Fachhochschule Köln
Cologne University of Applied Sciences

Institut für Informatik

Webbasierte Anwendungen 2

Dozent:

Prof. Dr. Kristian Fischer

Betreuer:

# Dokumentation

**Phase 2**

Mathias Rothert

Matrikelnummer: 11082436

Matthias Rößler

Matrikelnummer: 11083608

Medieninformatik Bachelor

4. Semester

# Inhaltsverzeichnis

# 1. Einleitung

Im Workshop Webbasierte Anwendungen 2 geht es um die Konzeptionierung und Entwicklung eines verteilten Systems. Die Interaktion zwischen den Systemkomponenten soll synchron unter Anwendung des Architekturstils REST und asynchron unter Anwendung von XMPP erfolgen. Des weiteren, soll ein Client erstellt werden, welcher die Funktionalität des Systems verdeutlicht. Das System ist frei wählbar.

Diese Dokumentation soll die Vorgehensweise zur Entwicklung des verteilten Systems zeigen. Hierzu gehören die Ideenfindung sowie die Planung, Konzeptionierung und Prüfung der für das System benötigten XML Schemata, die Entwicklung der HTTP-Operationen und Ressourcen im Kontext der RESTful Webservices, die Erstellung des RESTful Webservice und XMPP Client in Java und die Entwicklung des grafischen User Interfaces.

# 2. Ideenfindung

Das finden einer Problemstellung, welche mit einem verteilten System, das alle o. g. Technologien und Architekturstile sinnvoll nutzt, gelöst werden kann, stellt den Anfang des Projektes dar.

Ein Kellnersystem mit mehreren mobilen Kassen, benötigt eine synchrone Kommunikation mit einer serverseitigen Systemkomponente, welche alle benötigten Ressourcen bereitstellt. So lassen sich über eine POST HTTP-Operation z. B. Bestellungen hinzufügen oder bei einem Zahlungsvorgang über DELETE eine Bestellung löschen. Die asynchrone Interaktion findet statt, wenn eine Bestellung am Server eingeht bei der Essen enthalten ist, der Server sendet hier asynchron eine Nachricht an den Koch, der wiederum benachrichtigt den Server, sobald das Essen fertig ist. Die Mobile Kasse, die sich für die Nachricht subscibed hat, erhält die Aufforderung an den Kellner, das Essen abzuholen.

# 3. Konzept

## 3.1. Kommunikationsabläufe

### 3.1.1. Synchrone Datenübertragung

Die meiste Kommunikation läuft synchron ab, d. h. eine Mobile Kasse, bzw. das Terminal in der Küche ruft über HTTP Methoden Ressourcen vom Server ab und erhält im Anschluss eine Antwort.

#### Bestellvorgang

Beim Bestellvorgang schickt die Mobile Kasse die Tischnummer und eine Bestellung an den Server und erhält im Anschluss einen HTTP-Statuscode.

#### Zahlungsvorgang

Die Mobile Kasse fragt über eine GET-Methode alle Bestellungen zu einem Tisch ab und wartet auf die Antwort. Eine oder mehrere Bestellungen werden an den Server übertragen um dort als Gezahlt markiert zu werden. Auch dieser Vorgang wird mit einem HTTP-Statuscode quittiert.

