Ausarbeitung

Entwicklung einer Satelliten-Finder-App für Android

Daniel Kaiser

Daniel Kaiser

Entwicklung einer Satelliten-Finder-App für Android

Ausarbeitung im Rahmen des Moduls Mobile und verteilte Systeme

Studiengang Bachelor Mobile Computing

Fachbereich II

Fachhochschule Bingen

Betreuer : Prof. Dr.-Ing. Cornelius Wille

Durchführung: Norman Peitek

11. September 2015

Inhalt

Einleitung

Im Rahmen der Vorlesung „Mobile und verteilte Systeme“ soll eine Android-App erstellt werden, bei der die erlernten Fähigkeiten und Fertigkeiten in die Praxis umgesetzt werden. Das Projekt wird bewertet und stellt die Prüfung des Moduls dar.

Zielbestimmungen

Motivation

Wenn der Himmel klar ist, kann man vor allem am Abend und in der Nacht häufig Satelliten sehen. Diese sich „schnell bewegenden Sterne“ sind heutzutage keine Seltenheit mehr, da der Orbit um die Erde mit vielen Satelliten unterschiedlicher Funktion gefüllt ist. Wettersatelliten, Kommunikationssatelliten, Spionagesatelliten, Erdbeobachtungssatelliten, Raumstationen und vielem mehr.

Idee der App

Satelliten und andere Objekte im Orbit finden und anzeigen.

Anwendungsszenarien

1. Der Anwender entdeckt am Himmel ein sich konstant bewegendes Objekt - er vermutet einen Satelliten. Die App zeigt an, welche Satelliten sich gerade über ihm befinden.
2. Der Anwender möchte wissen, wo sich welche Satelliten gerade befinden.

Verwendungsgründe

Die App zeigt die genauen Positionen von Objekten als Augmented Reality an. Des Weiteren besitzt die App ein übersichtliches User Interface und enthält keine Werbung.

Design

Die App soll mit einem einfachen User-Interface erscheinen. Deshalb wurde folgendes Design erdacht.

