Universität Ilmenau

Car - Android - Arduino

Entwurfsdokumentation CarDuinoDroid

Bearbeiter:
Behlau, Benjamin
Haueisen, Sven
Lewandowski, Benjamin
Lewandowski, Felix
Schulze, Christian
Strenger, Robin
Thorwirth, Paul
Vogel, Lars

Betreuer:
Dietrich, Thomas

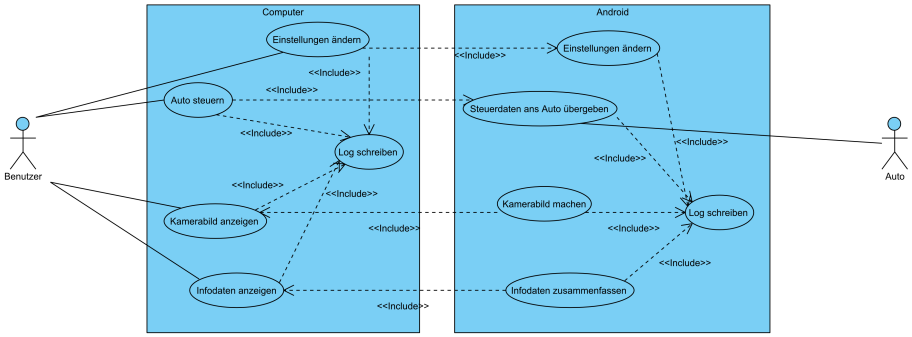
zuletzt geändert am 15. Mai 2012

Inhaltsverzeichnis

1 Einleitung

1.1 Anwendungsbereich

CarDuinoDroid dient der Fernsteuerung eines handelsüblichen Modellautos mittels eines Android-fähigen Mobiltelefons. Erkundungen und Informationsbeschaffungen sollen aus der Sicht einer Drohne ermöglicht werden. Besonders der Einsatz in schwer zugänglichem und unbekanntem Terrain aus sicherer Entfernung kann auf diese Weise realisiert werden.



1.2 Entwurfsziele

Korrektheit: Ist die Eigenschaft unseres Projektes den festgelegten Spezifikationen zu genügen.

Zuverlässigkeit: Nach der Installation soll das System problemlos laufen. Eventuell auftretende Fehler dürfen die Funktionsfähigkeit des Softwaresystems nichts beeinflussen.

Erweiterbarkeit: Der Aspekt der Weiterentwicklung steht im Kontext mit der Ausarbeitung des ersten Prototypen. Entsprechend dem Pflichtenheft sind unter anderem folgende Änderungen möglich:

- Automatische Bildqualitätsregulierung anhand der Verbindungsqualität
- Blitzlicht als Beleuchtung
- Verarbeitung von GPS-Daten (Wegstrecke, Einbindung auf Karte)

Veränderbarkeit: Denkbare Anpassungen des bestehenden Systems im Sinne der Anwenderfreundlichkeit sowie Datenmanagement sollen gewährleistet werden:

- Verbesserung der grafischen Oberfläche
- Einbindung einer Hilfe-Datei entpsrechend Erweiterungsgrad
- Anbindung Datenbank zur GPS-Datenspeicherung
- nachträgliche Fehlerbeseitigung

Verständlichkeit: Durch Dokumentation sowie das objektorientierte Programmierparadigma (Klassenaufbau) soll das Verständnis des Systems für nachfolgende Entwickler ermöglicht werden. Dazu wird der Quelltext beispielsweise mit javadoc oder ähnlichem dokumentiert.

Performanz: Eine effiziente Steuerung des Informationsflusses zwischen Android-Applikation und Java-Programm in Abhängigkeit der gewünschten Übertragungsmöglichkeiten soll gewährleistet werden.

Nutzerfreundlichkeit: Mit einer intuitiv gestalteten grafischen Oberfläche gewährleisten wir Bedienbarkeit und Interaktionsmöglichkeiten selbst für unerfahrene Nutzer. Besonders durch folgerichtige Fehlermeldungen und Rückinformationen erlangt der Anwender die Gelegenheit zur erneuten, angepassten Eingabe.

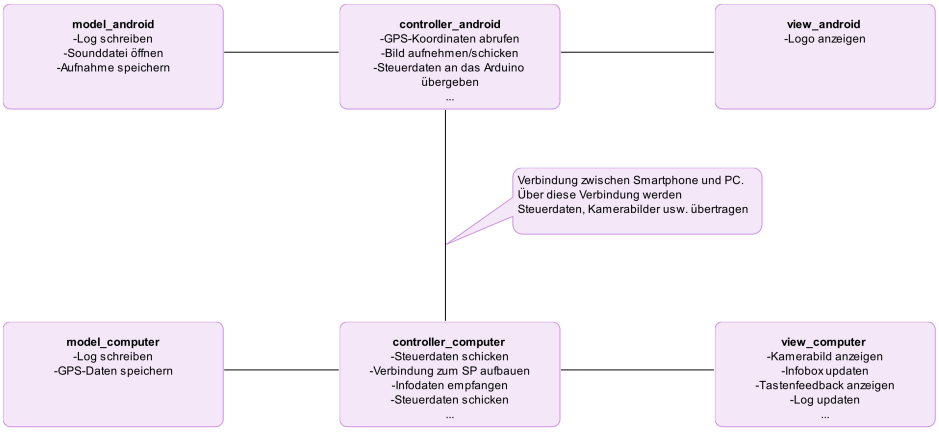
Portierbarkeit: Die Softwaresysteme sollen weitestgehend nur auf dem manifestierten Betriebssystem funktionsfähig sein.

2 Grobentwurf

2.1 Architekturmuster- und Zerlegung

2.1.1 Model-View-Control

Die Einheiten Datenmodell, Präsentation sowie Programmsteuerung vereinigen eine Architektur zur Strukturierung von Software-Entwicklungen. Im Mittelpunkt steht ein flexibler Programmentwurf, der inbesondere bei Änderungen und Erweiterungen Stärken zeigt. Mit reduzierter Komponentenabhängigkeit wird eine Wiederverwendung einzelner Systemkomponenten erreicht.



2.1.2 Begriffsbestimmung

Model Die Organisation und Aufbewahrung von verwendeten Datenmodellen (z.B. Datenbank, Speicherung von Eigenschaften etc.) wird mit dieser Modell-Einheit realisiert.

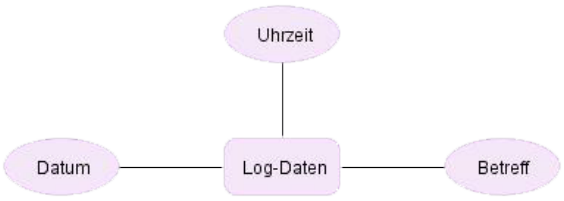
View Jegliche Darstellung benötigter Daten aus dem Modell sowie die Entgegennahme von Benutzerinteraktionen stellt den Zuständigkeitsbereich der Präsentation dar.

Controller Die Steuerung ist als Logikbaustein verantwortlich für den Programmablauf als auch die Verwaltung der vom View generierten Daten.

2.2 Management persistenter Daten

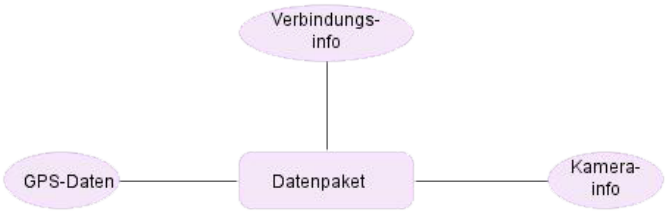
2.2.1 Log-Daten

Die Überwachung der Kontrollflüsse, besonders für Entwicklung und spätere Nutzung, stellt ein wichtiges Mittel zur Gewährleistung des Funktionsumfanges dar. Mit der Sammlung aller wichtiger Aktionen und ausgewählter Hintergrundaktionen besitzt dieses Werkzeug die Möglichkeit gewünschte aber ausbleibende Funktionen zu erkennen.



2.2.2 Informationsdaten

Das Dateiformat *CSV* (Comma-Separated Values) dient als Vorlage zum Speichern und Übermitteln der Informationspakete sowie sinnvollen Strukturierung des Datenbestandes.



