten und zweiten Meilensteins verlief recht gut. Fragen bezüglich der eventuellen Löschung von Elementen mit noch existierenden Referenzen auf dieses Element, wurde damit beantwortet, dass es mehrere Möglichkeiten gibt. Meine Idee, alle Elemente in einer allumfassenden Struktur zu traversieren, und Elemente mit übereinstimmenden Referenzen löschen, wäre eine Möglichkeit. So wird höchstwahrscheinlich auch meine Vorgehensweise aussehen. Evtl. lasse ich die Lösch-Operationen auch komplett unimplementiert. Dies werde ich erst im Verlauf der Umsetzung des REST Webservice entscheiden, je nachdem wie sich der Zeitdruck bemerkbar macht.
Heute habe ich zudem das Schema um die Möglichkeit erweitert, eine allumfassende Struktur zu validieren (auch Java-Klassen mittels JAXB generieren zu lassen), die es erlaubt alle Daten des Systems aufzunehmen. Dies wird für die Umsetzung des REST Webservice wichtig sein, da irgendwo die Daten gespeichert sein müssen. Idealerweise in nur einer einzigen XML-Dokumenten-Instanz, anstatt viele kleine XML-Dokumente zu organisieren, die die jeweiligen Ressourcen widerspiegeln.

7 2013-05-11 | Umsetzung des RESTful Webservice

Als nächster Schritt war vorgesehen, aufbauend auf Jersey als Referenzimplementation für JAX-RS und Grizzly als HTTP-Server, den RESTful Webservice umzusetzen. Vorteilhaft war, dass ich mich schon vor einigen Wochen mit Jersey und Grizzly beschäftigt hatte und somit sehr schnell in die Entwicklung einsteigen konnte. Dadurch, dass man in Phase 1 schon Erfahrung mit JAXB gesammelt hat bzw. den Umgang mit den von JXB generierten Klassen, konnte beides wunderbar miteinander recht schnell kombiniert werden. Als etwas schwieriger stellte sich aber die Lagerung der Daten auf der Serverseite heraus. Datensätze zu Benutzern, Farben, Farbpaletten und ihren Kommentaren, mussten auf umständliche Weise in eine vom Schema ebenfalls beschriebenen XML-Struktur eingebettet werden. Durch die kleinen Unterschiede in den aus den `complex types` generierten Klassen, mussten viele Datensätze immer wieder umgebettet werden, um erst dann an den Request-Handler zurück zureichen bzw. vorher zu marshallen. Diese Umständlichkeit wurden aber in Kauf genommen, da der Sinn der Umsetzung des Webservice darin lag, dass Daten auf beiden Seiten (Client u. Server) marshalled bzw. unmarshalled werden. Somit existiert auf der Serverseite eine einzige Instanz eines XML Dokuments, wo alle dem System bekannten Datensätze eingepflegt oder entnommen werden. Sie fungiert quasi als Datenbankdatei.
Neben der Umsetzung des RESTful-Webservice, habe ich mich schon etwas über das PubSub-Prinzip informiert. Dazu auch die Seminararbeit `Ereignisgesteuerte Systeme im Web` von Benjamin Krumnow leicht überflogen, die im Medieninformatik-Wiki auf der Aufgabenseite von Phase2 verlinkt wurde. Ich habe vorgesehen, mich bis zum nächsten Milestone näher mit dieser Seminararbeit zu beschäftigen, speziell mit den Kapiteln "2.1 Publish Subscribe" und "3.1 XMPP".
Hinzu habe ich mich auch nach Beispielcode für SMACK umgeschaut und bin auch fündig ge-

worden. In den gefundenen Beispielcodes wird die Verwendung des vom XMPP unterstützten PubSub-Verfahrens veranschaulicht, genauer der Erstellung von Nodes, dem Versehen von Payloads und der Erstellung von Listenern.

