ulm university universität
uulm

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

Softwaregrundprojekt Meilenstein 5

Softwaregrundprojekt an der Universität Ulm

Vorgelegt von:
Gruppe 10

Dozent:
Florian Ege

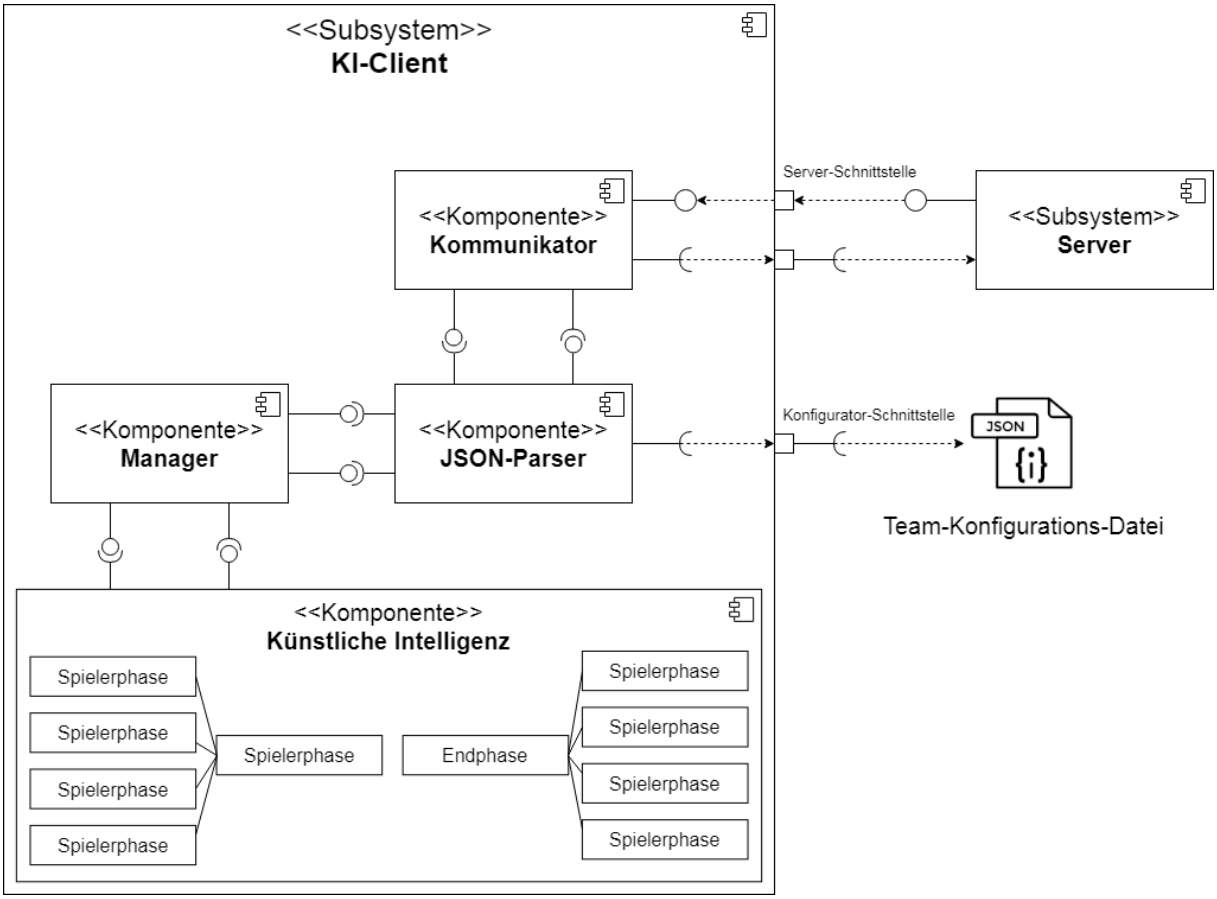
Betreuer:
Stefanos Mytilineos

2019

Inhaltsverzeichnis

1 KI-Client

1.1 UML2-Komponentendiagramm



1.2 Beschreibungen

KI-Client: Das Subsystem KI-Client ist eine Kommandozeilenanwendung, die sich wie ein Nutzer-Client mit einem Server verbindet, einer Partie beitritt und, gesteuert von einer KI, einen menschlichen Spieler simuliert.