Ein Beispiel für die Umsetzung der Datensendung veranschaulicht folgendes Bild:

```
1  10,9199;50,6912;EDGE;Rueck
2  Breitengrad  Längengrad
```

2.3 Sicherheit

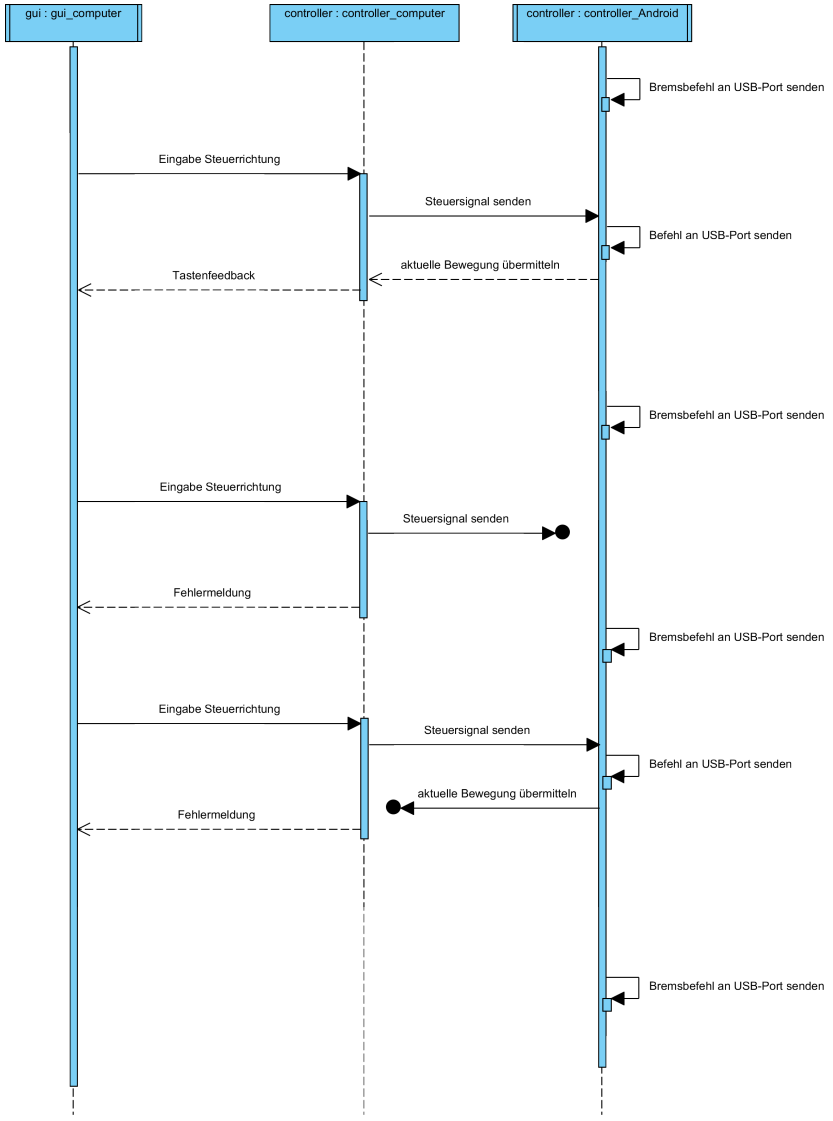
Mit den im Log gespeicherten Daten wird dem Nutzer ein Werkzeug zur Sicherung und Veranschaulichung des Funktionsumfanges der CarduinodroidSteuerung gegeben. Über seperat abgespeicherte Dateien wird zudem eine komplette Auswertung zu einem späteren Zeitpunkt ermöglicht.

2.4 Globaler Kontrollfluss

Das verwendete MVC-Modell ist Grundlage für den Kontrollfluss des Gesamtsystems. Die verwendeten Sequenzdiagramme stellen den groben Kommunikationsablauf zwischen den Elementen Model, View und Controller sowohl für die Android-Applikation als auch dem Computerprogramm dar. Es werden neben synchronen auch asynchrone Flüsse verwendet. Besonderer Hauptaugenmerk liegt bei den Kontrollflüssen auf der Server-Client-Verbindung zwischen dem Computerprogramms und der Android-Applikation.

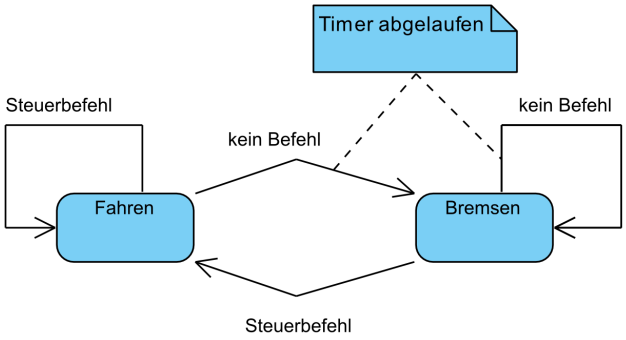
2.4.1 Steuerung

Der Controller des Java-Programms übermittelt die in der GUI eingegebenen Steuerdaten an den Controller der Android-Applikation. Diese sendet daraufhin den entsprechenden Befehl an den USB-Port. Anschließend erfolgt eine Bestätigung zurück an den Computer und wird als Tastenfeedback auf der Benutzeroberfläche ersichtlich. Die gewählte Richtung wird für einen bestimmten Zeitraum beibehalten. Bleiben weitere Steuersignale aus, sendet der Controller der Android-Applikation automatisch den Bremsbefehl an den USB-Port. Dieser Befehl kann erst durch neue ankommende Steuersignale unterbrochen werden. Diese automatische Bremse dient der Sicherheit. Sollte eine Zeitüberschreitung bei der Antwort zum Computer-Controller vorliegen, wird automatisch eine Information ausgegeben, die den Nutzer darüber informiert, dass ein Fehler aufgetreten ist.



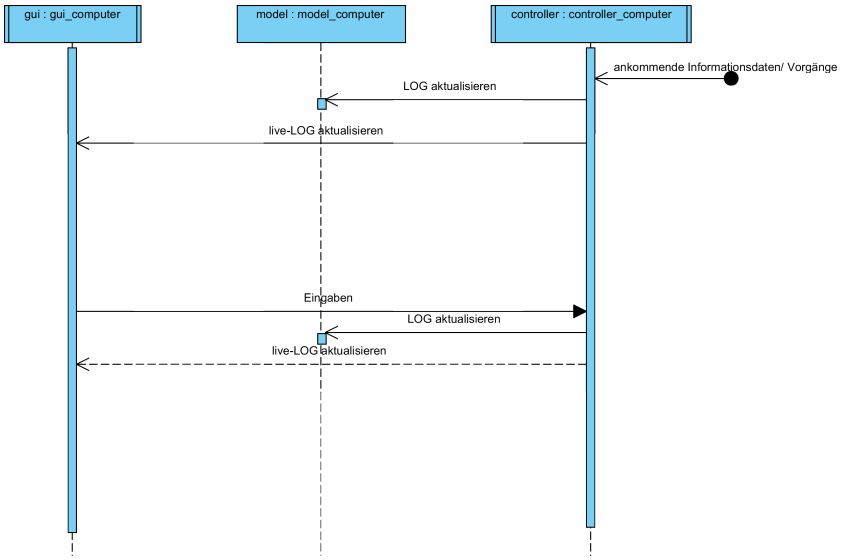
2.4.2 Steuerungszustand

Entsprechend der *Steuerung* wird mit dem folgenden Diagramm der Wechsel der Zustände durch Steuerbefehle verdeutlicht.



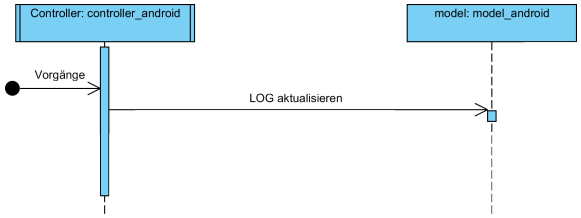
2.4.3 Log-Computer

Sämtliche Vorgänge (eingehende und ausgehende Befehle und Signale, sowie Benutzereingaben) werden in einer Logdatei auf dem Computer gespeichert und auf der GUI zusätzlich live ausgegeben.



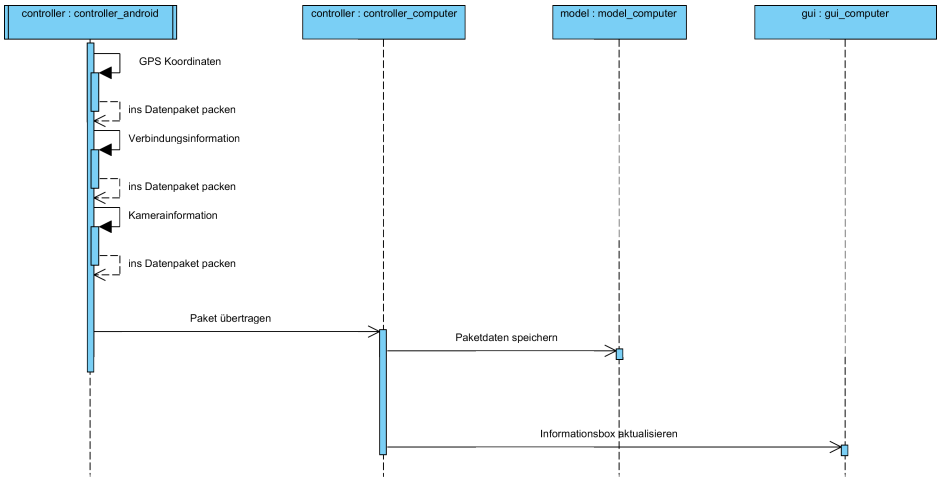
2.4.4 Log-Android

Sämtliche Vorgänge (eingehende und ausgehende Befehle und Signale) werden in einer Logdatei auf dem Android-Smartphone gespeichert.