8 2013-05-13 | Meilenstein 3 - RESTful Webservice

Die Abnahme des dritten Meilensteins verlief sehr gut. Es gab keine Probleme und ich kann somit die Meilensteine 4 + 5 angehen. Dafür habe ich mich, wie schon im Eintrag "Umsetzung des RESTful Webservice" erwähnt, in die Kapitel zu den Grundlagen zu ereignisgesteuerten Systemen im Web und PubSub eingelesen.

9 2013-05-18 | Openfire lokal eingerichtet / Einarbeitung in die Smack-API

Um den XMPP-Clienten umsetzen zu können, musste ich zuerst den Openfire-Server aufsetzen und einrichten. Dies ging auch recht schnell, wodurch ich auch sofort die SMACK-API ausprobieren konnte, um mit dem Openfire-Server kommunizieren zu können. Zu aller erst habe ich versucht Benutzer über einen Beispiel-Code am Server zu registrieren, um sich danach direkt über die Anmeldedaten einzuloggen. Auch dies ging sehr schnell und ohne Probleme. Erst als es darum ging LeafNodes zu erstellen, sie zu veröffentlichen (publish) und auf der anderen Seite zu abonnieren (subscribe) kam es zu vielen Fehlermeldungen und Exceptions-Würfen (SMACK setzt stark auf Routinen zur Fehlerbehandlung und weniger auf die Auswertung von Rückgabewerten). Nach einer intensiven Suche nach der Lösung zu meinem Problem, wurde klar, dass um explizit LeafNodes zu veröffentlichen und zu abonnieren bzw. den reibungslosen Zugriff auf die Nodes zu ermöglichen, beim Verbindungsaufbau als JID (Jabber-ID) `pubsub.<hostname>` angegeben werden muss. Zu aller erst hatte ich versucht nur über den `<hostname>` (in meinem Fall `localhost`) auf den Server zuzugreifen. Dies gelang am Anfang auch ohne nennenswerte Probleme. Erst als es darum ging die Existenz von LeafNodes abzufragen, kam es zu Fehlern und Exceptions, da nicht die entsprechende JID angegeben wurde. Diese muss aber explizit angegeben werden, um die PubSub-Funktionalität ohne Probleme verwenden zu können. Mit der Lösung dieses Problems war es dann auch möglich ohne Probleme LeafNodes zu veröffentlichen. Dies sogar mit oder ohne Payload (Fat Ping und Light Ping). Als nächstes werde ich mich stärker mit Swing beschäftigen, um die grafische Oberfläche des XMPP-Clienten zusammenstellen zu können.

10 2013-04-26 | Benutzerauthentifizierung bezogen auf XMPP

Da der RESTful Webservice nur als Prototyp gedacht ist und auf Benutzerauthentifizierung einfachheitshalber komplett verzichtet, der XMPP-Server aber bedingt durch die Veröffentlichungsmechanismus der PubSub-Funktion eine Authentifizierung voraussetzt, stellt sich na-

türlich die Frage, wie man diese Bereiche zusammenführen könnte. Sinnvoll wäre es eine Schnittstelle einzurichten, die sich alleinig darum kümmert sich beim RESTful Webservice sowie beim XMPP-Server gleichermaßen zu registrieren bzw. anzumelden. Somit müsste sich der Client nicht an mehreren Stellen anmelden, sondern nur an einer einzigen Stelle. Diese Anmeldeschnittstelle wäre dann dafür zuständig im Hintergrund die Aktionen des Benutzers entsprechend seiner Befugnisse auszuwerten und entsprechend zu erteilen. Für das Projekt ist aber nur vorgesehen, dass sich der spätere Client nur am XMPP-Server anmelden muss um asynchrone Benachrichtigungen zu erhalten, aber seine Befugnisse über auf dem RESTful Webservice auf keinster Weise beschränkt werden. Dies hat damit zu tun, dass der Webservice, wie bereits erwähnt, nur als Prototyp gedacht ist. Die Konzeptionierung einer Schnittstelle zur Benutzerauthentifizierung würde die Entwicklung nur unnötig verkomplizieren. Eine Umsetzung wäre aber nach Komplementierung der Phase2 noch denkbar und möglich.