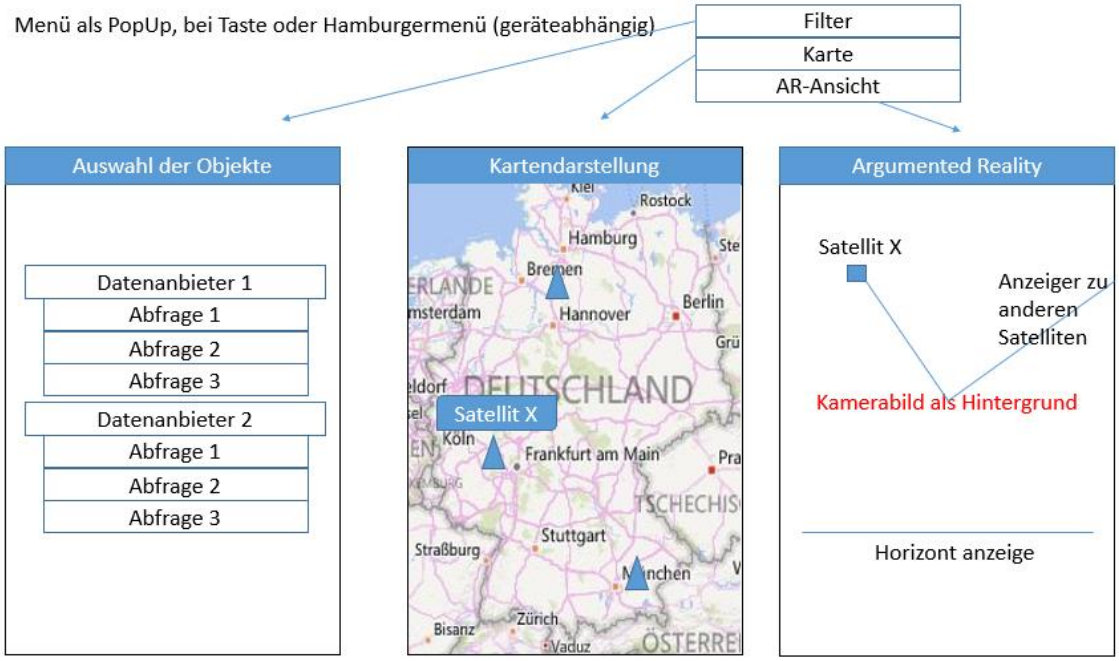

Abbildung 1 UI-Design Mockup

Hinweis: Da das UI nicht im Vordergrund steht, wird dieser Teil nicht weiter erläutert.

Implementation

Datenanbieter

Um die gewünschten Funktionen der App zu realisieren, insbesondere die Anzeige der *aktuellen* Satellitendaten, kann man auf mehrere Datenanbieter zurückgreifen. Notwendig für die AR-Funktion sind dabei Daten, die die Umlaufbahn beschreiben und nicht nur die aktuelle Position über Grund zurückgeben. Für die Kartenfunktion reicht diese jedoch aus.

Nach gründlicher Recherche haben sich zwei Datenanbieter als nützlich herausgestellt, da diese eine sehr gute Dokumentation der Schnittstellen bereitstellen und der Datenumfang als ausreichend erachtet wurde:

Space-Track[1] und [2] OpenNotify[2]

1. OpenNotify bietet eine einfache Möglichkeit, die aktuelle Position der ISS abzufragen und wurde hauptsächlich zu Testzwecken implementiert.

2. Space-Track bietet umfassende Daten zu Satelliten ab einer Größe von ca. 10cm, d.h. es sind auch Wrackteile und andere Objekte in diesem System erfasst. Aktuell sind rund 97 Millionen Objekte in der Datenbank.

Abfragemöglichkeiten

OpenNotify

Folgende Abfragemöglichkeiten werden angeboten:

1. Aktuelle Position der Internationalen Raumstation (ISS)
2. Aktuelle Besatzung der ISS
3. Überflugvorhersage für eine Position

Interessant ist für die Anwendung ist Nr. 1.

Space-Track

Die angebotenen Abfragemöglichkeiten sind sehr umfangreich. Eine komplette Übersicht findet man unter: Request Classes[3]

Für die App interessant ist dabei die Abfrage nach „latest_tle"-Objekten, die die letzten Objekt-Daten liefert. Die Abfragen müssen zudem nach weiteren Kriterien spezifiziert werden, da die Ergebnismenge sonst zu groß ist und Space-Track keine Daten liefert. Die Parameter zur Filterung der zu liefernden Ergebnisse sind hier[4] beschrieben.

Space-Track bietet einige Standardabfragen an, um interessante Ergebnisse zu erhalten. Darunter befinden sich Abfragen nach Wettersatelliten, Raumstationen, Amateursatelliten und viele weitere.

Diese Abfragen bieten sich zur Verwendung in der App an, da damit bereits einige Tausend Satelliten angezeigt werden können.

Formate und Regeln

OpenNotify

OpenNotify liefert Ergebnisse in JSON (JavaScript Object Notation) und aktualisiert die Daten jede Sekunde. OpenNotify weist darauf hin, dass in den meisten Fällen 5 Sekunden als Intervall zur Abfrage der ISS-Position ausreichend sind.

Eine Autorisierung ist nicht erforderlich.

```
{
  "message": "success",
  "timestamp": UNIX_TIME_STAMP,
  "iss_position": {
    "latitude": CURRENT_LATITUDE,
    "longitude": CURRENT_LONGITUDE
  }
}
```

Abbildung 2 Format JSON OpenNotify

Space-Track

Space-Track bietet eine Vielzahl Formate an: XML, HTML, CSV, TLE, 3LE und JSON Space-Track weist ausdrücklich darauf hin, maximal 20 Abfragen pro Minute durchzuführen. Häufigere Abfragen führen zur temporären Sperrung des Accounts. Für die App wird eine Aktualisierung einmal pro Stunde empfohlen.