Manager: Diese Komponente verwaltet die Daten des KI-Client, insbesondere die aktuelle Spielsituation und verarbeitet die Anwendungsparameter.

Der Manager empfängt vom JSON-Parser die aktuelle Spielsituation und aktualisiert seine gespeicherten Daten entsprechend. Er übergibt seine Daten an die KI, damit diese Entscheidungen über die durchzuführenden Aktionen treffen kann. Anschließend empfängt er die Entscheidungen der KI und aktualisiert die gespeicherte Spielsituation entsprechend, bevor er sie dem JSON-Parser übergibt.
Zu Beginn einer Partie erhält der Manager vom JSON-Parser die Daten aus der zu verwendende Team-Konfigurationsdatei.
Der Manager ist die zentrale Komponente des Subsystems und behandelt den Programmverlauf und die Parameter, damit die KI sich ausschließlich mit dem Entscheiden befassen kann.

JSON-Parser: Diese Komponente fungiert als Dolmetscher für die Kommunikation zwischen KI-Client, Server und Team-Konfigurator.
Der JSON-Parser empfängt die vom Kommunikator kommenden Nachrichten im JSON-Format und extrahiert daraus Daten über die aktuelle Spielsituation, die er anschließend dem Manager übergibt. Andersherum empfängt er Daten vom Manager, verpackt sie in einer Nachricht im JSON-Format und sendet sie an den Kommunikator.
Außerdem liest der JSON-Parser Team-Konfigurationsdateien, extrahiert die Daten und gibt sie an den Manager weiter.
Das Übersetzen der JSON-Dateien wird in eine eigene Komponente ausgelagert, damit der Manager unabhängig von den Konventionen der JSON-Nachrichten ist und das Subsystem leicht an diese angepasst werden kann.

Künstliche Intelligenz Diese Komponente trifft Entscheidungen über durchzuführende Aktionen anhand der aktuellen Spielsituation.
Die künstliche Intelligenz, kurz KI, empfängt Daten vom Manager und verarbeitet sie in ihrer jeweiligen Logik, um die nächste durchzuführende Aktion zu ermitteln. Sobald sie eine Entscheidung getroffen hat, teilt sie diese dem Manager zur weiteren Verarbeitung mit.
Die interne Struktur der künstlichen Intelligenz enthält eine Logik für jede Spielfiguren-Rolle und jede mögliche Einmischung, da jede Spielfigur anhand ihrer Aufgabe und Spezialisierung handeln muss.
Die KI soll dabei unabhängig von den anderen Komponenten sein, damit sie problemlos zu jedem Zeitpunkt optimiert und für unterschiedliche Schwierigkeitsstufen ausgetauscht werden kann.

Kommunikator Diese Komponente ist dafür zuständig, mit dem Server zu kommunizieren.
Der Kommunikator stellt eine Verbindung mit dem angegebenen Server her, hält sie

aufrecht und versucht, sie bei Verbindungsabbruch wiederherzustellen. Er empfängt Nachrichten vom Server und gibt sie unverarbeitet an den JSON-Parser weiter und überträgt umgekehrt die vom JSON-Parser kommenden Daten an den Server.
Diese Funktionen werden von einer separaten Komponente übernommen, da sie in anderen Subsystemen ebenfalls gebraucht werden und der Kommunikator dort wiederverwendet werden kann.

