Architekturdiagramm

Zur Erstellung des Architekturdiagramms galt es die Kommunikation der unterschiedlichen Softwarekomponenten zu modellieren, zu denen zusätzlich die Benennung der jeweiligen Kommunikationsprinzipien und Protokolle notwendig ist. Des Weiteren ist es erforderlich die Repräsentation der Netzwerkdaten klar aufzuzeigen, während außerdem die Verteiltheit des Systems deutlich zu erkennen ist.

Zuerst ist zu erwähnen, dass die Trennung des Systems in einen Dienstgeber und einen Dienstnutzer den Zweck verfolgt, sowohl dem letztendlichen Endgerät als auch dem API-Anbieter „VRS“ eine markante Menge Rechenarbeit abzunehmen. Dies hat zur Folge, dass zum einen im Optimalfall der Benutzer keine langen Ladezeiten erdulden muss und zum anderen wir uns damit an die vom API-Anbieter gestellten Nutzungsvereinbarungen halten. Denn diese besagen Sinngemäß unter anderem, dass das von uns entwickelte System zu keinen signifikanten Performanceeinbrüchen seitens der API-Server führen darf, beziehungsweise sollte. Um diesem Punkte so gut wie möglich gerecht zu werden, entschlossen wir uns, die verschiedenen Nutzer unseres Systems sich nur so selten wie möglich mit der API der VRS verbinden zu lassen.
So schickt der Dienstnutzer nur zu Beginn der Zeitplanerstellung eine einmalige Anfrage an die API, wodurch dieser in der Lage ist, die für den Benutzer benötigten Zeitplan korrekt zu erstellen. Da es bei öffentlichen Verkehrsmitteln allerdings zu Verspätungen und Ausfällen kommen kann, wäre es dem Dienstnutzer notwendig in regelmäßigen Abständen eine Anfrage an die API zu stellen, also „polling“ zu betreiben. Dies hat allerdings den großen Nachteil, dass es bei einer großen Nutzermenge zu einer enormen Anfragemenge pro Minute kommen kann. Um dieses Problem zu lösen, entschlossen wir uns, dem Dienstgeber die Aufgabe zuzuteilen sich um die Verspätungen der Verkehrsmittel zu kümmern. So fragt der Dienstgeber nach jetzigem Stand gemäß einem gewählten Zeitintervall die Daten zur Bestimmung von Verspätungen ab und stellt diese den Dienstnutzern bereit, welche sich dadurch für die Aktualisierung von Abfahrtszeiten ausschließlich an den Dienstgeber richten. Da damit das eingegangene Problem allerdings nur verschoben und nicht behoben ist, entschlossen wir uns zudem, das Publish-Subscribe Kommunikationsprinzip zu verwenden. Unter Verwendung dessen, stellen die Dienstnutzer nicht mehr die Anfragen direkt an den Dienstgeber, sondern bekommen die für sie notwendigen Daten bei Veränderung automatisch mitgeteilt, sodass damit das Problem der möglichen Überbelastung weitestgehend aus dem Weg geräumt wurde.

Da es dem Dienstgeber jedoch notwendig ist, die Daten der VRS-API abzurufen, stellt dieser beispielsweise alle 30 Sekunden lediglich eine Anfrage an den Anbieter. Das Ergebnis dieser architektonischen Umstellung ist eine signifikant niedrigere Anfragenmenge an den API-Anbieter, denn verglichen mit dem zuvor genannten Szenario steigt die Anzahl der Anfragen pro Minute mit jedem Benutzer unseres Systems, während sich die Anfragenmenge nun stetig auf einem vergleichsweise niedrigen Niveau befindet.

Zur Bestimmung der Laufdauer zu einer Bushaltestelle und dem letztendlichen Ziel ist es notwendig einen weiteren API-Anbieter, in unserem Fall OpenStreetMap, in das System einzubinden. Mit diesem wird allerdings das Gerät jedes Nutzers direkt kommunizieren, da dort nur eine niedrige Anfragenanzahl pro Verwendungsfall zu erwarten ist.

Da unser System allerdings zusätzlich in der Lage sein soll, für die Zeitplanerstellung notwendige Daten selbstständig annähern und ermitteln zu können, umfasst die Kommunikation unter den Systemkomponenten außerdem den Kommunikationskanal zwischen Dienstnutzer und Dienstgeber.

In diesem werden bei der initialen Nutzung eines Endgerätes einmalig Daten vom Dienstgeber erfragt, während im späteren Verlauf die Kommunikation überwiegend vom Dienstnutzer ausgehend stattfindet. Aus dem Grund, dass der Dienstgeber über „predictive analytics“ notwendige Daten ermittelt, ist es erstrebenswert möglichst viele Einflussnehmende Daten in die Erstellung von Vorhersagen einfließen zu lassen. Um dies zu gewährleisten schickt der Dienstnutzer in einem zu bestimmenden Intervall die erforderlichen Daten an den Dienstgeber, sodass dieser sie für die Berechnung aufgreifen und verwenden kann.

Mit Ausnahme des bereits genannten Publish-Subscribe Prinzips findet die Übermittlung von Daten stets in synchroner Art statt. Dies hat den Grund, dass die verwendeten API-Anbieter lediglich eine solche synchrone Kommunikation anbieten, was allerdings auch den Vorteil hat, dass es jedem Nutzer ermöglicht, die für sich notwendigen Daten spezifisch abzufragen. Des Weiteren ist die Netzwerkdarstellung mit einer Ausnahme stets als XML definiert. Diese Verwendung liegt auch hier darin Begründet, dass die Kommunikation mit den API-Anbietern stets über XML stattfindet. Sowohl Anfragen als auch Antworten werden in diesem Format dargestellt. Da beide API-Anbieter ein XML-Schema bereit stellen ist es dadurch möglich alle einkommenden und ausgehenden Netzwerkdaten überprüfen zu können um sicher zu stellen, dass keine Fehlerhaften oder falschen Daten von uns abgeschickt oder zur Verarbeitung herangezogen werden.
In der zuvor genannten Ausnahme erfolgt die Kommunikation zwischen Dienstnutzer und Dienstgeber mittels der JSON-Darstellung. Aus dem Grund, dass die Überprüfung der Daten unter verschiedenen Systemen entfällt, wurde sich an dieser Stelle aufgrund der übersichtlichen Notation und der Unterstützung von zahlreichen Programmiersprachen für JSON entschieden. Dadurch, dass der Dienstnutzer in unserem System in Java und der Dienstgeber in JavaScript realisiert wird, wird, aufgrund des letzteren Aspektes, der JSON-Darstellung in diesem Kommunikationsfall ein besonders hoher Stellenwert bemessen.