Eine Autorisierung ist erforderlich.


```
{
  "ORDINAL": "1",
  "COMMENT": "GENERATED VIA SPACETRACK.ORG API",
  "ORIGINATOR": "JSPOC",
  "NORAD_CAT_ID": "7530",
  "OBJECT_NAME": "OSCAR 7",
  "OBJECT_TYPE": "PAYLOAD",
  "CLASSIFICATION_TYPE": "U",
  "INTLDES": "74089B",
  "EPOCH": "2015-09-09 12:34:10",
  "EPOCH_MICROSECONDS": "690176",
  "MEAN_MOTION": "12.53616459",
  "ECCENTRICITY": "0.0012157",
  "INCLINATION": "101.5308",
  "RA_OF_ASC_NODE": "225.6431",
  "ARG_OF_PERICENTER": "136.91",
  "MEAN_ANOMALY": "337.7016",
  "EPHEMERIS_TYPE": "0",
  "ELEMENT_SET_NO": "999",
  "REV_AT_EPOCH": "86782",
  "BSTAR": "0.00013678",
  "MEAN_MOTION_DOT": "-2.2e-07",
  "MEAN_MOTION_DDOT": "0",
  "FILE": "1923552",
  "TLE_LINE0": "0 OSCAR 7",
  "TLE_LINE1": "1 07530U 74089B   15252.52373484 -.00000022  00000-0  13678-3 0  9997",
  "TLE_LINE2": "2 07530 101.5308 225.6431 0012157 136.9100 337.7016 12.53616459867824",
  "OBJECT_ID": "1974-089B",
  "OBJECT_NUMBER": "7530",
  "SEMIMAJOR_AXIS": "7827.541",
  "PERIOD": "114.867",
  "APOGEE": "1458.922",
  "PERIGEE": "1439.890"
}
```


Abbildung 3 Format JSON Space-Track

Folgerungen

Um die Implementierungen einfach zu halten bietet sich an, JSON als Standard-Format in der App einzusetzen. JSON ist ein häufig verwendetes Format zur Beschreibung von Objekten, daher sind in allen gängigen Programmiersprachen bereits Klassen und Funktionen zum Umgang mit JSON implementiert.

Um die erhaltenen Daten in Java verwenden zu können, ist es nötig, sie aus dem Text in ein Objekt zu deserialisieren. Dazu müssen Klassen erstellt werden, die alle Felder der Notation enthalten und entsprechend abbilden.

Es wird für jeden Datenanbieter in der App eine entsprechende Klasse zum Deserialisieren der JSON-Daten erstellt.

Software-Architektur

Überlegungen

Die App ist wie im UI-Design sichtbar in 3 UI-Teile aufgeteilt. In einem UI-Element wählt der Nutzer die abzufragenden Daten aus. In den beiden anderen Elementen werden die Daten jeweils auf unterschiedliche Weise dargestellt.

Somit ergeben sich zwei Objekt-Typen: Datenquelle und Datensenke.

Für den Fall, dass weitere Senken hinzukommen, soll auf eine zu feste Verbindung zwischen Quelle und Senke verzichtet werden. Es muss daher ein praktischer Weg gefunden werden, Änderungen an den jeweiligen Komponenten durchführen zu können, ohne Anpassungen an der Verbindung zu tätigen.

Des Weiteren muss beachtet werden, dass die anzuzeigenden (Satelliten-) Daten in einem Intervall < 1 Minute neu berechnet werden müssen, damit die Positionen stets aktuell sind.

Zusätzlich wird davon ausgegangen, dass die Daten nur in eine Richtung gesendet werden, d.h. eine Datensenke wird nie selbst Ereignisse auslösen.

Eine ereignisbasierte Architektur stellt somit eine praktikable Lösung dar.