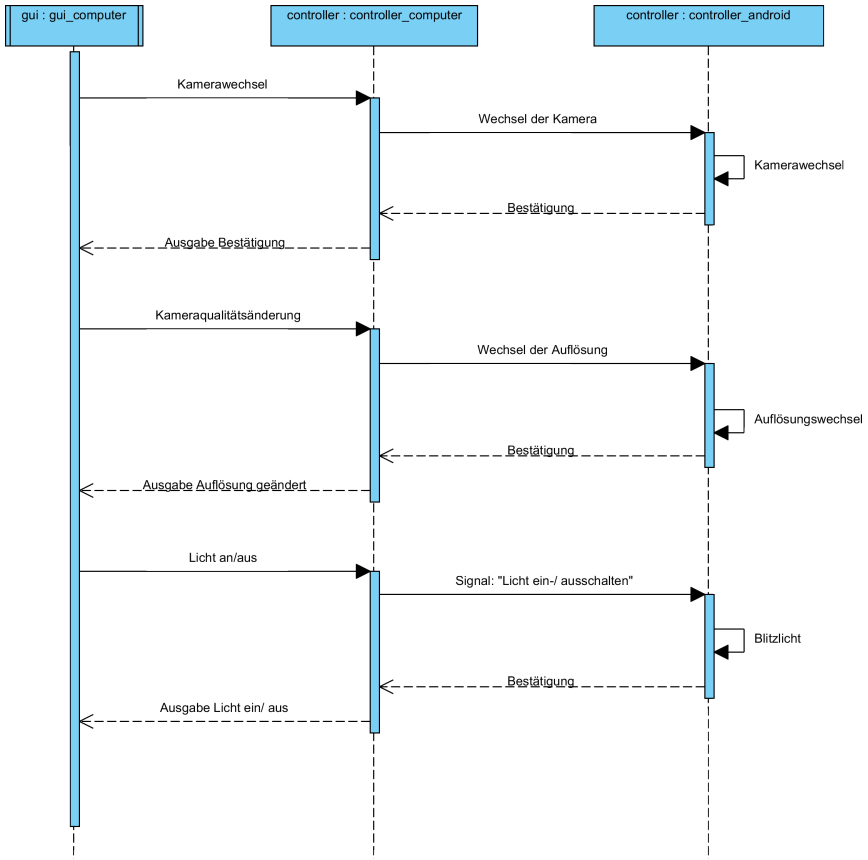
2.4.5 Informationsübertragung

Der Controller der Android-Applikation holt sich die verschiedenen Statusdaten aus den einzelnen Funktionsmodulen (GPS-Koordinaten, Verbindungsinformationen, Kamerainformationen) und fasst sie zu einem Paket zusammen. Dieses Datenpaket wird dann über das Netzwerk zum Computer-Controller übertragen und die einzelnen Paketdaten auf dem Computer gespeichert. Außerdem werden die Informationen, die die Informationsbox der grafischen Benutzeroberfläche anzeigt, aktualisiert. Der gesamte Vorgang wiederholt sich ständig.



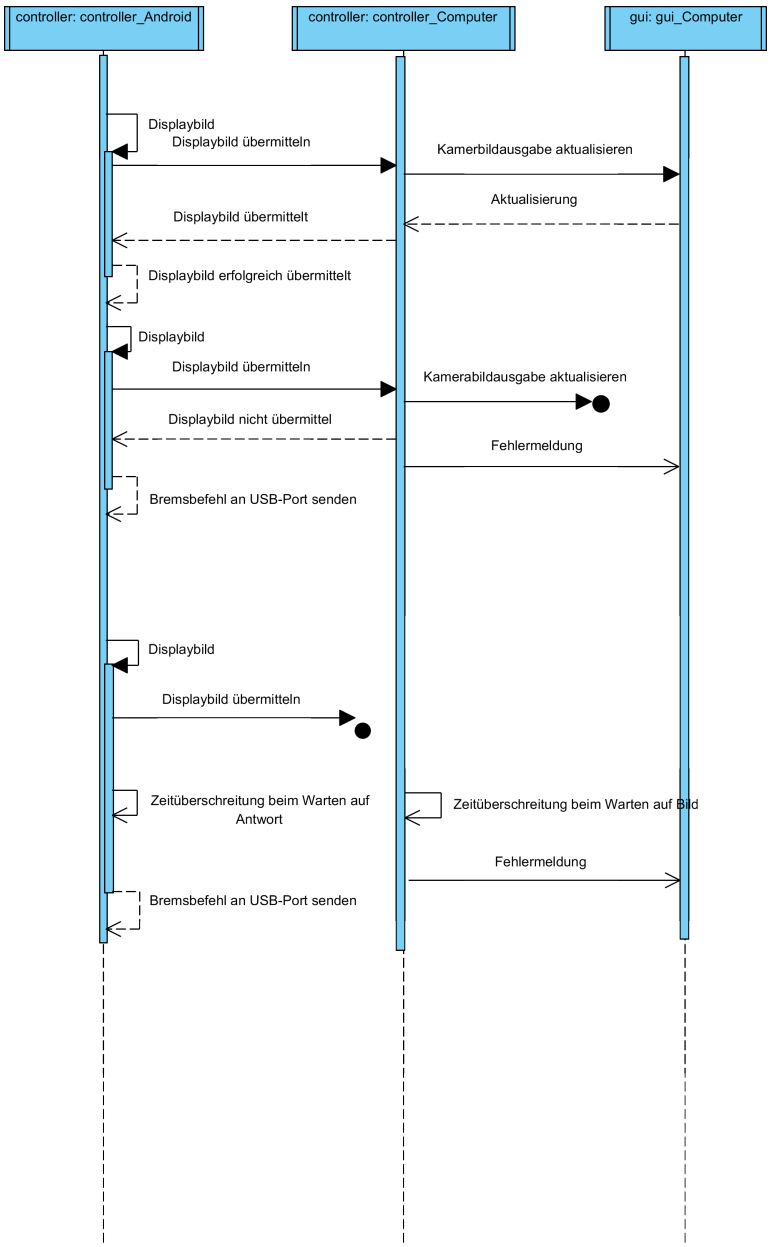
2.4.6 Kameraeinstellung

Eingaben auf der Benutzeroberfläche zur Kamera (Kamerawechsel, Qualitätsänderungen, Licht an/aus) werden über das Netzwerkmodul des Computer-Controllers an den Controller der Android-Applikation übermittelt. Dieser wiederrum kommuniziert mit dem Kameramodul und setzt die Nutzerwünsche um.



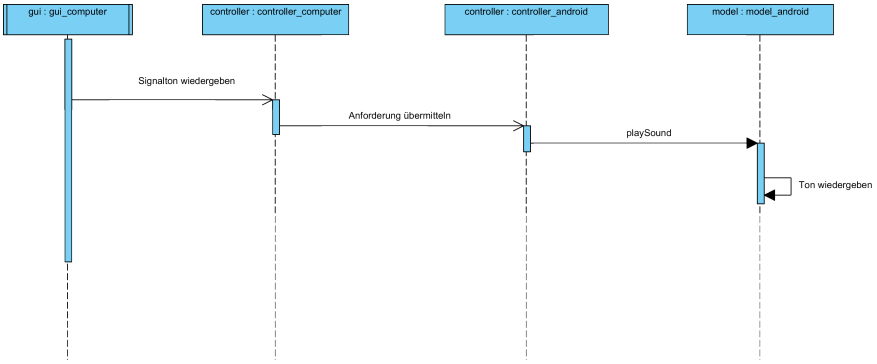
2.4.7 Kamerabild

Das Kamerabild der Smartphone-Kamera wird ständig durch den Controller ausgelesen und über das Netzwerk-Modul an den Computer übermittelt. Dieser gibt das entsprechende Bild durch Aktualisierung auf der GUI aus. Passende Bestätigungen sorgen so für eine Live-Übertragung. Sollte es Fehler bei Ausgabe oder Übermittlung geben, wird das Auto gestoppt und eine entsprechende Fehlermeldung ausgegeben.



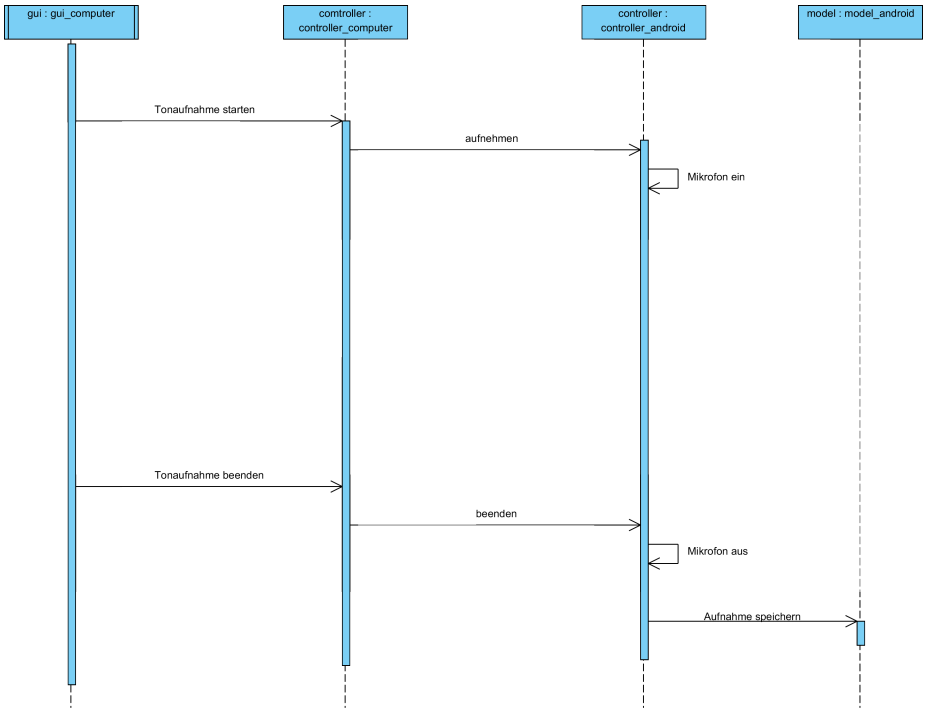
2.4.8 Tonausgabe

Wird auf der Benutzeroberfläche der Signaltonbutton betätigt, übermittelt der Computer-Controller diese Anforderung dem Smartphone. Der Controller des Mobiltelefons ruft daraufhin die dafür vorher eingerichtete Sounddatei auf und gibt sie über die Lautsprecher des Smartphones wieder.



