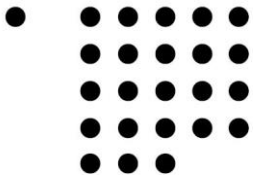

Fachhochschule Köln
Cologne University of Applied Sciences

Projekttagebuch zum Projekt Smart Building System von

Christian Ries 11085640

15.04.2013

An dem ersten Termin war die Projekt-Idee schnell gefunden. Ein Thema was momentan sehr aktuell ist, ist das Automatisieren von Gebäuden.Unsere Idee war also ein System zu entwickeln welches es einem Benutzer erlaubt verschiedene Funktionen eines Gebäudes zu steuern bzw. ab zu fragen.

Nachdem die Idee schnell gefunden war, machten wir uns sofort an die Überlegungen welche Funktionen und Rollen in dem System integriert werden sollen.

20.04.2013

Die Grundfunktionen und Rollen des Systems sind soweit festgelegt. Teilweise wurden anfängliche Ideen wieder verworfen da es den Rahmen des Projektes sprengen würde. Um die verschiedenen Funktionen besser zu veranschaulichen haben wir uns ein paar Szenarien überlegt und diese in der Dokumentation festgehalten.

22.04.2013

XML Schema vorläufig + Ausarbeitung und Dokumentation

Die weitere Ausarbeitung der Rollen und Funktionen sowie deren Dokumentation wurden vorgenommen. Des Weiteren wurde das XML-Schema besprochen und damit begonnen es mit Eclipse umzusetzen.

Das vorläufige XML-Schema wurde entwickelt, wobei es jedoch noch Verbesserungsmöglichkeiten gibt. Es wurde die Frage aufgeworfen welchen Datentyp wir für die ID’s, also die eindeutige Zuordnung der verschiedenen Ressourcen nehmen. Wir haben uns auf den Datentyp Integer geeinigt. Die Vorteile bzw. Nachteile der verschiedenen Typen werden in der Doku besprochen.

30.04.2013

Es wurden sich Gedanken über die grundlegende Kommunikation & Interaktion des Systems gemacht. Zur Verdeutlichung haben wir ein Sequenzdiagramm erstellt welches den groben Ablauf einer Kommunikation zwischen Client & Server darstellt.
Zusätzlich wurde die erste Version der REST-Hierarchie in Form eines Baum-Diagramms erstellt, und in die Dokumentation eingefügt.

06.05.2013

REST Hierarchie + HTTP Operationen festgelegt + Dokumentation

Nachdem wir alle Ressourcen definiert haben, machten wir uns daran die HTTP-Operationen für die verschiedenen Ressourcen zu definieren. Dafür haben wir uns eine Tabelle erstellt welche uns einen Überblick verschafft, und auch im weiteren Verlauf des Projektes hilfreich sein wird.

Diese Übersicht ist allerdings nur eine erste Version die im weiteren Verlauf des Projektes warscheinlich noch verändert wird. Das ganze haben wir dann erstmal in der Dokumentation festgehalten.

13.05.2013

Die ersten Schritte mit dem Grizzly Server & Jersey Framework wurden gemacht. Nach anfänglicher Ratlosigkeit wurde dann schnell das Konzept verstanden und wir haben probiert die erste, mit JAXB generierte Ressource mit Hilfe der RESTClient GUI anzusprechen.

Diese einfache GET-Anfrage hat funktioniert, lieferte aber noch kein XML. Um dann das entsprechende Format angeben zu können haben wir die @Produces Annotation gefunden mit welcher man den Datentyp definieren kann.

Es tauchte die Frage auf wie wir die verschiedenen Ressourcen also die Request-Handles aufbauen müssen. Sollte das verschachtelt werden oder sollte für jede Ressource ein eigener Request-Handler geschrieben werden. Nach umfangreichen Gesprächen mit fachlichen Mitarbeitern der FH kamen wir dann zum Schluss dass es sinnvoll ist einen einzigen Request-Handler zu schreiben in welchem die Ressourcen verschachtelt werden.

Wir können also nun fortfahren und damit beginnen alle Ressourcen zu implementieren und die nötigen HTTP-Operationen zu integrieren.

17.05.13

Ein XMPP Server sollte eingerichtet werden. Entschieden haben wir uns für einen Openfire Server.

Für diesen gibt es im Wiki ein Tutorial welches den Einstig erleichtert und wir zur Hand genommen haben. Daraufhin wurde der Server eingerichtet und und dessen Funktionen genauer unter die Lupe genommen. Die Smack-APi wurde für den Zugriff auf den Servern genommen. Da ich mich zumindest mit Smack-API nicht auskannte, musste ich mich hier erstmal reinlesen. Dazu habe ich vorwiegend das Internet zum recherchieren benutzt.

Auseinandersetzung mit Openfire Smack API

Für den zweiten Teil der Phase2 ist ein XMPP Server erforderlich. Dieser muss installiert und konfiguriert werden. Die Vorgabe ist ein Openfire-Server. Zur Einrichtung dieses Servers gibt es ein Tutorial welches wir uns angesehen haben, und anhand dessen wir dann auch erfolgreich den Server installiert haben.

Damit die Kommunikation mit dem Server realisiert werden kann, sollen wir die Smack(x) API benutzen. Da diese aber noch Neuland für uns ist, ist erstmal eine Einarbeitung in die API erforderlich. Zusätzlich müssen wir uns mehr mit dem Thema Publish-Subscribe auseinander setzen, da uns die Grundlagen noch teilweise fehlen.

03.06.2013

Leaf (Topics), definierte Asynchrone Funktionen, Welche Daten sollen übertragen werden ?
Wer ist Subscriber ?

Da wir ganz am Anfang in der Konzeptionsphase uns schon Gedanken drüber gemacht haben welche asynchronen Funktionen wir einbauen möchten, fiel die Fall der Topics recht leicht.
Da wir auch schon die Rollen definiert haben war die Frage nach Publisher, Subscriber schnell geklärt. Die Besonderheit ist jedoch das der Publisher nicht ein Benutzer ist sondern die Feuermelder, Bewegungsmelder und die Kontakte selber die Publisher sind. Dies kommt daher das die Sensoren durch eine Zustandsänderung eine Nachricht auslösen und somit als Publisher fungieren.

Als Daten haben wir eine Simplepayload welches unsere XML-Struktur enthält. Somit sieht der Benutzer sofort welcher Sensor die Meldung ausgelöst hat.

10.06.2013 - 23.06.2013

Ausarbeitung des Clients und der XMPP-Funktionen

Dadurch dass wir viele einzelne Funktionen in unserem System haben, hat die Ausarbeitung des Clients sehr viel Zeit in Anspruch genommen.Wir haben uns überlegt wie dieser Ausehen könnte, und haben uns dafür verschiedene Mockups erstellt. Jedoch merkten wir schnell dass sich im Laufe der Entwicklung viel geändert hat bzw. hinzugekommen ist und wir uns nicht an die anfänglich erstellen Mockups halten konnten.

Es stellte sich herraus dass, der Anfangs für klein betrachtete Teil „XMPP“ doch nicht so klein war und viele Fehler aufgetreten sind. Zum einen war nicht ganz klar wie es möglich ist eine Meldung zubekommen sobald eine Nachricht gepublished wird, und zum anderen gab es teilweise Probleme beim Publishen, da die Konfiguration der Nodes nicht richtig war.