Verteilte Systeme Praktikum

### Middleware Dokumentation

### Wintersemester 2022/2023

### Gruppe: Beta 5

#### Dominik Martin, Can Heintze, Dominik Mueller

# 

# Einführung und Ziele

## Aufgabenstellung

**Inhalt**

Dieses Dokument beschreibt die Middleware der Advanced Client – Server von Tron.

Die Middleware soll folgende Anforderungen erfüllen:

### Use Cases

**Nummer: UC-1**

**Titel:** Nameserver

**Beschreibung:**

* Die Middleware benötigt einen Nameserver, bei dem sich Komponenten mit ihren Methoden registrieren können.
* Der Nameserver sollte dann alle Methoden kennen und wo sie zu finden sind.
* Andere Komponenten sollen dann Anfragen an den Nameserver stellen können, um zu erfragen, wo man die jeweiligen registrierten Methoden findet

**Nummer: UC-2**

**Titel:** Marshaling & unmarshiling

**Beschreibung:**

Die Middleware soll Nachrichten so formatieren, dass sie über das Netzwerk geschickt werden können um beispielsweise Methodenaufrufe zu initieren. Diese Nachrichten sollen einmal “gepackt” und auch wieder “entpackt” werden.

**Nummer: UC-3**

**Titel:** Remote Invocation

**Beschreibung:**

* Die Middleware soll Nachrichten über das Netzwerk verschicken können. Die Nachrichten sollen Methodenaufrufe beinhalten. Die Nachrichten sollen im richtigen format geschickt werden und beim Empfänger einen methodenaufruf auslösen.
* Dabei soll sowohl Synchrone als auch asynchrone aufrufe gedeckt werden.
* Dabei sollen die nachrichten per TCP versendet werden

## Motivation

Die wesentliche Motivation für uns, diese Middleware zu implementieren ist es, die PVL zu erhalten. Weitere Motivationspunkte wären aber auch, neues zu lernen und unser bisheriges Wissen zu vertiefen.

## Qualitätsziele

|  |
| --- |
| **Qualitätsziele:** |
| Gut definierte Schnittstellen |
| Kompatibilität zu einer anderen Gruppe (Mindestens zwei Teams müssen miteinander spielen können) |
| Fehlertoleranz (Wenn ein Spieler abstürzt, egal welcher Spieler, dann geht das Spiel trotzdem weiter) -> Stabilität |
| Die Transparenz Kriterien sollten so gut wie möglich erfüllt werden. |

## Stakeholder

**Inhalt**

Unsere Stakeholder sind die Entwickler (Studenten), der Kunde (Professor) und die Spieler (Studenten).

Die Stakeholder mit deren Kontakt werden in der unteren Tabelle aufgelistet:

| Rolle | Kontakt | Erwartungshaltung |
| --- | --- | --- |
| *Entwickler* | [Dominik.martin@haw-hamburg.de](mailto:Dominik.martin@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Entwickler* | [*Can.heintze@haw-hamburg.de*](mailto:Can.heintze@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Entwickler* | [*Dominik.mueller@haw-hamburg.de*](mailto:Dominik.mueller@haw-hamburg.de) | *Ein gutes Spiel zu programmieren & die PVL zu erhalten* |
| *Kunde* | [*Martin.becke@haw-hamburg.de*](mailto:Martin.becke@haw-hamburg.de) | *Ein Lauffähiges Spiel, bei dem 6 Spieler gleichzeitig ein komplettes Spiel ohne Fehler durchspielen können.* |
| *Spieler / Nutzer* | *n/a* | *Ein funktionierendes Spiel spielen und dabei Spaß haben* |