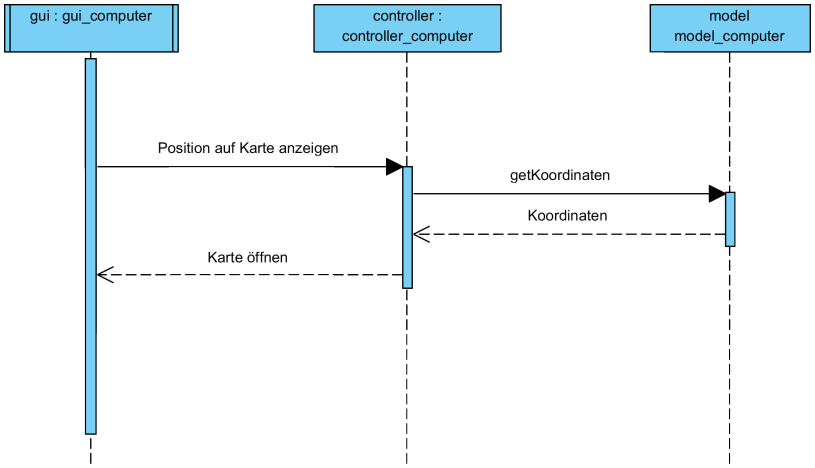
2.4.9 Tonaufnahme

Durch das Betätigen eines Schalters auf dem GUI wird dies über das Netzwerk dem Mobiltelefon mitgeteilt. Der Android-Controller sorgt daraufhin für die Aktivierung des Mikrofones und beginnt die Aufnahme. Kommt der Befehl zum Stoppen der Tonaufnahme, wird diese beendet und auf dem Smartphone gespeichert.



2.4.10 GPS-Karte

Sollte der Benutzer die Koordinaten der Drohne auf der Karte sehen wollen, wird dies dem Controller des Java-Programmes mitgeteilt. Dieser wiederrum holt sich die entsprechenden Koordinaten aus dem Speicher (model). Danach öffnet sich dem Benutzer die entsprechende Karte.



2.5 Randbedingungen

Das System besteht einerseits aus einer Android-Applikation andererseits aus einem Java-PC-Programm. Die Android-Applikation soll:

- ab der Version 2.3.3 von Android laufen.
- Befehle vom Java-Programm entgegennehmen und diese zum Arduino-Controller weiterleiten.
- Kamerabilder an das Java-Programm schicken.
- Statusinformationen, wie GPS oder Verbindungsqualität, an das Java-Programm übertragen.
- sowohl im WLAN als auch im mobilen Internet funktionstüchtig sein.
- keine aufwendige Oberfläche besitzen.
- im Google-Play-Store verfügbar sein.

Das Java-Programm soll:

- eine Verbindung zum Android-Mobiltelefon aufbauen, nachdem die Android-Applikation gestartet wurde.
- Steuerinformationen an das Android-Smartphone schicken.
- eine Oberfläche besitzen, die aus folgenden Komponenten besteht: Kamerabild, Informationsbox mit Statusinformationen des Mobiltelefon, Log und dem Tastenfeedback der Steuertasten.
- auf jedem Computer laufen, wo Java in Version 1.7 oder höher installiert ist.
- eine Einstellmöglichkeit haben, mit der man die Bildqualität ändern kann.

Sonstiges:

- Es soll eine Logfunktion für die Android-Applikation und für das Java-Programm vorhanden sein.
- Das System soll leicht erweiterbar sein.
- Der Arduino-Controller muss nicht programmiert werden.
- Beide Komponenten müssen sich im gleichen Netzwerk aufhalten, damit das System funktioniert.
- Der Source-Code für beide Einzelprogramme (Android-Applikation und Java-Programm) soll mitsamt Dokumentation und einer Anleitung, zur Erweiterung des Systems um neue Funktionen, auf einer frei zugänglichen Plattform, wie zum Beispiel www.sourceforge.net, unter einer Open-Source-Linzenz verfügbar sein.

3 Feinentwurf

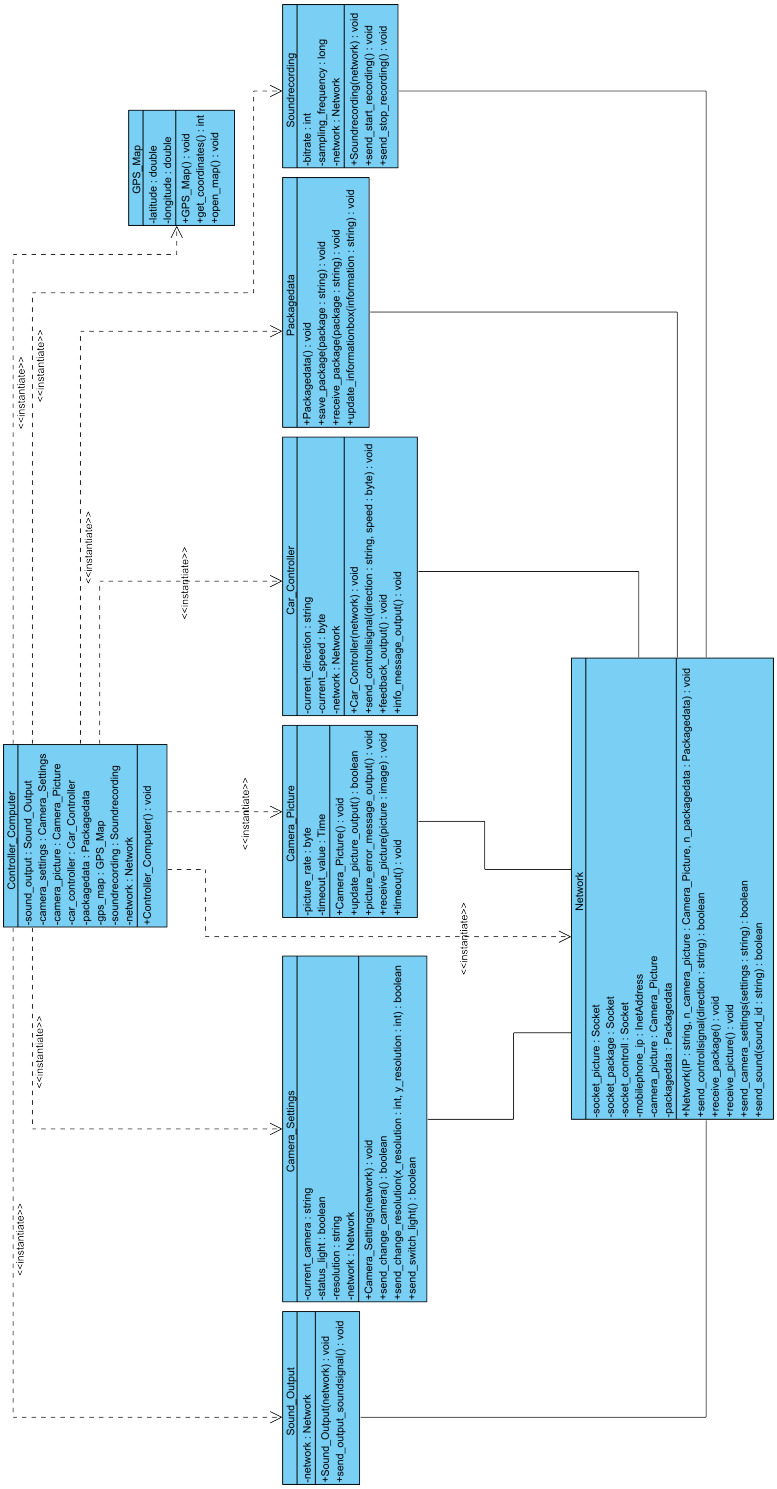
3.1 Richtlinien der Dokumentation

Entsprechend den gesteckten Zielen der Modularität sowie leichte Erweiterbarkeit wird für eine umfassende Dokumentation des Quelltextes gesorgt. Bei der Wahl eines entsprechenden Werkzeuges wird keine konkrete Festlegung vorgenommen, aufgrund der Vielzahl von Hilfsmittel. Javadoc ist eine mögliches Software-Dokumentationswerkzeug, welches aus unseren Java-Quelltexten automatisch HTML-Dokumentationsdateien erstellen kann.

3.2 Klassen und Schnittstellen

3.2.1 Controller (Computerprogramm)

Der Computer-Controller ist die zentrale Steuereinheit des Computerprogramms von Carduinodroid. Er besteht aus mehreren Funktionsmodulen.
Dieser modulare Aufbau soll mithilfe von Klassen realisiert werden. Jede Klasse ist damit als ein Funktionsmodul für einen Typ von Aufgaben zuständig. Soll das Programm später erweitert werden, muss für eine neue Funktion ein neues Modul, also eine neue Klasse, implementiert werden.


