ulm university universität
uulm

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Softwaretechnik und Programmiersprachen

Softwaregrundprojekt Meilenstein 6

Softwaregrundprojekt an der Universität Ulm

Vorgelegt von:
Gruppe 10

Dozent:
Florian Ege

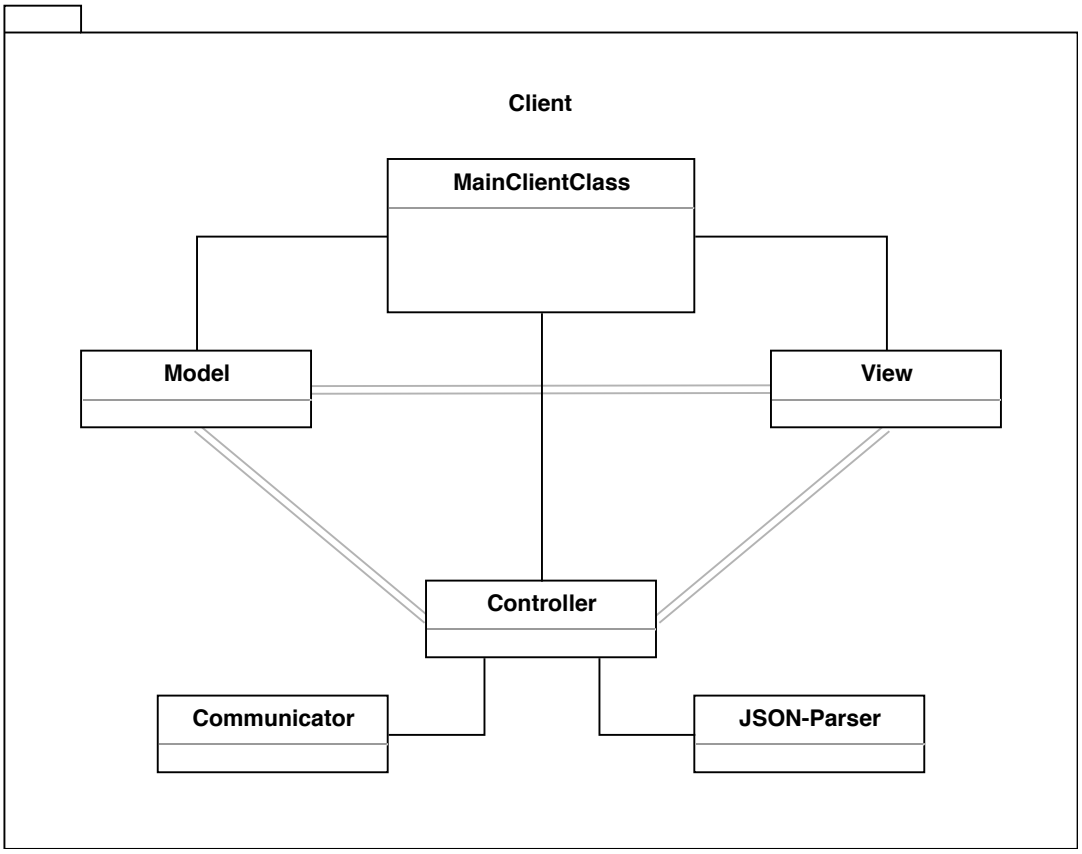
Betreuer:
Stefanos Mytilineos

2019

Inhaltsverzeichnis

1 Klassen Übersicht

1.1 Klassen-Diagramm

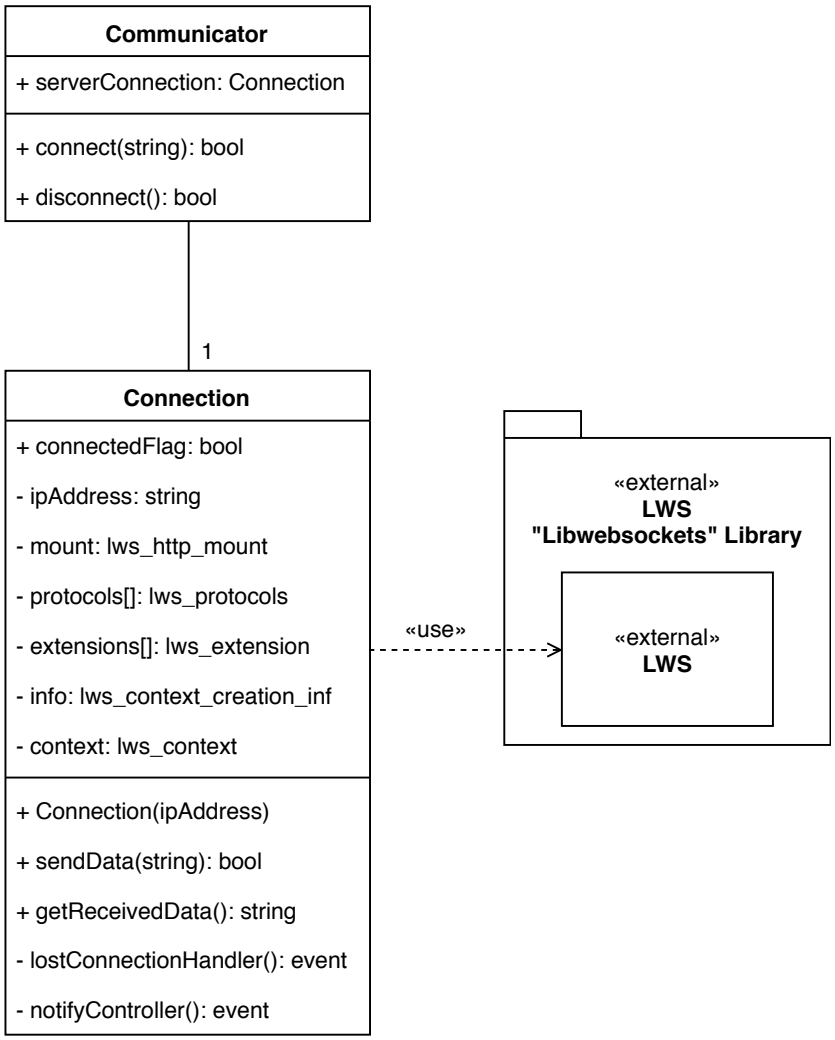


1.2 Beschreibung

Die Grundlegende Struktur der Client Anwendung basiert auf dem Model-View-Controller Modell (MVC Modell). Dabei werden alle Aufgaben weitestgehend auf drei unabhängige Klasse verteilt. In der Model Klasse sind alle für die Funktion der Anwendung benötigten Datenstrukturen enthalten. Die View Klasse ist dafür verantwortlich alle Daten in einer geeigneten Form darzustellen. Die Controller Klasse dient dazu, basierend auf verschiedenen Ereignissen, wie z.B. Benutzereingaben, die Daten im Model anzupassen, sodass in diesem Fall z.B. eine neue Spielsituation von der View abgebildet werden kann. Die Klassen Communicator und JSON-Parser stelle Hilfsklassen der Controller Klasse dar.

2 Kommunikator

2.1 Klassen-Diagramm



2.2 Beschreibung

2.2.1 Connector (Klasse)

Im Allgemeinen dient die Connector Klasse dazu, die Kommunikation zwischen der Client Anwendung und der Server Anwendung über das Netzwerk zu verwalten.

connect (Methode) Die connect Methode dient dazu, eine neue Verbindung mit einem Server aufzubauen. Die IP-Adresse des Servers wird dabei als Parameter übergeben. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false.