# 

# Kontextabgrenzung

## Technischer Kontext

Diagram

Description automatically generated

# Lösungsstrategie

### Funktionale Anforderungen

|  |  |  |  |
| --- | --- | --- | --- |
| **Methoden Signatur** | **Beschreibung** | **Klasse** | **UseCase** |
| Void register(String methodName, Array<String> parameter) | * Methode, damit sich der caller an den Nameservice verbindet * Registriert die jeweilige Methode, in einer map unter dem Port / IP Adresse vom jeweiligen Caller | NameServer: RunnableTCPWorker | UC-1 |
| Array<String> [Ip, Port] query(String methodName, Array<String> parameter) | * Methode, damit der Nameservice die richtige Komponente findet * Gibt den port & die IP Adresse zurück, an welcher Stelle die Methode gefunden wird | NameServer: RunnableTCPWorker | UC-1 |
| String receivePacket() | * Läuft in einem Thread und empfängt die Pakete im NameServer | NameServer: RunnableTCPWorker | UC-1 |
| Void sendPacket(String) | * Sendet ein Packet über einen outputStream | NameServer: RunnableTCPWorker | UC-3 |
| String marshal(String, long, Object…) | * Erstellt eine Nachricht in JSON-Format, welche dann über das Netzwerk geschickt werden kann * In der JSON-Nachricht sind folgende Informationen enthalten: methodenName, messageID, args | Marshaler | UC-2 |
| String marshalReturnValue(long, Object) | * Erstellt Nachrichten in JSON-Format, welche für den rückgabewert zuständig sind, bei synchronen Methoden aufrufen | Marshaler | UC-2 |
| JSONArray getJSONArray(Object[], MarshalArray) | * Gibt einen JSONArray zurück | Marshaler | UC-2 |
| JSONObject. getJSONObject() | * Gibt das Klassenattribut mapJSON als JSONObject zurück | Marshaler: MarshalMap | UC-2 |
| Void buildTypeString(Map) | * Baut aus einer Map einen String | Marshaler: MarshalMap | UC-2 |
| String getType() | * Gibt den typen des Inhalts der Map für das Marshaling zurück | Marshaler: MarshalMap | UC-2 |
| Void buildJSONObject(Map) | * Baut aus einer Map ein JSONObject | Marshaler: MarshalMap | UC-2 |
| String getReturnType() | * Getter für den returnType * Der return type stellt den Typen dar, welcher das gemarshalte Objekt beinhaltet | Marshaler: MarshalArray | UC-2 |
| Void setReturnType(String) | * Setter für den returnType | Marshaler: MarshalArray | UC-2 |
| JSONArray getJSONArray() | * Getter fur die jsonArray Variable | Marshaler: MarshalArray | UC-2 |
| Void setJSONArray(JSONArray) | * Setter fur die jsonArray Variable | Marshaler: MarshalArray | UC-2 |
| Void addLayers(int) | * Anhand der Dimensionen, werden die „[ ]“ Klammern gesetzt um den String | Marshaler: MarshalArray | UC-2 |
| Void createCompleteReturnType(Object[]) | * Anhand der gezählten Dimensionen, wird ein String erstellt mithilfe der addLayers Methode | Marshaler: MarshalArray | UC-2 |
| Int countDimensions(Object[]) | * Es werden die Ebenen / Dimensionen des übergangenen Arrays gezählt und zurück gegeben | Marshaler: MarshalArray | UC-2 |
| ReponseObject unmarshalClientStub(String) | * Der übergebene String in JSON Format, wird in ein ResponseObject umgewandelt (messageID, Rückgabewert) | Unmarshaler | UC-2 |
| RequestObject unmarshalServerStub(String) | * Der Übergebene String in JSON Format, wird in ein RequestObject umgewandelt (methodenname, Argumente, Argumenten Typen) | Unmarshaler | UC-2 |
| Object unmarshalObject(String, String) | * Der Übergebene String wird zurück in ein Java Objekt (int, short, … etc.) umgewandelt | Unmarshaler | UC-2 |
| Object[] unmarshalArray(String, JSONArray) | * Das übergebene JSONArray wird mithilfe dieser Methode in ein Object Array umgewandelt | Unmarshaler | UC-2 |
| Map unmarshalMap(JSONObject, String) | * Das übergebene JSONObject stellt eine Map dar. Diese Map wird mithilfe dieser Methode wieder in eine Java Map umgewandelt | Unmarshaler | UC-2 |
| String marshalQueryRequest(String) | * Mithilfe dieser Methode werden die Query Anfragen an den NameServer gestellt | NameServerMarshaler | UC-2 |
| String marshalRegisterRequest(String, int) | * Mithilfe dieser Methoden werden die Register Anfragen an den NameServer gestellt | NameServerMarshaler | UC-2 |
| String marshalQueryResponse(List<List<String>>) | * Mithilfe dieser Methode werden die Query Anfragen an den NameServer gestellt | NameServerMarshaler | UC-2 |
| NameServerRequestObject unmarshalRequest(String) | * Die übergebene Nachricht al String wird als NameServerRequestObject umgewandelt. * In dem NameServerRequestObject befindet sich der Methoden Typ und der Methoden Name | NameServerMarshaler | UC-2 |
| List<NameServerResponseObject> unmarshalResponse(String) | * Die übergebene Nachricht als String, wird in eine Liste von NameServerResponseObjects gepackt. * Dort befindet sich dann die IP, und der Port | NameServerMarshaler | UC-2 |
| String marshalNameServerRequest(String, String) | * Hier wird der Methodentyp und methodenName der übergebenen Methode in ein JSONObject gepackt | NameServerMarshaler | UC-2 |
| String getNameServerIP() | * Gibt die IP Adresse vom NameServer zurueck | MiddlewareConfig | UC-1 |