Controller_Computer
-sound_output : Sound_Output
-camera_settings : Camera_Settings
-camera_picture : Camera_Picture
-car_controller : Car_Controller
-packagedata : Packagedata
-gps_map : GPS_Map
-soundrecording : Soundrecording
-network : Network
+Controller_Computer() : void
GPS_Map
-latitude : double
-longitude : double
+GPS_Map() : void
+get_coordinates() : int
+open_map() : void
Soundrecording
-bitrate : int
-sampling_frequency : long
-network : Network
+Soundrecording(network) : void
+send_start_recording() : void
+send_stop_recording() : void
Packagedata
+Packagedata() : void
+save_package(package : string) : void
+receive_package(package : string) : void
+update_informationbox(information : string) : void
Car_Controller
-current_direction : string
-current_speed : byte
-network : Network
+Car_Controller(network) : void
+send_controllsignal(direction : string, speed : byte) : void
+feedback_output() : void
+info_message_output() : void
Camera_Picture
-picture_rate : byte
-timeout_value : Time
+Camera_Picture() : void
+update_picture_output() : boolean
+picture_error_message_output() : void
+receive_picture(picture : image) : void
+timeout() : void
Network
-socket_picture : Socket
-socket_package : Socket
-socket_controll : Socket
-mobilephone_ip : InetAddress
-camera_picture : Camera_Picture
-packagedata : Packagedata
+Network(IP : string, n_camera_picture : Camera_Picture, n_packagedata : Packagedata) : void
+send_controllsignal(direction : string) : boolean
+receive_package() : void
+receive_picture() : void
+send_camera_settings(settings : string) : boolean
+send_sound(sound_id : string) : boolean
Camera_Settings
-current_camera : string
-status_light : boolean
-resolution : string
-network : Network
+Camera_Settings(network) : void
+send_change_camera() : boolean
+send_change_resolution(x_resolution : int, y_resolution : int) : boolean
+send_switch_light() : boolean
Sound_Output
-network : Network
+Sound_Output(network) : void
+send_output_soundsignal() : void
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>
<<instantiate>>

Klassen- und Methodenbeschreibung

Der Controller besteht aus einer Klasse „Controller_Computer“ welche die anderen Klassen („Sound_Output“, „Camera_Picture“, „Camera_Settings“, „Car-_Controller“, „Packagedata“, „Soundrecording“, „GPS_Map“und „Network“) als Instanzen aufruft.

Controller_Computer

„Controller_Computer“dient als Schnittstelle zwischen dem GUI und den einzelnen Funktionen. Sämtliche Eingabe-Methoden des GUI kennen nur den Controller_Computer, nicht die einzelnen benötigten Funktionen. Damit ist es leichter spätere Programmerweiterungen hinzuzufügen. Diese müssen damit nur noch als eigene Instanzklassen implementiert werden. Das verwendete Prinzip nennt sich auch „Fassaden-Entwurfsmuster“. Ähnlich dem Singleton-Entwurfmuster soll der Controller_Computer zur Fehlervermeidung jeweils nur eine Instanz der Funktionen aufrufen. Als Attribute enthält „Controller_Computer“sämtliche Instanzklassen (*sound_output*, *camera_settings*, *camera_picture*, *car_controller*, *packagedata*, *gps_map*, *soundrecording* und *network*). Folgende Methode ist enthalten:

Controller_Computer() ist der Konstruktor der Klasse.

Sound_Output

Diese Klasse ist dafür verantwortlich, dass die entsprechende Nutzereingabe das Abspielen eines auf dem Mobiltelefon eingespeicherten Signaltons zur Folge hat. Das Attribut *network* bezieht sich auf die Network-Klasse. „Sound_Output“ ist ein Konstruktur von „Controller_Computer“ und besitzt die folgende Methoden:

send_output_soundsignal() übermittelt dem Android-Controller über das Netzwerk die Aufforderung, eine Sounddatei abzuspielen. Sind mehrere Signaltöne eingespeichert, kann auch der Name der gewünschten Datei mit gesendet werden. Die Methode enthält kein Rückgabewert.

Sound_Output(network) ist der Konstruktor der Klasse.

Camera_Picture

„Camera_Picture“ dient dem Empfang der Video-Bilder, welche von der Smartphone-Kamera aufgenommen werden. Weiterhin ist diese Klasse für die Aktualisierung der Bilder, die dem Nutzer angezeigt werden, zuständig. Der Controller empfängt in bestimmten Intervallen die Bilder vom Mobiltelefon und stellt diese dem Nutzer zur Verfügung. Sollte es dabei zu Fehlern kommen, müssen sowohl der Nutzer als auch das Smartphone darüber in Kenntnis gesetzt werden. Um dies zu gewährleisten gibt es die Attribute *picture_rate* und *timeout_value*. *Picture_rate* ist die Frequenz, in der Bilder gesendet werden (z.B. 5 Bilder pro Sekunde). *timeout_value* ist die Zeit, die der Controller auf ein Bild wartet ohne eine Fehlermeldung auszugeben. Ist dieser Wert überschritten, wird das

Nicht-Empfangen eines Bildes als Fehler gewertet. „Camera_Picture“ ist ein Konstruktor von „Controller_Computer“ und besitzt die folgenden Methoden:

update_picture_output() ist für die Aktualisierung der angezeigten Bilder auf der GUI zuständig. Der Rückgabewert ist eine bool'sche Variable. Ist der Wert des Rückgabewertes „true“ weiß der Controller, dass das Bild erfolgreich angezeigt wurde.

picture_error_message_output() ist eine Methode ohne Rückgabewert und ist dazu da, den Anwender zu informieren, wenn Fehler bei der Bildübertragung aufgetreten sind.

receive_picture(picture:image) ist eine Funktion, um die Bilder vom Android-Mobiltelefon zu empfangen. Sie liefert keinen Rückgabewert.

timeout() zählt einen Timer runter. Ist in dieser Zeit kein Bild empfangen worden, wird ein Fehler ausgegeben. *Timeout()* liefert keinen Rückgabewert.

Camera_Picture() ist der Konstruktor der Klasse.

Camera_Settings

Die Klasse „Camera_Settings“ ist für Änderungen der Kameraeinstellungen am Android-Smartphone zuständig. Dazu zählen das Wechseln der verwendeten Kamera (Vorder- oder Rückkamera), Änderungen der Bildqualität (Auflösung) sowie das Einschalten des Blitzlichtes als Beleuchtung. Attribute der Klasse sind *current_camera* (aktuell verwendete Kamera), *status_light* (Licht an = true, Licht aus = false), *resolution* (derzeitige eingestellte Auflösung) und *network* (bezieht sich auf die Klasse „Network“). „Camera_Settings“ ist ein Konstruktor von „Controller_Computer“ und enthält die folgenden Methoden:

send_change_camera() ist eine Methode mit einer bool'schen Variable als Rückgabewert. Mit *send_change_camera()* sendet der Computer-Controller dem Android-Telefon das Signal, die Kamera zu wechseln. Bei Erfolg ehält sie ein „true“ als Rückgabewert.

send_change_resolution() sendet dem Android-Telefon den Befehl die Auflösung der verwendeten Kamera auf den gewünschten Wert zu stellen. Dazu ist eine entsprechende Übergabe-Variable nötig. Nach erfolgreichem Einstellen der Auflösung erhält *send_change_resolution()* ein „true“ als bool'schen Rückgabewert.

send_switch_light() befiehlt dem Andoid-Smartphone das Ein- bzw. Ausschalten des Blitzlichtes. Bei Erfolg wird ein „true“ als bool'scher Rückgabewert empfangen.

Camera_Settings(network) ist der Konstruktor der Klasse.

Car_Controller

Der „Car_Controller“ ist für die Umsetzung der Eingaben des Anwenders zur Steuerung der Drohne verantwortlich. Er übermittelt Steuersignale, gibt die aktuelle Fahrtrichtung in Form eines Tastenfeedbacks an und informiert über aufgetretene Fehler. Er ist ein Konstruktor von „Controller_Computer“ und besitzt die Attribute *current_direction* (aktuelle Steuerungsrichtung) und *current_speed* (aktuelle maximale Geschwindigkeit) sowie *network* (bezieht sich auf die Klasse „Network“). Folgende Methoden sind enthalten:

send_controllsignal() ist für die Übermittlung der eingegebenen Steuerrichtung an den Android-Controller zuständig. Dazu werden die gewünschte Richtung sowie die Geschwindigkeit als Parameter übergeben. Diese Methode liefert keinen Rückgabewert.

feedback_output() ist eine Methode ohne Rückgabewert, welche dem Nutzer die aktuelle Bewegung des Fahrzeuges als Tastenfeedback mitteilt.

info_message_output() setzt den Nutzer über Fehler bei der Übermittlung der Steuersignale in Kenntnis. Diese Methode liefert keinen Rückgabewert.

Car_Controller(network) ist der Konstruktor der Klasse.

Packagedata

Der Konstruktor „Packagedata“ (von „Controller_Computer“) ist für die Verarbeitung der vom Mobiltelefon übergebenen Pakete zuständig, welche die Statusdaten GPS-Koordinaten, Verbindungsinformation und Kamerainformationen enthalten. „Packagedata“ speichert die Paketinformationen in einer externen Datei ab und informiert den Nutzer mithilfe der Informationsbox über die verschiedenen Statusdaten. „Packagedata“ enthält folgende Methoden:

save_package() speichert die Paketinformationen in einer externen Datei und liefert dabei keinen Rückgabewert. Übergeben wird das jeweilige Paket.

receive_package() sorgt dafür, dass der Controller Pakete als String empfangen kann. Diese Methode besitzt keinen Rückgabewert.

update_informationbox() aktualisiert die Informationsbox, die der Anwender sieht, mit den Informationen, welche durch get_information() zurückgegeben werden. *update_informationbox()* selber liefert keinen Rückgabewert.