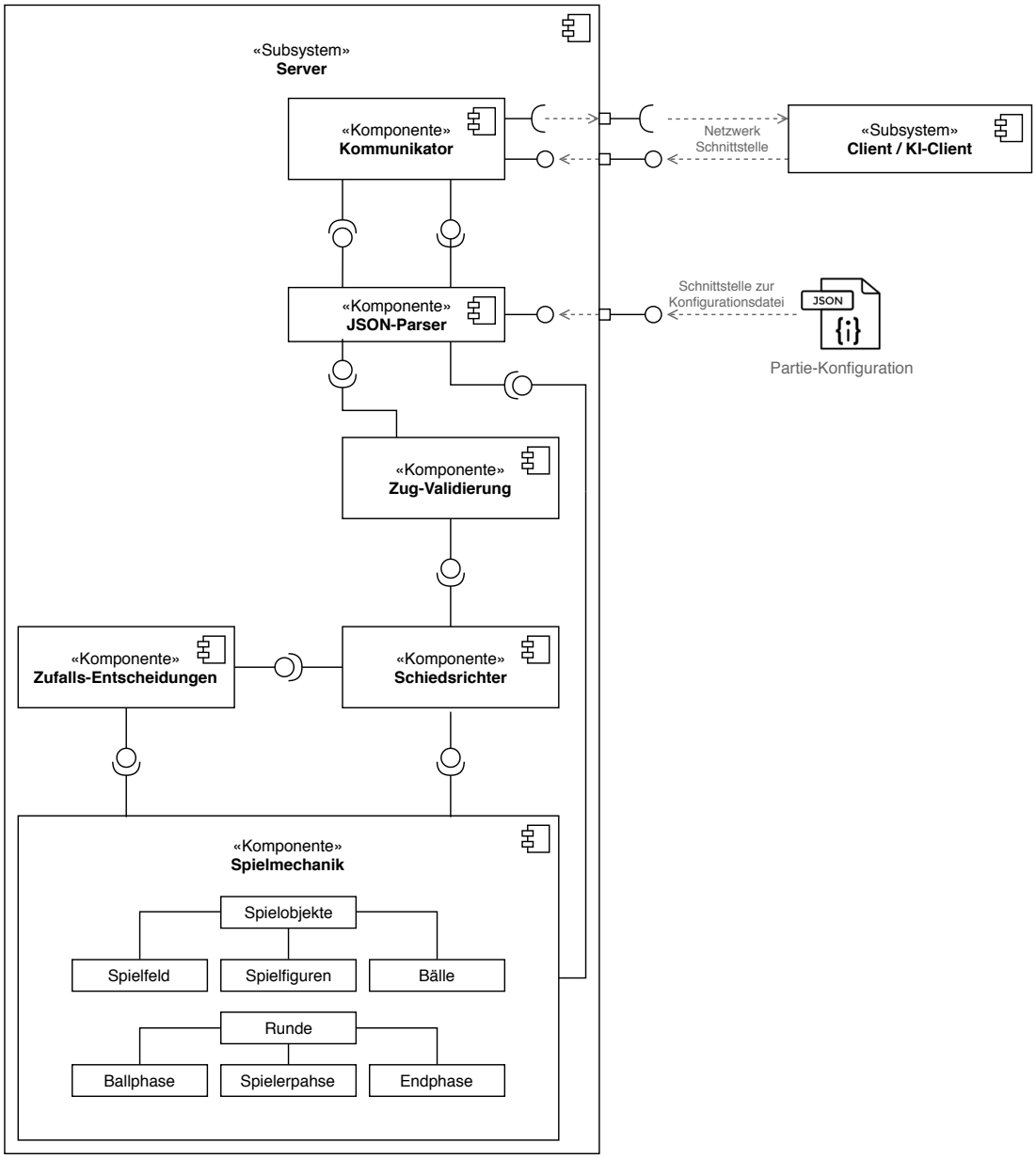
1.3 Zuordnung der Funktionalen Anforderungen

Die funktionalen Anforderungen gemäß dem Pflichtenheft werden den Komponenten folgendermaßen zugeteilt:

Komponente	**Abgedeckte funktionale Anforderungen**
Manager	FA1 - FA8 FA10 - FA20 FA44 - FA47 FA73 - FA75
JSON-Parser	FA54
Künstliche Intelligenz	FA17 - FA 21 FA26 - FA28 FA30 - FA35 FA37 - FA42 FA46 - FA47 FA50
Kommunikator	FA55

2 Server

2.1 UML2-Komponentendiagramm



2.2 Beschreibungen

Server Beim Subsystem Server handelt es sich um die Konsolenanwendung, die als Schnittstelle für zwei oder mehr Clients dient. Die Serveranwendung kümmert sich dabei primär um die Spielsteuerung und agiert dabei unabhängig von den Client-Anwendungen.