11 2013-06-01 | XMPP und XML-Stanzas

Um die Dokumentation um näheres bezüglich XMPP zu erweitern, habe ich mich näher mit den RFCs beschäftigt, in denen XMPP in seinen Grundzügen beschrieben ist. Dadurch, dass XMPP auch über sogenannte XMPP Extension Protocols erweitert wird, habe ich mich auch über diese informiert. Dies war auch nötig, da einzelne Funktionen des Openfire-Servers, die man in Phase2 nutzt, nur über XEPs beschrieben sind. Als Beispiel sei das `XEP-0060` genannt, in dem XMPP um ein `Publish-Subscribe-Pattern` erweitert wird, aber auch `XEP-0030`, in dem die Möglichkeit beschrieben wird, wie Informationen über einzelne Entitäten im System bezogen werden können. Dies ist unter der Bezeichnung `Service Discovery` zusammengefasst. Genauer erlaubt es das "Service Discovery" nicht nur Indentifizierungsmerkmale einer Entität zu "entdecken", sondern alle von der Entität unterstützten Funktionen und Protokolle (evtl. ist die Entität ein PubSub-Node). Zusätzlich erlaubt es auch das Identifizieren von `Items`, die in Verbindung mit der Entität stehen. Abgebildet auf die Publish-Subscribe-Erweiterung sind mit `Items` die über einen Node veröffentlichten Daten (mit oder ohne Payload; Fat Ping u. Light Ping).
Neben dem `Service Discovery`, habe ich mich näher mit dem Aufbau von Streams und den drei XML-Stanza-Typen informiert. Hierbei war es interessant herauszufinden, dass die drei XML-Stanzas `message`, `presence` und `iq` Kommunikationsabfolgen repräsentieren.

- `Message` → verbindungslos (erwartet kein Response)
- `presence` → broadcast (`1`:n Kommunikation; orientiert sich am Publish-Subscribe-Pattern)
- `iq` → verbindungsorientiert (auf ein Request folgt Repsonse)

12 2013-06-03 | Meilenstein 4 + 5 - Konzeption asynchrone Kommunikation + XMPP – Client

Die Präsentation der Ergebnisse verlief gut. Obwohl mein XMPP-CLient noch kleine Probleme hatte, konnten diese beim Präsenztermin noch behoben werden. Problematisch war, dass ein

Benutzer, der veröffentlicht ebenfalls eine Benachrichtigung erhielt, somit im System ebenfalls Abonnent dieses Topics war, über den er veröffentlicht hat. Dies geschah wohl dadurch, er den entsprechenden Node direkt vor dem Veröffentlichen erzeugt hat und somit mit dem Node gekoppelt wurde. Als nächstes steht die Umsetzung des finalen Clienten an. Dazu werden ich auf die Codebasis des XMPP-Clients aufbauen, da diese schon mit Swing umgesetzt und die dafür geschriebene Verbindungsschnittstelle modular konzipiert wurde, wodurch sie wiederverwendet werden kann. Für die Kommunikation mit dem REST Endpoint muss ich mich aber noch einlesen.

13 **2013-06-17** | Fertigstellung des finalen Clients

Der Client wurde in seinen wichtigen Funktionen fertig umgesetzt. Die synchrone Anbindung zum REST Endpoint sowie die asynchrone Benachrichtigung durch den XMPP-Server funktionieren nun problemlos. Einzig die Kommentarfunktion wurde nicht implementiert, da sie zu unnötiger Implementierungsarbeit geführt hätte. Die Kommunikation mit dem RESTful Webservice wird schon an der Auflistung der Farben und Paletten gezeigt, sowie dem Erzeugen dieser.