disconnect (Methode) Die disconnect Methode dient dazu, eine bestehende Verbindung mit einem Server zu trennen. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false.

2.2.2 Connection (Klasse)

sendData (Methode) Die öffentliche sendData Methode dient dazu, einen im JSON-Format formatierten String an den Server zu übertragen. Ist die Aktion erfolgreich, so wird true zurück gegeben, anderenfalls false. Zur Umsetzung dieser Methode werden Funktionen und Strukturen der Drittanbieter-Bibliothek LWS benötigt.

getReceivedData (Methode) Mit Hilfe der öffentlichen Methode getReceivedData kann der letzte vom Server gesendete JSON Datensatz ausgelesen werden. Zur Umsetzung dieser Methode werden ebenfalls Funktionen und Strukturen der Drittanbieter-Bibliothek LWS benötigt.

lostConnectionHandler (Methode) Die private Methode lostConnectionHandler kümmert sich im Falle eines außerplanmäßigen Verbindungsverlust darum, dass alle notwendigen Schritte eingeleitet werden, indem der Controller über den Verbindungsabbruch informiert wird. Auch hier werden zur Umsetzung Funktionen und Strukturen der Drittanbieter- Bibliothek LWS benötigt.

notifyController (Methode) Die private notifyController Methode ist dafür zuständig den Controller über das Eingehen von neuen Nachrichten des Servers zu informieren.

2.2.3 LWS (Externe Bibliothek)

Bei dieser Klasse Handelt es sich um eine externe Bibliothek, die verschiedenste Methoden und Strukturen zur Verfügung stellt um in einem C/C++ Projekt einfache Server Client Verbindungen mittels Web-Sockets zu implementieren. Genauere Informationen zu dieser Bibliothek sind unter folgendem Link zu finden:
`https://libwebsockets.org/`