| Int getNameServerPort() | * Gibt den Port vom NameServer zurück | MiddlewareConfig | UC-1 |
| Void loadConfigFile(String) | * Ladet die verschiedenen Parameter, wie IP adresse des NameServers aus der Config File | MiddlewareConfig | UseCase Ubergreifend |
| Object invokeSynchronously(String, Object…) | * Ruft query auf, auf den Nameserver um die Informationen über IPs und Ports zu erhalten * Verpackt die Nachricht mithilfe des Marshalers * Sendet über einen TCP-Socket | ClientStub | UC-3 |
| Void invokeAsynchronously(String, TrnasportType, Object…) | * Ruft query auf, auf den Nameserver um die Informationen über IPs und Ports zu erhalten * Verpackt die Nachricht mithilfe des Marshalers * Sendet mit UDP bei zeitkritischen Aufrufen (Spiele Logik), sendet mit TCP bei nicht zeitkritischen aufrufen | ClientStub | UC-3 |
| ResponseObject invokeTCP(Strint, int byte[], String, bool) | * Versendet Nachrichten über einen TCP Socket | ClientStub | UC-3 |
| Socket initTCPSocket(String, int) | * Initialisiert einen TCP-Socket auf einem bestimmten Host & Port | ClientStub | UC-3 |
| List<String> lookup(String) | * Ruft query auf dem nameServer auf uns schaut nach einer Methode, mit dem jeweiligen Methodennamen * Die Methode, gibt dann die IPs und Ports zurück, an der sich die jeweilige Methode befindet | ClientStub | UC-1 |
| Void sendUDPPacket(byte[], int, String, int, DatagramSocket) | * Versendet über den UDP-Socket ein Packet | ClientStub | UC-3 |
| String receiveUDPPacket(DatagramSocket) | * Liest über den UDP-Socket ein empfangenes Packet ein und gibt dieses als String zurück | ClientStub | UC-3 |
| Void sendTCPPacket(byte[], DataoutputStream) | * Versendet über TCP ein Packet | ClientStub | UC-3 |
| String receiveResponseTCPPacket(BufferedReader) | * Liest über den inputStream das Packet und wandelt es in einen String um | ClientStub | UC-3 |
| Void registerMethod(String) | * Registriert die jeweilige Methode beim NameServer | ServerStub | UC-1 |
| Void startTCP() | * Startet die Interne Klasse für TCP mit einem Thread | ServerStub | UC-3 |
| Void startUDP() | * Startet die Interne Klasse für UDP mit einem Thread | ServerStub | UC-3 |
| String receivePacket() | * Empfängt Pakete über TCP | ServerStub: RunnableTCPWorker | UC-3 |
| Void sendPacket(String) | * Versendet ein Packet über TCP | ServerStub: RunnableTCPWorker | UC-3 |
| String processUDPPacket() | * Erstellt einen String aus den Daten vom DatagramPacket und der Länge des Paketes | ServerStub: RunnableUDPWorker | UC-3 |
| Void sendUDPPACKPacket(byte[], int) | * Versendet ein UDP packet über einen DatagramSocket | ServerStub: RunnableUDPWorker | UC-3 |
| Object callImplementation(String, Object…) | * Wendet die Argumente auf die jeweiligen Methode an und ruft diese auf | ImplManager | UC-3 |
| Bool isPrefixedArg(String, int) | * Prüft, ob eine Methode schon geprefexid ist | ImplManager | UC-3 |
| Bool isAsyncMethod(String) | * Prüft, ob eine Methode Asynchron ist | ImplManager | UC-3 |
| Void registerSyncImplementation(String, Function<Object[], Object>, bool[]) | * Registriert die jeweilige Methode auf dem NameServer über den ServerStub * Nur für Synchrone Methoden | ImplManager | UC-1 |
| Void registerAsyncImplementation(String, Function<Object[], Object>, bool[]) | * Registriert die jeweilige Methode auf dem NameServer über den ServerStub * Nur für Asynchrone Methoden | ImplManager | UC-1 |
| Object onKeyPressWrapper(Object..) | * Wrapper Methode für die onKeyPress Methode aus dem Tron Controller | IPlayerInputManagerImplWrapper | UC-3 |
| Object getValidKeysWrapper(Object…) | * Wrapper Methode für die getValidKeys methode aus dem Tron Controller | IPlayerInputManagerImplWrapper | UC-3 |
| Void registerWrapperMethods() | * Ruft registerSyncImplementation auf und registriert somit die Wrapper Methoden dieser Klasse | IPlayerInputManagerImplWrapper | UC-1/UC-3 |
| Object commitAndChangeToNextSceneWrapper(Object…) | * Wrapper Methode für die commitAndChangeToNextScene Methode aus dem Tron Controller | ISceneChangerImplWrapper | UC-3 |
| Object changeToPreviousSceneWrapper(Object…) | * Wrapper Methode für die changeToPreviousScene Methode aus dem Tron Controller | ISceneChangerImplWrapper | UC-3 |
| Object changeToNextSceneWrapper(Object…) | * Wrapper Methode für die changeToNextScene Methode aus dem Tron Controller | ISceneChangerImplWrapper | UC-3 |
| Void registerWrapperMethods() | * Ruft registerSyncImplementation auf und regisitriert somit die Wrapper Methoden dieser Klasse | IScenChangerImplWrapper | UC-1/UC-3 |
| Object showLobbyScreenWrapper(Object…) | * Wrapper Methode für die showLobbyScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Object showGameScreenWrapper(Object…) | * Wrapper Methode für die showGameScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Object showStartScreenWrapper(Object…) | * Wrapper Methode für die showStartScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Void registerWrapperMethods | * Ruft registerSyncImplementation auf und registriert somit die Wrapper Methoden dieser Klasse | IScreenHandlerImplWrapper | UC-1/UC-3 |