Packagedata() ist der Konstruktor der Klasse.

GPS_Map

„GPS_Map“ ist ein Konstruktor von „Controller_Computer“ und für die Darstellung der GPS-Position des Autos auf einer Karte zuständig. Enthaltene Attribute sind die Koordinaten in Breitengrad (*latitude*) und in Längengrad (*longitude*). Implementierte Methoden sind:

get_coordinates() liest die in einer Datei gespeicherten Koordinaten aus und erhält sie in Form eines Integer-Wertes zurück.

open_map() zeigt die von *get_coordinates* erhaltenen Daten auf einer Karte an. Diese Methode liefert keinen Rückgabewert.

GPS_Map() ist der Konstruktor der Klasse.

Soundrecording

„Soundrecording“ ist für das Ein- und Ausschalten des Android-Mikrofons zuständig, womit eine Tonaufnahme realisiert wird. „Soundrecording“ ist ein Konstruktor von „Controller_Computer“ und enthält die Bitrate (*bitrate*), die Abtastfrequenz (*sampling_frequenz*) der Tonaufnahme und die eingebundene „Netzwerk“Klasse (*network*) als Attribute. Folgende Methoden werden verwendet:

send_start_recording() übermittelt dem Android-Controller die Nutzereingabe, das Mikrofon des Mobiltelefons einzuschalten. Diese Methode liefert keinen Rückgabewert.

send_stop_recording() sendet dem Android-Controller die Nutzereingabe, das Mikrofon des Smartphones abzuschalten. Die Methode liefert dabei keinen Rückgabewert.

Soundrecording(network) ist der Konstruktor der Klasse.

Network

„Network“ stellt die Kommunikation mit dem Netzwerk dar. Dies beinhaltet das Empfangen und Weiterleiten der Informationen vom Android-Mobiltelefon. Folgende Attribute sind festgelegt: *socket_picture* (Sockel zum Empfangen der Videobilder), *socket_package* (Sockel zum Empfangen der Statuspakete), *socket_control* (Sockel über den die Kommunikation zur Steuerung der Drohne erfolgt) und die Adresse des Android-Mobiltelefons in Form einer IP Adresse (*mobilephone_ip*). Über *packagedata* und *camera_picture* sind die Klassen „Packagedata“ und „Camera_Picture“ eingebunden. Zu implementierende Methoden sind:

Network(IP: string, n_camera_picture: Controller_Computer.Camera-Picture, n_packagedata: Controller_Computer.Packagedata) ist der Konstruktor der Klasse.

send_controllsignal() ist eine Methode um die gewünschte Fahrtrichtung an das Mobiltelefon weiterzugeben. Sie empfängt diese vom Car_Controller. Als Bestätigung zum Empfang der Daten beim Smartphone soll eine bool'sche Variable als Rückgabewert dienen.

receive_package() ist für das Empfangen der Statuspaketdaten zuständig. Diese Methode liefert keinen Rückgabwert.

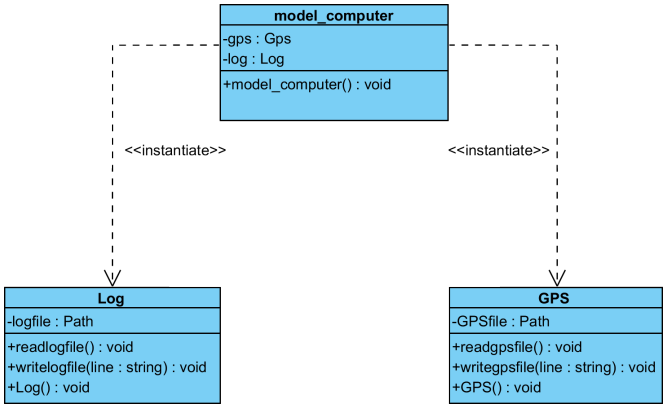
receive_picture() empfängt ständig die vom Smartphone gesendeten Bilder und gibt diese an die Klasse Camera_Picture weiter. *receive_picture()* liefert keinen Rückgabewert.

send_camera_settings() muss die, von Camera_Settings, erhaltenen Informationen an das Android-Gerät senden. Die gewünschten Daten werden dazu als String übergeben. Ein bool'scher Rückgabewert gibt Auskunft, über den Empfang auf Android-Seite.

send_sound() ist eine Funktion, die den Wunsch zur Wiedergabe eines Signaltons an das Android-Mobiltelefon weiterleitet. Dazu wird die ID der Sounddatei übergeben. Hat das Smartphone den Signalton abgespielt, wird „true“ zurückgegeben.

3.2.2 Model (Computerprogramm)

Das *model:computer* ist für das Lesen und Schreiben des Logs zuständig. Dies unterteilt sich in die Log-Datei und die GPS-Datei.



Das Modell besteht aus der Klasse „model_computer“ heißt. Hierbei handelt es sich um 2 Konstruktoren, zum Einen der Log und zum Anderen das GPS.

Klassen- und Methodenbeschreibung

model_computer

Diese Steuerklasse (Manager) vereint jegliche Funktionen des „model_computer“.

Log

Diese Klasse ist für das Einlesen und Schreiben der Logdatei zuständig. Sie besitzt folgende Methoden:

readlogfile(): void ist die Methode zum Einlesen der Logdatei.

writelogfile(line: string): void ist für das Schreiben in die nächste Zeile verantwortlich.

GPS

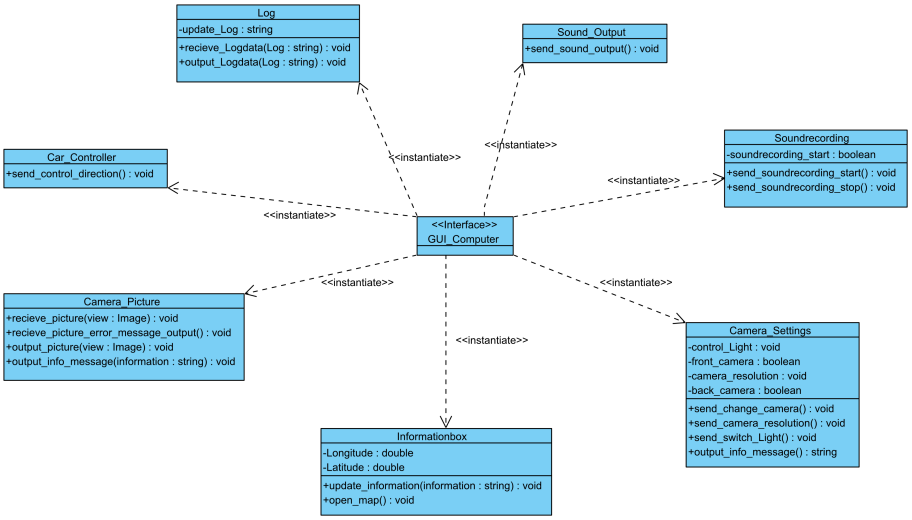
Zur optionalen Speicherung von GPS-Daten in einer seperaten Datei um eine Wegstrecke verdeutlichen zu können. Dazu werden folgende Methoden benötigt:

readgpsfile(): void ist die Methode zum Lesen der GPS-Datei.

writegpsfile(line: string): void entspricht in seinem Funktionsumfang der Methode *writelogfile(line: string): void*.

3.2.3 View (Computerprogramm)

Der view_computer dient dazu dem Nutzer die Funktionen des Systems bereitzustellen. Einige Funktionen können auf Wunsch aktiviert und wieder abgeschaltet werden. Der view_computer ist in einzelne Klassen aufgeteilt. Je Funktion existiert somit eine eigene Klasse. Weitere Funktionen können durch einfügen von Klassen einfach hinzugefügt werden.



Klassen- und Methodenbeschreibung

Car_Controller

Diese Klasse dient zum Übermitteln der Steuerdaten, die vom Nutzer eingegeben werden, zum Visualisieren des Tastenfeedbacks sowie zum Ausgeben von Informationsmeldungen bei auftretenden Fehlern. Folgende Methode soll dies umsetzen:

send_control_direction() dient zum Übermitteln der eingegebenen Steuerdaten an den Controller_Computer. Sowohl Daten zur Richtungsbestimmung als auch zum Beschleunigen und Abbremsen des Fahrzeugs werden gesendet.

Camera_Picture

Mit dieser Klasse soll das Anzeigen des Kamerabildes des Mobiltelefons realisiert werden um eine Koordination des Fahrzeugs zu ermöglichen. Bei auftretenden Fehlern soll der Nutzer entsprechend informiert werden. Folgende Methoden sind dazu enthalten:

recieve_picture() dient zum Empfangen des Kamerabildes, welches vom Controller_Computer gesendet wird.

recieve_picture_error_message_output() soll im Falle eines Fehlers die entsprechende Fehlermeldung empfangen.

output_picture(view: Image) übernimmt das Darstellen der vom Smartphone gesendeten Bilddaten auf dem Computerbildschirm.

output_info_message(information: string) gibt im Falle eines Fehlers die vorher übermittelte Fehlermeldung aus um den Nutzer darüber zu informieren.

Informationbox

Die Informationbox dient dazu wichtige Informationen über das System anzuzeigen, welche in einem festgelegten Intervall aktualisiert werden. Hier sollen unter anderem Informationen über die Kamera des Smartphones, als auch GPS-Daten, in Form von *Longitude: double* und *Latitude: double*, zur Verfügung gestellt werden. Folgende Methoden sind enthalten:

update_information(information: string) soll zum Empfangen und Verarbeiten der gesendeten Datenpakete und anschließendem Ausgeben in der dafür vorgesehenen Informationsbox dienen.

open_map stellt die Möglichkeit bereit die gegebenen GPS-Daten auf einer Karte anzeigen zu lassen um den Standort des Fahrzeuges genau bestimmen zu können.