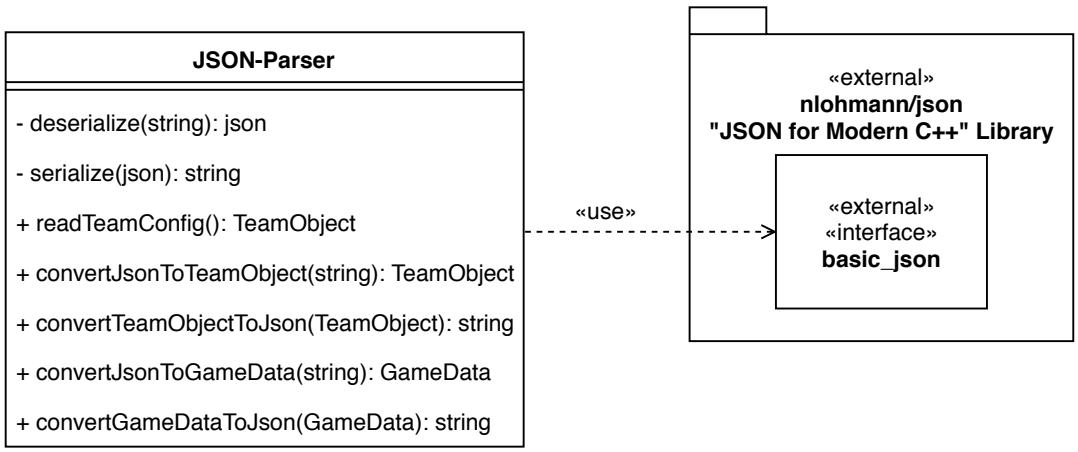
2.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Funktionale Anforderungen	Methoden
FA55	Connector::connect
FA55	Connector::disconnect
FA55	Connection::sendData
FA55	Connection::getReceivedData
FA55	Connection::lostConnectionHandler
FA55	Connection::notifyController

3 JSON-Parser

3.1 Klassen-Diagramm



3.2 Beschreibung

3.2.1 JSON-Parser (Klasse)

Der JSON-Parser übernimmt das Serialisieren und Deserialisieren von Objekten in einen String im JSON-Format. Diese Funktionalität wird benötigt, da alle Daten, die über das Netzwerk versendet werden oder aus einer Datei eingelesen werden diesem Format entsprechen müssen.

deserialize (Methode) Die private deserialize Methode dient dazu, einen dem JSON-Format konformen String in ein JSON-Object umzuwandeln. Dabei wird bei der Umsetzung dieser Funktionalität auf Methoden und Strukturen aus einer Softwarebibliothek eines Drittanbieters zurückgegriffen.

serialize (Methode) Die private serialize Methode dient dazu, aus einem JSON-Object ein dem JSON-Format genügenden String zu erzeugen. Wie auch schon bei der deserialize Methode wird dabei auf Methoden und Strukturen aus einer Softwarebibliothek eines Drittanbieters zurückgegriffen.

readTeamConfig (Methode) Mit Hilfe der readTeamConfig Methode kann die Team-Konfiguration aus der Team-Konfigurationsdatei eingelesen werden und in ein internes TeamObject umgewandelt werden. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertJsonToTeamObject (Methode) Die convertJsonToTeamObject Methode dient dazu, die vom Server empfangenen Daten über das gegnerische Team in ein internes TeamObject umzuwandeln. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertTeamObjectToJson (Methode) Die convertGameDataToJson Methode bildet das Gegenstück zur convertJsonToTeamObject Methode. Es kann also ein internes GameTeam in einen JSON konformen String konvertiert werden, welcher dann an den Server versendet werden kann. Im Hintergrund greift diese Methode auf die serialize Methode zurück.

convertJsonToGameData (Methode) Die convertJsonToGameData Methode dient dazu, die vom Server empfangenen Daten über das aktuelle Spielgeschehen aus dem JSON-Format in ein Internes GameObject zu konvertieren. Diese wird dann im Model hinterlegt. Im Hintergrund greift diese Methode auf die deserialize Methode zurück.

convertGameDataToJson (Methode) Die convertGameDataToJson Methode bildet das Gegenstück zur convertJsonToGameData Methode. Es kann also ein internes GameObject in einen JSON konformen String konvertiert werden, welcher dann an den Server versendet werden kann. Im Hintergrund greift diese Methode auf die serialize Methode zurück.

3.2.2 nlohmann::json (Externe Bibliothek)

Bei dieser Klasse Handelt es sich um eine externe Bibliothek, die verschiedenste Methoden und Strukturen zur Verfügung stellt um in einem C++ Projekt mit Objekten mit JSON-Struktur einfach zu arbeiten. Vermutlich wird die von *Niels Lohmann* entwickelte

freie JSON Bibliothek namens *JSON for Modern C++* verwendet werden. Genauere Informationen zu dieser Bibliothek sind unter folgendem Link zu finden:
`https://github.com/nlohmann/json`

3.3 Zuordnung der Funktionalen Anforderungen

Die Funktionalen Anforderungen werden den Methoden folgendermaßen zugeteilt:

Funktionale Anforderungen	**Methoden**
FA54	readTeamConfig
FA54	convertJsonToTeamObject
FA54	convertTeamObjectToJson
FA53	convertGameDataToJson
FA53	convertJsonToGameData