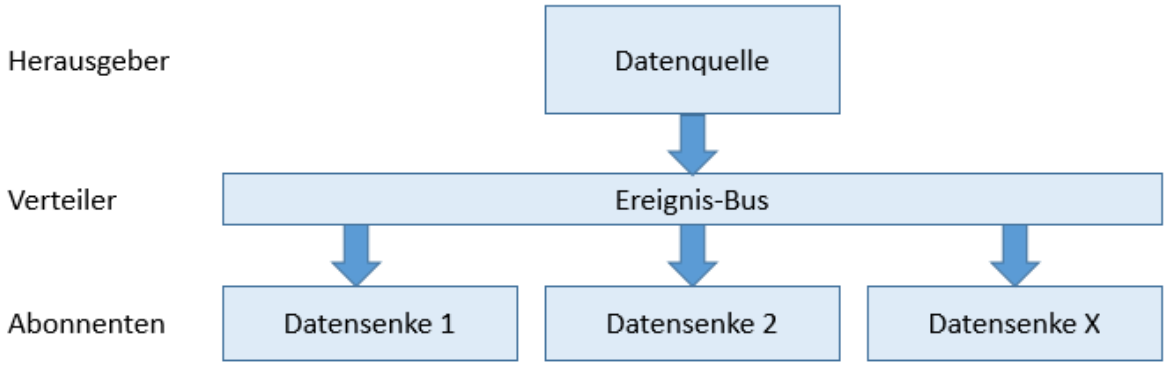

Abbildung 4 Entwurf Software-Architektur

Die Datenquelle löst Ereignisse aus, die für Abonnenten Daten liefern. Für die Kartendarstellung ist ein Ereignis relevant, bei dem neue Positionsdaten übermittelt werden. Für die AR-Darstellung sind Ereignisse mit Richtungsangaben interessant. Des Weiteren soll es möglich sein, Komponenten mitzuteilen, wenn von einer Datenquelle keine weiteren Aktualisierungen kommen. Dies kann nützlich sein, um z.B. die Kartendarstellung entsprechend zu bereinigen.

Ein weiterer Vorteil der ereignisbasierten Architektur ergibt sich durch die Entkopplung der unterschiedlichen Datenformate und Komponenten. Eine Datenquelle kann nur ein bekanntes Ereignis auslösen, es ist also die Aufgabe der Datenquelle, die speziellen Datenformate in allgemein bekannte Ereignisformate umzuwandeln.

Ereignisbasierte Architektur

Um die Kommunikation zwischen den Komponenten zu ermöglichen, werden mehrere Ereignisse definiert:

Ereignis	**Inhalt**	**Abonnenten**
MapLocationEvent	Positionsdaten und Objektinformationen	Kartendarstellung
SpotDirectionEvent	Richtungsangaben und Objektinformationen	Augmented Reality Darstellung
SourceStoppedEvent	Information über eine Datenquelle, die keine weiteren Aktualisierungen liefert	Kartendarstellung, Augmented Reality Darstellung

Als Verteiler-System wird EventBus verwendet. EventBus ist ein für Android optimiertes System zur Veröffentlichung und dem Empfang von Ereignissen. Weitere Informationen und Beispiele gibt es bei GitHub[5] und im Kapitel Frameworks und Dependencies.