Camera_Settings

Diese Klasse dient dazu dem Nutzer verschiedene Einstellungen bezüglich des angezeigten Kamerabildes über die Attribute *control_Light*, *front_camera*, *back_-*

camera sowie *camera_resolution* zu ermöglichen. Sie soll es ermöglichen die Kameraauflösung zu ändern, zwischen den beiden Kameras des Mobiltelefons zu wechseln, sowie das Licht der Kamera ein- und auszuschalten. Dies soll über folgende Methoden umgesetzt werden:

Send_change_camera() sendet den Befehl zum Wechseln zwischen den Kameras an den „controller_computer“.

Send_camera_resolution() sendet den Befehl zum Ändern der Kameraauflösung an den „controller_computer“.

Send_switch_Light() teilt dem *controller_computer* mit, dass das Licht an- oder ausgeschaltet werden soll.

output_info_message() informiert den Nutzer über entstandene Fehler bei dem Versuch Einstellungen vorzunehmen.

Soundrecording

Soundrecording ist eine Klasse die dem Nutzer ermöglichen soll Ton mithilfe des Mobiltelefons über das Attribut *soundrecording_start* aufzunehmen. Diese Klasse enthält folgende Methoden:

Send_soundrecording_start() übermittelt dem „controller_computer“den Befehl zum Starten der Tonaufnahme.

Send_soundrecording_stop() übermittelt dem „controller_computer“der Befehl zum Stoppen der Tonaufnahme.

Sound_Output

Der Nutzer soll in der Lage sein mithilfe des Systems einen Ton am Mobiltelefon ausgeben zu lassen. *Sound_output* soll dies ermöglichen, mit folgender Methode:

Send_sound_output() sendet den Befehl zum Ausgeben der vorher festgelegten Audiodatei an den „controller_computer“.

Log

Mit der Klasse Log soll die komplette Kommunikation zwischen dem Computer und dem Mobiltelefon nachvollziehbar gemacht werden. Dazu soll die Klasse die Informationen in eine Datei speichern und zusätzlich alle neuen Informationen im Log-Fenster der GUI aktualisieren. Das Aktualisieren der Daten erfolgt über das Attribut „update_Log“. Die Klasse enthält folgende Methoden:

recieve_Logdata(Log: string) dient zum Empfangen und abspeichern der von den jeweiligen Klassen gesendeten Kommunikationsinformationen.

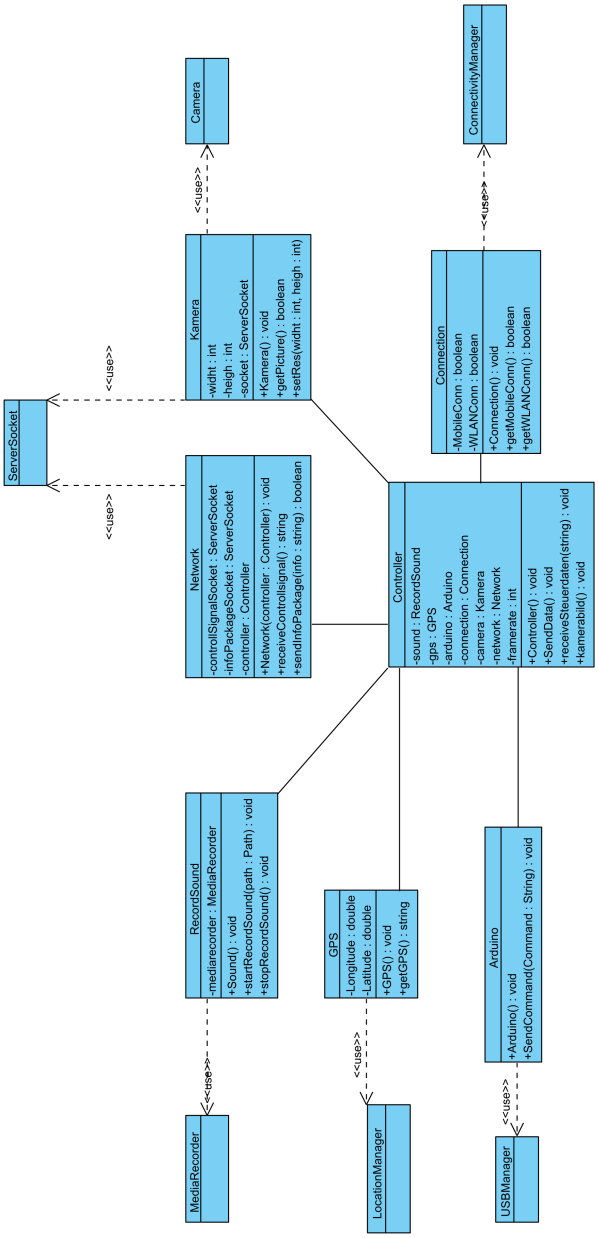
Output_Logdata(Log: string) soll das Log-Fenster der GUI mit aktuellen Informationen versorgen.

Prototyp Oberfläche

Die Oberfläche wird ein wichtiges Element während der Arbeit zum funktionsfähigen Programm sein. Besonders die intuitive Steuerung sowie Handhabung aller wichtigen Elemente stellt eine Herausforderung dar. Der nachfolgende Prototyp beinhaltet in groben Zügen eine mögliche Umsetzungsvariante.

3.2.4 Controller (Android-Applikation)

Der Controller ist für die Steuerung der Android-Funktionen und des Arduino-Boards zuständig. Dabei gibt es für jede benutzte Handykomponente eine eigene Klasse. Des Weiteren gibt es die Klasse „Controller“, die für die Steuerung der einzelnen Komponenten zuständig ist. Dieser Aufbau soll dabei helfen das System zu erweitern, weil nur in wenigen bestehenden Klassen etwas geändert werden muss.


Camera
ConnectivityManager
<<use>>
<<use>>
Kamera
-widht : int
-heigh : int
-socket : ServerSocket
+Kamera() : void
+getPicture() : boolean
+setRes(widht : int, heigh : int)
Connection
-MobileConn : boolean
-WLANConn : boolean
+Connection() : void
+getMobileConn() : boolean
+getWLANConn() : boolean
<<use>>
ServerSocket
<<use>>
Network
-controllSignalSocket : ServerSocket
-infoPackageSocket : ServerSocket
-controller : Controller
+Network(controller : Controller) : void
+receiveControlsignal() : string
+sendInfoPackage(info : string) : boolean
Controller
-sound : RecordSound
-gps : GPS
-arduino : Arduino
-connection : Connection
-camera : Kamera
-network : Network
-framerate : int
+Controller() : void
+SendData() : void
+receiveSteuerdaten(string) : void
+Kamerabild() : void
RecordSound
-mediarecorder : MediaRecorder
+Sound() : void
+startRecordSound(path : Path) : void
+stopRecordSound() : void
GPS
-Longitude : double
-Latitude : double
+GPS() : void
+getGPS() : string
Arduino
+Arduino() : void
+SendCommand(Command : String) : void
<<use>>
<<use>>
<<use>>
<<use>>
MediaRecorder
LocationManager
USBManager

Klassen- und Methodenbeschreibung

Controller

Die Klasse „Controller“ dient der allgemeinen Steuerung des Programmablaufs und verwendet die Methoden der Klassen „RecordSound“, „GPS“, „Arduino“, „Network“, „Camera“ und „Connection“ zur Realisierung der Programmfunktionen. Hierzu sind alle Konstruktoren als Attribute in der Klasse „Controller“ enthalten.

Konstruktor *Controller(): void* initialisiert sämtliche Konstruktoren und bereitet das System zum Senden und Empfangen von Daten und Kommandos vor.

SendData(): void sendet ein Datenpaket mit Info- und Statusmeldungen an den Computer.

ReceiveControlSignal(ControlSignal: string): void empfängt einen Steuerstring vom Computer und leitet diesen an die entsprechenden Klassen weiter.

CameraPicture(): void nimmt ein Bild von der ausgewählten Kamera auf und bereitet dies zum Senden an den Computer vor.

RecordSound

Diese Klasse ist für das Aufnehmen von Sounds zuständig, die man später auf den Computer übertragen kann.

Sound() ist der Konstruktor und erzeugt eine Instanz von MediaRecorder.

startRecordSound(path: Path) startet die Aufnahme. Dabei wird der Dateipfad für diese Aufnahme angegeben.

stopRecordSound() stoppt die Aufnahme.

Network

Diese Klasse ist für die Verbindung zum Computer zuständig. Sie empfängt und sendet die benötigten Daten um das Fahrzeug zu steuern.

Netzwerk(controller: Controller) ist der Konstrukter dieser Klasse. Dabei werden Sockets erstellt um die Verbindung aufzubauen.

receiveControllsignal() ist für den Empfang der Steuerdaten zuständigt und leitet die zum Controller weiter.

sendInfoPackage(info: string) ist für das Senden der Informationsdaten zuständig. Es gibt dem Controller eine Rückmeldung über den Erfolg des Sendens zurück.

Camera

Diese Klasse ist für das Aufnehmen und das Übertragen der Kamerabilder zuständig.