# Bausteinsicht

**Komponenten Diagramm auf oberster Ebene:**

Graphical user interface, diagram

Description automatically generated

**Klassendiagramme**

**Graphical user interface

Description automatically generated**

# Graphical user interface, website Description automatically generatedGraphical user interface, application, website Description automatically generated

# Graphical user interface, text Description automatically generatedGraphical user interface, timeline Description automatically generated

Graphical user interface, website

Description automatically generated

# Graphical user interface, text Description automatically generated

# Graphical user interface, text Description automatically generated

# Laufzeitschicht

**Sequenzdiagramme**

Diagram, text

Description automatically generated

Diagram

Description automatically generated

Graphical user interface, text, application, chat or text message

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

# Verteilungssicht

Dieses Diagramm visualisiert die Verteilungssicht. Computer 1 bis 6 stellen die maximal 6 Spieler dar. Dabei ist die View vom Tron Spiel und die Middleware (ohne die NameServer Komponente) als eine Jar auf jedem Computer, welcher mitspielen will, vorhanden.

Zusätzlich gibt es noch eine Jar für die Controller & Model Komponente aus dem Tron Spiel, welche in diesem Fall die Spiele Logik darstellen und eine Jar für die NameServer Komponente. Diese beiden Jars müssen nur auf einem der 6 Computer installiert sein und laufen (in der Grafik unten, sind diese Jars auf Computer 1, sie können aber genauso gut auf einen anderen Computer).

# Graphical user interface, application Description automatically generated

# Architekturentscheidungen

**Marshaler:**

In unserer Komponenten Architektur haben wir uns dafür entschieden, die Marshaler Komponente als eigene Komponente neben dem Client-, Application- und Serverstub zu bauen. Da sowohl ClientStub als auch ServerStub auf diese Komponente zugreifen müssen um Nachrichten Pakete zu „Paketieren“.

**Remote Procedure Calls:**

Für das Gruppen interne Zusammenspiel haben wir uns für Remote Procedure Calls über das Netzwerk entschieden. Die Nachrichten für die Remote Procedure Calls sind in JSON-Format.

Die RPC`s sehen wie folgt aus:

Anfragen an den NameServer:

*{*

*“methodType”:”register/query”,*

*“methodname”:”classname methodname param1 param2…”*

*}*

Der NameServer gibt folgende Antwort auf die query() Methode:

*{*

*“providers“:[*

*{*

*“ip“:“xx.xx.xxx.x“,*

*“port“:“xxx“*

*},*

*{*

*“ip“:“xx.xx.xxx.x“,*

*“port“:“xxx“*

*}*

*]*

*}*

JSON für den Methodenaufruf (callImplementation()):

{

“method\_name“:“xxxx“,

“args“:[

“param1“,

“param2“

],

“arg\_types“: [

“int“,

„String“

],

“msg\_id“:“0x123“

}

Falls ein Synchroner Methodenaufruf getätigt wird, kommt der Rückgabewert über folgende JSON zurück:

*{*

*“return\_type“:“int“,*

*“return\_value“:“xx“,*

*“msg\_id“:“0x124“*

*}*

**Wrapper Code Generator:**

Für die Wrapper Klassen im ApplicationStub, welche unsere Methoden aus den Interfaces (Tron-View, Tron-Controller) Wrappen, haben wir einen Code Generator genutzt.

Mittels der Annotation “@RemoteImplementation“ wird eine Wrapper Klasse für das jeweilige Interface generiert. Diese Annotation haben wir an den Interfaces gesetzt, welche über JAR Files hinweg kommunizieren. Also einmal an den “IScreenHandler“, “IPlayerInputManagerImpl“ & “ISceneChangerImpl“ Interfaces vom Tron Spiel.

Diese Wrapper Klassen benötigen wir, um die Remote Invocation zu realisieren. Dafür haben wir zusätzlich noch mit der Annotation “@RemoteInterface“ an denselben Interfaces RPC Invoker Klassen generieren lassen.