Spielmechanik Die Spielmechanik ist die zentrale Komponente des Server Subsystems. Diese bildet dabei die komplette Partie intern ab. In der Spielmechanik sind also zu jeder Zeit die aktuellsten Status der Spielobjekte hinterlegt. Während eines Spiels senden die Clients ihre gewünschten Züge an den Server. Die Spielmechanik wertet die Züge, nachdem sie validiert und vom Schiedsrichter geprüft wurden, aus und aktualisiert dann gegebenenfalls die aktuelle Spielsituation. Diese wird im Anschluss wieder vom Server aus zu den einzelnen Clients ausgegeben. Als zentrale Komponente des Server Subsystems ist es zwingend notwendig, dass diese Komponente als Einheit gesehen wird, da ohne diesen Teil kein das Spielen nicht möglich wäre.

Zug-Validierung Die Zug-Validierung prüft, ob die Züge, die von einem Spieler über seinen Client übermittelt werden, grundsätzlich möglich sind. Dabei wird jedoch nicht geprüft ob der gewünschte Zug ein Foul darstellt. Notwendig ist diese Prüfung, da nicht sichergestellt ist, dass jede Client-Anwendung tatsächlich prüft ob die Züge die ein Spieler tätigen will auch grundsätzlich möglich sind. Die Komponente ist aus der eigentlichen Spielmechanik ausgegliedert, da die Zug-Validierung auch in anderen Subsystem eingesetzt werden könnte, z.b. einer Client-Anwendung.

Schiedsrichter Die Schiedsrichter Komponente des Server Subsystems. Möchte ein Spieler einen verbotenen Zug tätigen, wird in der Schiedsrichter Komponente die Entscheidung getroffen, ob der Spieler bestraft wird. Diese Komponente ist aus der Spielmechanik ausgegliedert, da da das Spiel grundsätzlich auch ohne diese Komponente möglich ist und daher diese Komponente in einem nachgelagerten Entwicklungsschritt implementiert werden kann.

Zufalls-Entscheidungen Dies Komponente stellt der Spielmechanik und der Schiedsrichter Komponente eine Schnittstelle zur Verfügung, die es erlaubt die vielen Zufalls-Ereignisse in einer Partie auszuwerten.

Kommunikator Der Kommunikator bildet die Komponente, die sich um die Datenübertragung zwischen der Server- und den Client-Anwendungen kümmert. Dabei werden dort zum einen die Status der Verbindungen zu den Clients überwacht und verwaltet. Zum anderen wir die Datenübertragung in beide Richtungen bereitgestellt. Da auch in den Client-Anwendungen eine ähnliche Komponente von Nöten ist, bietet es sich an, diese Funktionalitäten in einer eigenen Komponente auszulagern.

JSON-Parser Der JSON-Parser (de-) serialisiert die Objekte der Spiellogik. Da auch in den Client-Anwendungen eine ähnliche Komponente von Nöten ist, bietet es sich

an, diese Funktionalitäten in einer eigenen Komponente auszulagern.