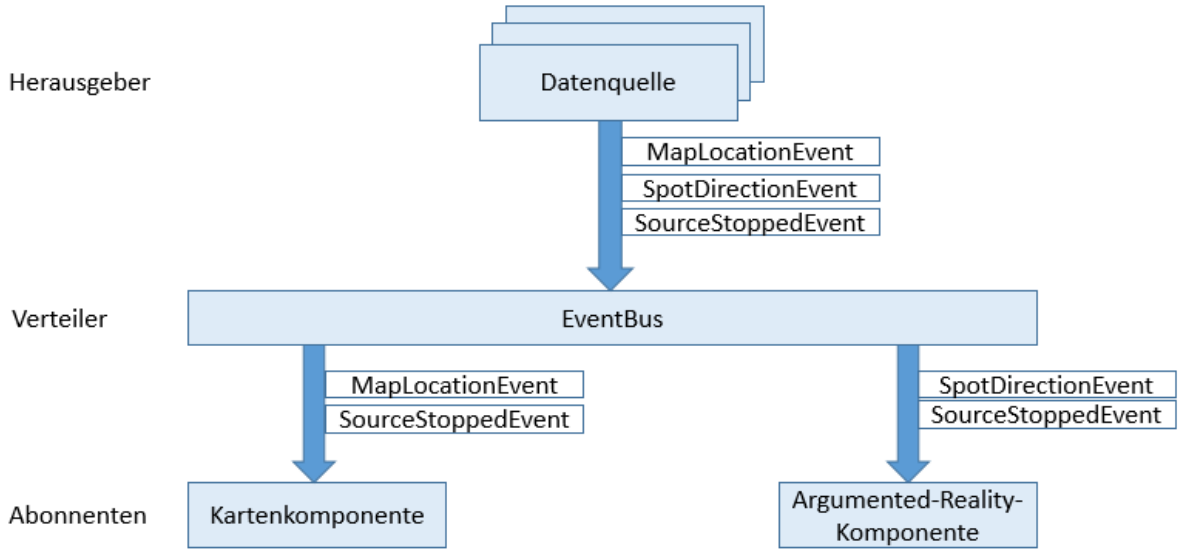

Abbildung 5 Entwurf Software-Architektur

Die Ereignisse werden von der Datenquelle ausgelöst. Es können mehrere verschiedene Datenquellen parallel existieren. Sie nutzen den gleichen Bus und sind nicht voneinander abhängig.

System-Architektur

Überlegungen

Die Anwendung ist nach Ermitteln des groben UI-Designs und der Softwarearchitektur darauf ausgelegt, nur Daten abzufragen, zu verarbeiten und darzustellen. Es werden keine (Arbeits-) Daten zu einem Server gesendet. Es ist je nach Datenquelle notwendig sich auf einem Server zu autorisieren. Die Anwendung erfüllt somit die Client-Server-Architektur.

Client-Server-Architektur

Die einzelnen Datenanbieter stellen Datenserver und APIs zur Verfügung, um mit einem Client darauf zuzugreifen. In einigen Fällen ist dazu eine Autorisierung erforderlich.

Die Anwendung benötigt für jeden zu verwendenden Datenanbieter eigene Implementationen in Form von Klassen und/oder Interfaces. Jede Implementierung muss sich dabei um die Abfrage und die Verarbeitung der vom Server empfangenen Daten kümmern und die Ergebnisse mittels vorgegebener Ereignis-Klassen veröffentlichen.

Jegliche Verarbeitung findet somit auf dem Client, der App, statt.

Konsistenzmodell

Überlegung

Die Umlaufbahnen von Satelliten ändern sich bis auf wenige Ausnahmen nicht, oder nur sehr langsam. Am Beispiel der Internationalen Raumstation z.B. dann, wenn es notwendig wird, in einen höheren Orbit zu steigen, um beispielsweise Weltraumschrott auszuweichen. Die Daten von Space-Track werden deshalb maximal täglich aktualisiert. Selbst einige Tage alte Daten sind für eine im App-Rahmen genaue Positionsfeststellung ausreichend. Des Weiteren sind die zu übertragenden Datenmengen überschaubar. Eine Client-seitige Datenänderung findet nicht statt.

Monotones Lesen

Es wird davon ausgegangen, dass die Daten von den Providern zum Zeitpunkt der Abfrage konsistent sind. Eine Inkonsistenz kann nicht festgestellt werden und hat auch keinerlei Auswirkung auf das Laufzeitverhalten der App.