### 3.1.2. Asynchrone Datenübertragung

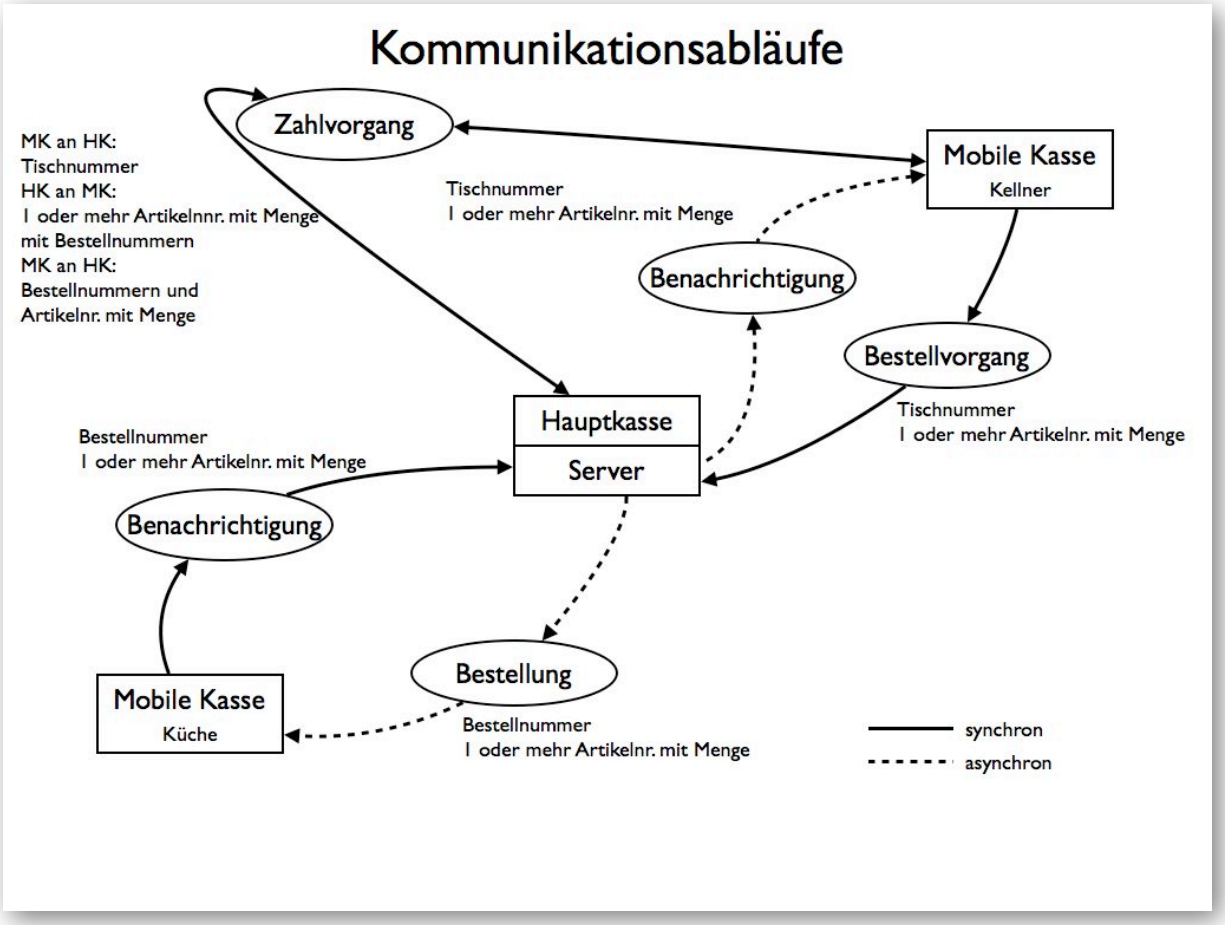
Soll das Terminal in der Küche oder eine Mobile Kasse benachrichtigt werden, so erfolgt diese Kommunikation asynchron.

#### Küche benachrichtigen

Die Küche wird vom Server über eine neue Bestellung benachrichtigt. Der Server wartet dabei nicht auf eine Antwort der Küche.

#### Kellner benachrichtigen

Ist eine Bestellung zum abholen bereit, werden die Kellner von dem Küchen Terminal hierüber benachrichtigt, auch diese Kommunikation ist asynchron.




### 3.2. XML Schemata

Die serverseitige Systemkomponente soll alle Daten zu Bestellungen und Tischen persistent in einer XML Datei bereitstellen. Zu einer Bestellung gehören eine bis mehrere verschiedene Artikel mit einer Menge. Da es außerdem eine XML Datenbasis für bestellbare Artikel geben muss besteht die Möglichkeit über identifizierende Artikelnummern aus der XML-Datei für die Tisch- und Bestelldaten auf diese Artikel Datenbasis zu referenzieren. So wird zu den Bestellungen immer nur die Artikelnummer sowie die Menge im XML-Dokument geschrieben, Artikelname, Preis und ggf. weitere Informationen sind in der Artikel-XML gespeichert. Problem bei diesem vorgehen ist, dass die Artikel nicht während einer Sitzung (Schicht) geändert werden dürfen, da sich ansonsten bspw. der Preis zwischen der Bestellung und dem Zahlungsvorgang ändern könnte. Die Alternative wäre auch diese Daten mit in dem XML Dokument für das komplette Kellnersystem zu schreiben.

Soll nur eine Bestellung abgefragt und übertragen werden, ist es erforderlich eine neue XML Struktur zu erstellen und diese Bestellung zu einem Tisch hinzuzufügen, diese Struktur zu übertragen und vom ersten und einzigen Tisch die erste und einzige Bestellung  anzusprechen. Es muss also eine XML-Baumstruktur mit unnötig vielen Knoten übertragen werden. Die Alternative wäre hier für die verschiedenen Ressourcen verschiedene XML Schemata zu erzeugen, allerdings würde dies bedeuten dass ebenso viele Java-Klassen über JAXB erstellt werden müssen und Typen dieser Klassen wären untereinander inkonsistent.