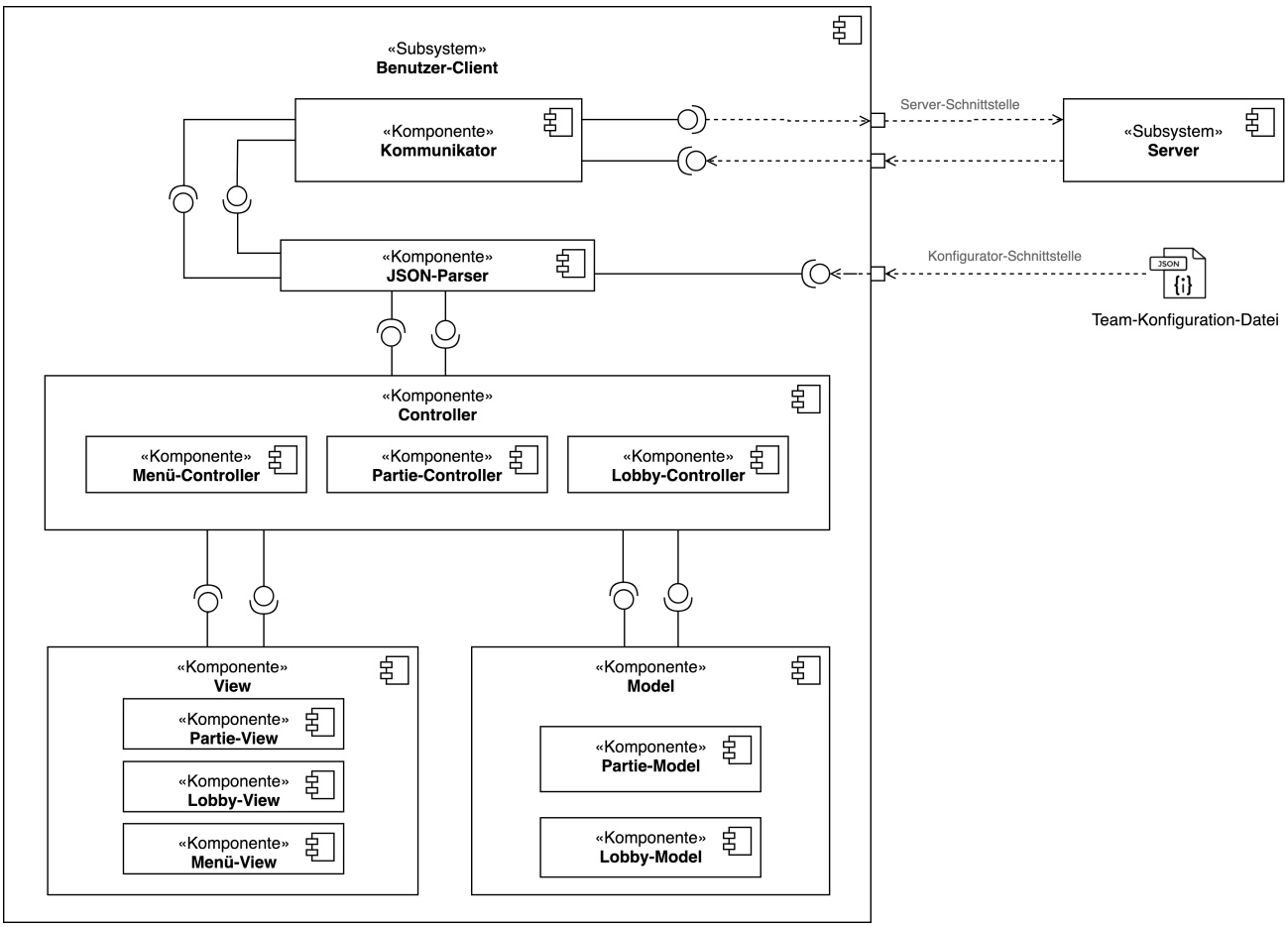
2.3 Zuordnung der Funktionalen Anforderungen

Die funktionalen Anforderungen gemäß dem Pflichtenheft werden den Komponenten folgendermaßen zugeteilt:

Komponente	**Abgedeckte funktionale Anforderungen**
Spielmechanik	FA1 - FA36 FA43 - FA52 FA56 FA59
Schiedsrichter	FA36 - FA42
Zug-Validierung	FA67
Zufalls-Entscheidungen	FA58
Kommunikator	FA55
JSON-Konverter	FA53 FA57

3 Benutzer-Client

3.1 UML2-Komponentendiagramm



3.2 Beschreibungen

Benutzer-Client Beim Benutzer-Client handelt es sich um eine Anwendung mit grafischer Benutzeroberfläche mit der *Fantastic Feasts*-Partien beigetreten werden kann und diese auf verschiedene Weisen verfolgt werden können. In einer Lobby können Partien auf einem Server erstellt werden oder es kann einer bestehenden Partie auf einem Server beigetreten werden. Die Spiellogik einer Partie läuft jedoch auf dem Server. Eine Partie lässt sich sowohl passiv als Beobachter verfolgen als auch aktiv