Frameworks und Dependencies

Im Folgenden sind alle verwendeten Frameworks und Abhängigkeiten angegeben

Framework / Dependency / Library	Verwendung	Grund
predict4java[6]	Bibliothek zur Berechnung der aktuellen Satellitendaten aus den Umlaufbahn-Daten (TLEs)	Alle Methoden für die Berechnung der Satellitendaten sind bereits vorhanden. Arbeitet zudem mit dem 3LE-Format. Somit keine Anpassungen notwendig
Google Play Services[7]	Google Maps Element in Kartendarstellung	Darstellung der Satelliten auf der Karte. Bekannte und verlässliche Kartendaten.
Apache Commons Lang[8]	Bibliothek verwendet von predict4java	predict4java
greenrobot's EventBus[9]	Realisierung der Ereignisbasierten Software-Architektur	Einfach zu implementieren. Threadübergreifend und anwendungsweit ohne weiteres nutzbar.
Square Inc's Retrofit[10]	http-Client-Framework, verwendet zum Implementieren der Daten-Provider-Klassen.	Kommunikationsparameter werden per Interface und Annotationen definiert. Somit einfaches Erstellen von Requests. Unterstützt asynchrone Operationen.

Finale Implementierung

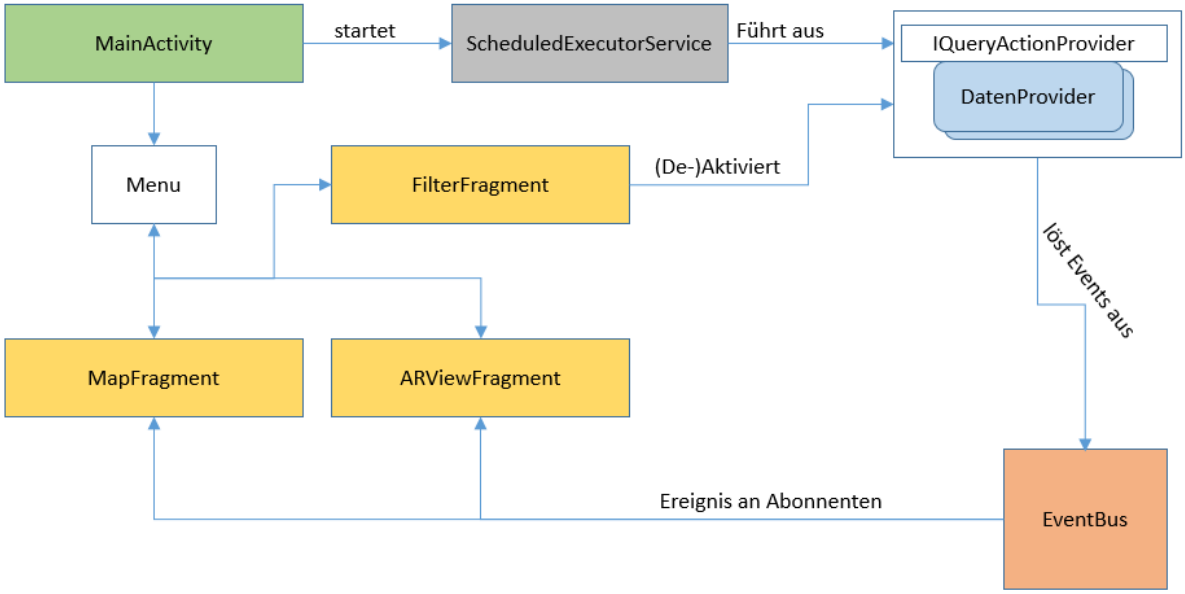

Abbildung 6 Vereinfachte Übersicht der wesentlichen Komponenten

Die MainActivity erstellt und befüllt den ScheduledExecutorService mit den Datenprovider-Komponenten. Die Datenprovider-Komponenten werden somit jeweils in einem separaten Thread ausgeführt. Die MainActivity stellt zudem eine Auflistung aller verfügbaren Datenprovider-Komponenten zur Verfügung, sodass das FilterFragment diese steuern kann.

Probleme

Während der Implementierung sind keine Probleme aufgetreten.

Ergebnis

Android-App

Alle Anforderungen konnten umgesetzt, die Anwendungsszenarien somit erfüllt werden. Im Folgenden die Oberflächen der App.