### 3.3. HTTP-Operationen und Ressourcen

Bei den Verschiedenen Kommunikationsabläufen müssen unterschiedliche Ressourcen abgefragt werden. So muss bspw. bei einer Bestellung eine POST Operation auf eine Ressource erfolgen die einen Tisch eindeutig identifiziert. Aus den erarbeiteten Kommunikationsabläufen ist ersichtlich welche Ressourcen benötigt werden.

| Ressource | URI | Methode |
|---|---|---|
| Liste aller Artikel | /artikel/ | GET, POST |
| Einzelner Artikel | /artikel/{id} | GET, PUT |
| Liste aller Tische | /bestellungen/ | GET |
| Liste aller Bestellungen zu einem Tisch | /tisch/{id}/bestellungen | GET, POST, DELETE |
| Einzelne Bestellung | /bestellungen/{id} | GET, PUT, DELETE |

# 4. Entwicklung

### 4.1. RESTful Webservice

Zur Umsetzung des RESTful Webservice wird auf den Grizzly-Server aufgesetzt. Java-Methoden werden so nicht über den Methodennamen sondern über die HTTP-Operationen und Ressourcen angesprochen. Mit den von JAXB erstellten Klassen wird das marshalling und unmarshalling der XML-Dokumente durchgeführt. Mit den

Methoden der Klassen wird dann das gewünschte Objekt XML-formatiert übertragen bzw. geändert.

Für die Entwicklung der Clienten ist es wichtig das diese Methoden möglichst einfach aufgerufen werden können, daher soll eine Schnittstellenklasse für alle möglichen HTTP-Operationen auf Ressourcen bereitstehen.

## 4.2. XMPP

### 4.2.1. API

Der XMPP-Server wird mit Openfire realisiert.
Als Schnittstelle zum XMPP-Server wird SMACK in der Version 3.3 benutzt.

### 4.2.2. Das Publish-Subscribe Prinzip

Das Publish-Subscribe Prinzip beruht auf einer verbindungslosen Kommunikation zwischen unterschiedlichen Usern, wobei es immer nur einen „Redner“ (Publish) und viele „Hörer“ (Subscribe) gibt, ähnlich einem Broadcast in einer Netzwerkumgebung. Die Kommunikation läuft über einen Server, welcher in der Mitte steht.
Ein Publisher erstellt einen sogenannten Node auf dem Server für den sich Subscriber anmelden können und an dem der Publisher Nachrichten senden kann. Interessiert sich ein Subscriber für dieses Thema(Node) so kann er sich für dieses Thema anmelden und bekommt alle Nachrichten vom Publisher.

### 4.2.3. Verwendung des Publish-Subscribe Prinzips

Das Publish-Subscribe Prinzip im Kellnersystem beruht auf zwei Topics, über welche die Kellner mit der Küche kommunizieren können. Die Kellner subscriben sich bei der Küche. Ein Koch schickt allen Kellnern eine Nachricht, damit diese das fertige Essen abholen. Ebenso subscriben sich die Köche beim REST-Server welcher ihnen mitteilt, dass ein Essen zubereitet werden muss.
Aufgrund der asynchronen Architektur muss weder der Server auf die Antwort eines Kochs warten noch ein Koch auf die Antwort eines Kellners um das System weiter zu nutzen.

# 5. Probleme und Anmerkungen

## 5.1. Begrenzter Funktionsumfang

Bei der Idee, Konzeptionierung und Entwickung wurde das System auf das Nötigste reduziert, da nicht von Anfang an abzusehen ist wie hoch der Aufwand sein wird. Auch ist aus der Aufgabenstellung nicht klar, welchen Umfang das verteilte System haben soll. So fehlen bspw. folgende für ein Kassensystem wichtige Funktionen:

- Bezahlmöglichkeiten einzelner gebuchter Artikel (unterschiedlicher Bestellungen) zu einem Tisch
- Bezahlte Bestellungen/Artikel müssen in einer weiteren Datenbasis persistent gespeichert werden
- Der Koch sollte keine Informationen über bestellte Getränke erhalten
- Verschicken von Nachrichten mit Simplepayload über XMPP

## 5.2. Persistente Datenspeicherung mittels XML

Während der Entwicklung fällt auf, dass der Umgang mit XML zur persistenten Datenspeicherung sehr ressourcenverschwendend ist. So ist es bspw. nicht möglich einzig über eine Bestellnummer eine Bestellung zu erhalten, da ansonsten jeder Tisch nach dieser durchsucht werden müsste. Dies ist auch der Grund warum die Ressource „/bestellung/id/“ für die GET-Methode in „/tisch/{id}/bestellung/{id}“ geändert ist. So ist erforderlich, dass die Tisch-Nummer bekannt ist wenn eine eindeutig identifizierte Bestellung abgefragt wird. Eine Lösung wäre hier sicherlich mit einer Datenbank zu arbeiten.

# 6. Leistungsmatrix

| Themenbereich | Mathias Rothert | Matthias Rößler |
|---|---|---|
| Konzeption | 50 % | 50 % |
| Implementierung | 50 % | 50 % |
| Dokumentation | 50 % | 50 % |

# 7. Literaturliste


REST und HTTTP Steafan Tilkov dpunkt.verlag