als Spieler gestalten. In diesem Fall nimmt der Benutzer-Client Spieleraktionen entgegen und sendet sie an den Server. Es wird eine Model-View-Controller-Architektur gewählt, da sowohl vom Server als auch vom Benutzer Eingaben erfolgen können. Das Behandeln dieser Eingaben wird zentral vom Controller übernommen, sodass beispielsweise das Umsetzen der Nutzereingaben nicht an die View gekoppelt ist. Das würde dazu führen, dass sich die View, also die reine Darstellung, nur schwer unabhängig von der Logik verändern ließe. Ein Model ist notwendig, um die Spielzustände auf dem Benutzer-Client eindeutig zu verwalten. Somit können beispielsweise vorab Einschränkungen auf erlaubte Spielzüge vorgenommen werden, ein Zug auf Vollständigkeit geprüft und die Darstellungen der View am Model orientiert werden. Das Model wird allerdings nicht die gesamte Spiellogik enthalten, da dies Aufgabe des Servers ist.

Controller Der Controller ist die Schaltstelle aller Aktionen im Benutzer-Client. Er besteht aus den drei Komponenten Menü-Controller, Spiel-Controller und Lobby-Controller.

Menü-Controller Der Menü-Controller beinhaltet die Logik, mit der im Menü zwischen verschiedenen Ansichten hin und her gewechselt werden kann. Über ihn kann das Spiel beendet werden, die Lobby betreten, zur Hilf-Ansicht gewechselt werden oder etwaige Einstellungen am Benutzer-Client vorgenommen werden. Er updatet die Menu-View und behandelt Nutzereingaben entsprechend.

Spiel-Controller Der Spiel-Controller hat zwei Aufgaben. Einerseits behandelt er alle Ereignisse und Nutzereingaben, die über die Spiel-View bereitgestellt werden und leitet sie an das Model bzw. den Server weiter. Andererseits nimmt er Spieldaten vom Server bzw. dem JSON-Parser entgegen und aktualisiert damit die Partie-View und das Partie-Model des Benutzer-Clients. Er sorgt dafür, dass die View nur Nutzereingaben zulässt, die erlaubt sind. Diese Information erhält er vom Model.

Lobby-Controller Mit dem Lobby-Controller werden die Interaktionen in der Lobby zwischen Server, Benutzer-Client-Anwendung und Nutzereingaben behandelt. Insbesondere kann über den Lobyy-Controller mithilfe des JSON-Parsers eine Schnittstelle zum Dateisystem hergestellt werden, wodurch eine Team-Konfiguration ausgewählt werden kann.

View Die View enthält alle Klassen, die die grafische Benutzeroberfläche des Benutzer-Clients bilden. Sie unterteilt sich in die drei Komponenten Menü-View, Lobby-View und Partie-View.

Partie-View Die Partie-View enthält alle Klassen, die die grafische Darstellung einer Partie sowohl im Beobachter- als auch im Spieler-Modus ausmachen. Nutzereingaben und Ereignisse werden an den entsprechenden Controller weitergeleitet.

Lobby-View Die Lobby-View enthält alle Klassen, die zur grafischen Darstellung der Partie-Lobby notwendig sind. Nutzereingaben werden an den entsprechenden Controller weitergeleitet.

Menü-View Die Menü-View enthält alle Klassen, die zur grafischen Darstellung des Menüs notwendig sind. Nutzereingaben werden an den entsprechenden Controller weitergeleitet.

Model Das Model ist eine Gruppe von Klassen, die wichtige Zustände des Benutzer-Clients repräsentieren und Zustandsübergänge festlegen.

Lobby-Model Das Lobby-Model fasst die Klassen zusammen, die die Zustände der Lobby repräsentieren. Dazu gehören zum Beispiel vorhandene Partien und beigetretene Spieler bzw. Beobachter. Auch bestehende Verbindungen mit dem Server werden hier repräsentiert.