Camera() ist der Konstruktor und erstellt ein Socket für das Senden der Kamerabilder. Des Weiteren wird die Kamera initialisiert.

getPicture() macht ein Kamerabild und schickt dieses an den Computer.

setRes(widht:int, heigh:int) setzt die Auflösung der Kamerabilder auf die zu übergebenden Werte.

GPS

Die Klasse „GPS“ ist für die Bereitstellung der Standortdaten des Smartphones zuständig. Dazu werden die Methoden der Android-Klasse „LocationManager“ verwendet:

Konstruktor *GPS(): void* registriert sich bei einer Instanz des LocationManagers.

GetGPS(): void ruft den aktuellen Standort ab und speichert ihn in den Attributen „Latitude“ und „Longitude“ ab.

Arduino

Die Klasse „Arduino“ stellt die Schnittstelle zum Arduino-Controller dar. Zur Kommunikation werden die Methoden des „ADK“ (Accessory Development Kit) verwendet.

Konstruktor *Arduino(): void* stellt eine USB-Verbindung mit dem Arduino-Board her und bereitet die Verbindung für das Senden von Steuerkommandos vor.

SendCommand(Command: string): void dekodiert den übergebenen String „Command“ und führt die in ihm verschlüsselten Befehle zur Steuerung der Motoren aus.

Connection

Durch die Klasse „Connection“wird der Status der aktuellen Verbindungsart überwacht. Hierzu werden die Funktionen der Androidklasse „ConnectivityManager“ benutzt:

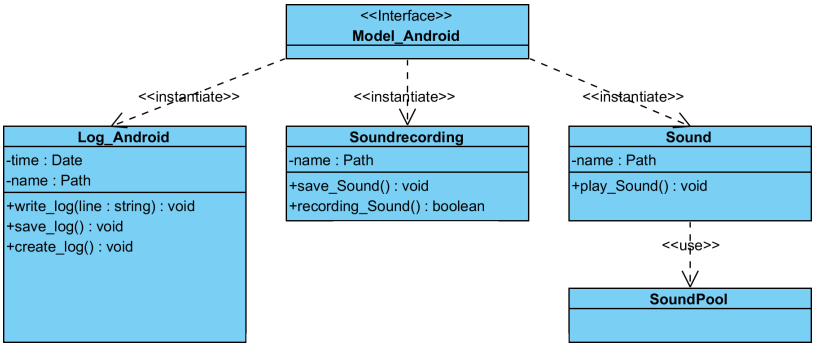
Konstruktor *Connection(): void* registriert sich bei dem „ConnectionManager“ des Smartphones zum Abruf des Verbindungsstatuses.

GetMobile(): void ruft den Status des mobilen Internets ab und speichert diesen in der Variable „MobileConn“.

GetWLAN(): void ruft den Status des WLAN ab und speichert diesen in der Variable WLANConn.

3.2.5 Model (Android-Applikation)

Das Model_Android ist für die Datenverarbeitung der Log-Datei auf dem Android-Smartphone zuständig. Das Model des Android- Smartphones besteht aus dem Interface Model_Android. Dieses besitzt die Funktionsmodule „Log_Android“, „Soundrecording“ und „Sound“.



Klassen- und Methodenbeschreibung

Model_Android

In dieser Klasse werden alle Funktionen des Models vereinigt und als Konstruktoren dargestellt.

Log_Android

In diesem Modul wird eine Log-Datei auf dem Android Smartphone gespeichert. Als Attribute sind der *name* und die dazugehörige Zeit (*time*) vorhanden. Die Methoden sind:

write_log(line: string): void führt dazu, das neue Daten als Zeichenkette in das Log geschrieben werden. Somit wird dieser ständig aktualisiert. Es gibt keinen Rückgabewert.

save_log(): void führt dazu, dass die Log-Datei gespeichert wird, sodass sie später wieder aufgerufen werden kann. Es gibt keinen Rückgabewert.

create_log(): void erstellt eine neue Log-Datei. Als Name enthält sie das Datum und die Zeit, an der das Log erstellt wurde. Es gibt keinen Rückgabewert.

Sound

Das Soundmodul stellt für die Anfrage eines Signaltones eine Audiodatei zur Verfügung. Als Attribut wird ausschließlich *name* verwendet. Folgende Methode ist vorhanden:

playSound(): void lässt eine Audiodatei aus dem Soundpool laden, damit diese dann abgespielt werden kann. Es gibt keinen Rückgabewert.

Soundrecording

Die Methode *Soundrecording* benutzt das Mikrofon vom Mobiltelefon um den Ton aufzunehmen. Der Mitschnitt wird dann auf dem Smartphone gespeichert und kann später angehört werden. Attribut ist *name*. Vorhandene Methoden sind:

save_sound(): void speichert den vom Mikrofon des Smartphones empfangenen Ton als Datei ab. Es gibt keinen Rückgabewert.

Mit *recording_sound(): boolean* wird die Tonaufnahme mit Hilfe des Mikrofons vom Smartphone gestartet. Der Rückgabewert wird auf „true“ gesetzt, wenn eine Aufnahme läuft. Bei keinem aktiven Soundmitschnitt wird „false“ zurückgegeben.

3.2.6 View (Android-Applikation)

Unter Android leiten alle Bedienelemente direkt oder indirekt von der Klasse *android.view.View* ab. Jede View belegt einen rechteckigen Bereich des Bildschirms, wobei ihre Größe und Position durch Layouts bestimmt werden. Diese wiederum erben von *android.view.ViewGroup*,ebenfalls ein Kind von *View*. Sie haben keine eigene grafische Repräsentation (Abstrakte Klasse), sondern sind Container für weitere Views und ViewGroups.

Als Manifest besteht die minimalistische Gestaltung der Oberfläche, die aus einer Anzeige der IP Adresse über ein Textfeld sowie unserem Logo besteht. Die Benutzung des „Home-Button“ bewirkt die Minimierung unserer Android-Applikation. Mit einem Icon im Benachrichtungsbereich soll die weiterhin bestehende Anwesenheit der Steuerung signalisiert werden.

Eine Android-Anwendung beschreibt ihre Benutzeroberfläche üblicherweise mittels XML-basierten Layoutdateien. Diese Art der Festlegung soll bei unserer Android-Applikation ebenso genutzt werden. Oberflächenbeschreibungen werden in *layout*, einem Unterverzeichnis von *res* gespeichert. Beim Anlegen des Projektes hat Eclipse dort die Datei *main.xml* bereits abgelegt. Die XML-Datei bildet die Hierarchie der Benutzeroberfläche ab. Demzufolge wird <LinearLayout> das Wurzelelement sein.

4 Glossar

Begriff	Erklärung
Android	ist ein Betriebssystem für Smartphones.
Applikation	ist ein Computerprogramm, das eine für den Anwender nützliche Funktion ausführt.
Arduino	ist eine aus Hard- und Software bestehende Mikrocontroller-Umgebung.
Arduino-Controller	ist ein Microcontroller, der benötigt wird um die Motoren des Fahrzeuges anzusteuern.
Bildqualitätsregulierung	bedeutet Anpassung der Bildrate und/oder der Bildauflösung.
Bool'sche Variable	bezeichnet eine Schaltvariable die nur „wahr“ oder „falsch“ annehmen kann.
Carduinodroid	ist die Zusammensetzung der Wörter Car, Arduino und Android zu einem Wort.
CSV	oder „Comma-Separated Values“ beschreibt den Aufbau einer Textdatei zur Speicherung oder zum Austausch einfach strukturierter Daten.
Datenbanken	dienen zur effizienten elektronischen Datenverwaltung.
Drohne	bezeichnet ein unbemanntes Fahrzeug.
GPS	ist ein globales Navigationssatellitensystem zur Positionsbestimmung und Zeitmessung.
GUI	oder „Graphical User Interface“ bezeichnet die grafische Benutzeroberfläche des Systems.

Icon	lässt sich als Symbol oder Sinnbild übersetzen.
Implementierung	bedeutet die Umsetzung von festgelegten Strukturen in einem System.
Integer	ist ein Datentyp in der Informatik der ganzzahlige Werte annehmen kann.
Interaktion	bezeichnet hier das wechselseitige Aufeinandereinwirken von Systemen und Strukturen.
Java	ist eine objektorientierte Programmiersprache.
Konstruktor	werden in der Programmierung spezielle Prozeduren bzw. Methoden bezeichnet, die Objekte und Variable aufrufen.
Log	ist die Zusammenfassung und Aufzeichnung bestimmter Ereignisse.
Live-log	ist das systematische und zeitnahe Aktualisieren des Logs.
Logo	stellt ein Produkt oder Firma in Form einer Grafik dar.
Modul	bezeichnet eine abgeschlossene funktionale Einheit.
Persistenz	steht für langfristiges Fortbestehen des Systems.
Prototyp	ist ein Vorab-Exemplar, welches später in Serie gefertigt werden soll bzw. kann.
Socket	dient zur Verbindung eines Computerprogramms mit einem Netzwerk.
Source-Code	bezeichnet den Quelltext eines Programmes.
SP	wird in diesen Dokument als Abkürzung für Smartphone benutzt.

String	ist ein Datentyp in der Informatik, der eine Zeichenkette als Wert annimmt.
Tastenfeedback	zeigt zur Bestätigung die Tasteneingabe seitens des Users an.
Tool	bezeichnet ein PC-Werkzeug in Form einer Anwendungssoftware.
USB-Port	ist die Schnittstelle für eine USB-Verbindung.
View	bezeichnet die Ansicht auf ein bestimmtes Objekt.