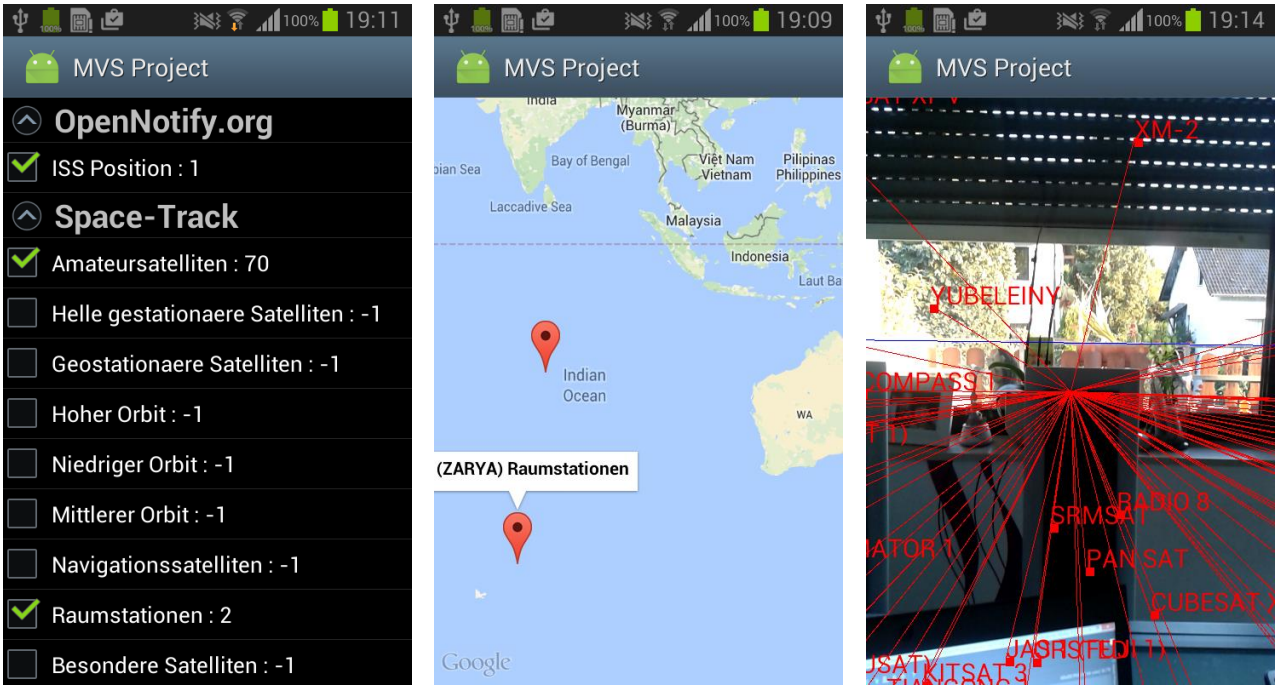

Abbildung 7 UI-Komponenten, Filter, Karten und AR-Ansicht

Technische Voraussetzungen

Allgemein:	- Internetverbindung - Android OS ab API-Level 15
AR-Element:	- aktive oder passive Ortung - Lagesensoren (Gyroskop, Beschleunigungsmesser, Kompass) - Kamera

Glossar

AR	Augmented Reality, erweiterte Realität
JSON	JavaScript Object Notation, Format zur Beschreibung von Objekten
TLE	Two-Line-Element, Ein Format zur Beschreibung von Umlaufbahnen
3LE	Three-Line-Element, TLE mit zusätzlichen Daten

Links

[1] https://www.space-track.org

[2] https://www.open-notify.org

[3] https://www.space-track.org/documentation%23api-requestClasses

[4] https://www.space-track.org/documentation#api-restOperators

[5] https://github.com/greenrobot/EventBus/blob/master/README.md

[6] https://github.com/badgersoftdotcom/predict4java

[7] https://developers.google.com/android/guides/overview

[8] https://commons.apache.org/proper/commons-lang/

[9] https://github.com/greenrobot/EventBus

[10] https://github.com/square/retrofit