Partie-Model Das Partie-Modell ist eine stark vereinfachte Variante der Spiellogik auf dem Server. Sie beschränkt sich darauf, einen Partie-Zustand eindeutig zu beschreiben und die Regeln für erlaubte Spielzüge zu kennen. So wird ermöglicht, dass der Spieler nur erlaubte Züge durchführen kann und diese Information nicht jedes mal vom Server angefordert werden muss. Zusätzlich hilft das Model dabei, eine konsistente Darstellung der Partie zu erhalten, da die View sich stets an den Zuständen des Models orientiert.

Kommunikator Der Kommunikator de bildet die Komponente, die sich um die Datenübertragung mit dem Server kümmert. Daten über den Spielzustand werden vom Server empfangen und Spieleraktionen werden an den Server gesendet.

JSON-Parser Der JSON-Parser wandelt die JSON-Daten in interne Objekte der Software um und umgekehrt. Da auch in den Client-Anwendungen eine ähnliche Komponente von Nöten ist, bietet es sich an diese Funktionalitäten in einer eigenen Komponente auszulagern.

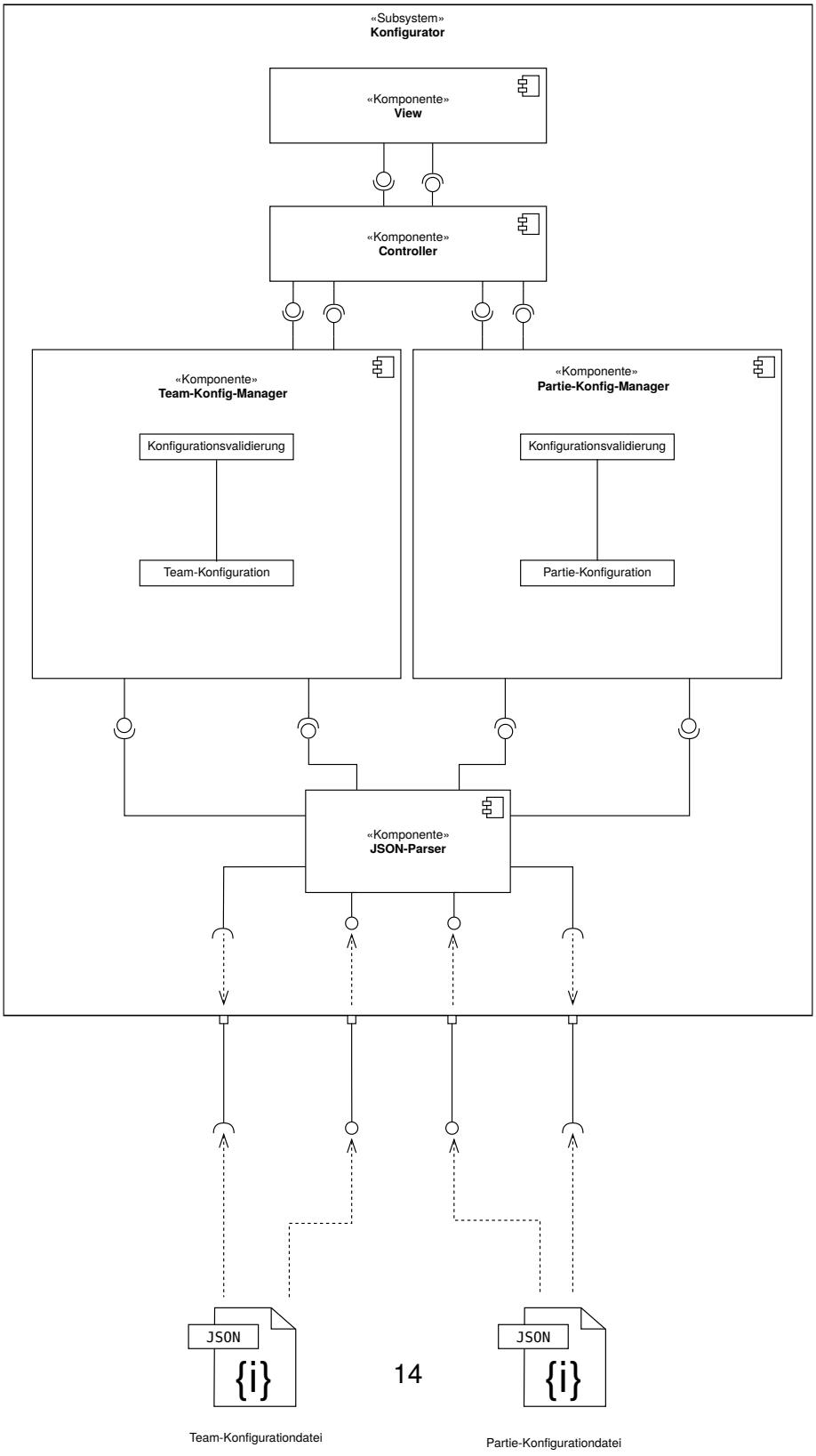
3.3 Zuordnung der Funktionalen Anforderungen

Die funktionalen Anforderungen gemäß dem Pflichtenheft werden den Komponenten folgendermaßen zugeteilt:

Komponente	Abgedeckte funktionale Anforderungen
Menü-Controller	FA56 FA60 FA65 FA67
Lobby-Controller	FA61 FA56 FA63 FA67
Partie-Controller	FA56 FA67-69
Menü-View	FA56 FA60 FA65
Lobby-View	FA61 FA63
Partie-View	FA62 FA64 FA66
Partie-Model	FA01-52 FA60 FA65
Lobby-Model	FA61 FA63 FA65
Web Socket	FA55
JSON-Konverter	FA53 FA57

4 Konfigurator

4.1 UML2-Komponentendiagramm



4.2 Beschreibungen

Konfigurator Beim Subsystem Konfigurator handelt es sich um eine graphische Benutzeroberfläche, die der Erstellung von Partie- oder Team-Konfigurationen dient. Der Konfigurator ist ein eigenes Programm, und somit unabhängig von den anderen Anwendungen.

Team-Konfig-Manager Der Team-Konfig-Manager verwaltet eine Team-Konfiguration. Er kommuniziert über den Controller mit der View und nimmt ebenfalls neue Daten von dieser entgegen. Bevor die Daten, die der Nutzer in der graphischen Oberfläche einstellt, in einem Team-Konfigurations-Objekt gespeichert werden, werden sie zuvor noch von der Konfigurationsvalidierung überprüft, sodass sichergestellt ist, dass nur gültige Konfigurationen erstellt werden können. Der Team-Konfig-Manager kümmert sich nur um die programminterne Repräsentation der Konfiguration und ist deshalb streng von der Anzeige sowie dem JSON-Parser getrennt.

Partie-Konfig-Manager Der Team-Konfig-Manager verhält sich analog zum Team-Konfig-Manager. Er unterscheidet sich lediglich in der Konfigurationsvalidierung, die natürlich auf eine ebenfalls unterschiedliche Repräsentation der Konfiguration zugeschnitten ist.

Controller Der Controller kümmert sich streng nach dem MVC-Pattern um die Kommunikation zwischen den Konfig-Managern un der graphischen Anzeige. Änderungen des Nutzers in der GUI werden an den passenden Konfig-Manager weitergegeben und andersherum erhält die GUI über den Controller immer den aktuellsten Zustand der Konfiguration.

View Die View visualisiert die Daten der Konfigurationen, damit sie bequem vom Nutzer angepasst werden können. Sie ist nach dem MVC-Pattern streng von den anderen Komponenten getrennt.

JSON-Parser Der JSON-Parser serialisiert bzw. deserialisiert die Konfigurationen, um sie zu speichern, bzw. zu laden. Er erhäkt von den Konfig-Managern ein Konfigurationsobjekt und serialisiert dieses, damit es im Filesystem gespeichert werden kann oder lädt eine Konfiguration aus einer Datei und gibt sie an den passenden Konfig-Manager weiter.

4.3 Zuordnung der Funktionalen Anforderungen

Die funktionalen Anforderungen gemäß dem Pflichtenheft werden den Komponenten folgendermaßen zugeteilt:

Komponente	Abgedeckte funktionale Anforderungen
View	FA70
Team-Konfig-Manager	FA53, FA71, FA72
Partie-Konfig-Manager	FA54, FA71, FA72
JSON-Konverter	FA71