Verteilte Systeme Praktikum

### Tron Game 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 auch zeitkritische aufrufe mitbedacht werden (TCP / UDP)

## 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* |

# Randbedingungen

# 

# Kontextabgrenzung

## Fachlicher Kontext

## Technischer Kontext

# 

# 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 zurueck, an welcher stelle die methode gefunden wird | NameServer: RunnableTCPWorker | UC-1 |
| String receivePacket() | * Laeuft in einem Thread und empfaengt die Packete im NameServer | NameServer: RunnableTCPWorker | UC-1 |
| Void sendPacket(String) | * Sendet ein Packet ueber einen outputStream | NameServer: RunnableTCPWorker |  |
| String marshal(String, long, Object…) | * Erstellt eine Nachricht in JSON Format, welche dann ueber 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 fuer den rueckgabewert zustaendig sind, bei synchronen Methoden aufrufen | Marshaler | UC-2 |
| JSONArray getJSONArray(Object[], MarshalArray) | * Gibt einen JSONArray zurueck | Marshaler | UC-2 |
| JSONObject. getJSONObject() | * Gibt das Klassenattribut mapJSON als JSONObject zurueck | 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 fuer das Marshaling zurueck | Marshaler: MarshalMap | UC-2 |
| Void buildJSONObject(Map) | * Baut aus einer Map ein JSONObject | Marshaler: MarshalMap | UC-2 |
| String getReturnType() |  | Marshaler: MarshalArray | UC-2 |
| Void setReturnType(String) |  | Marshaler: MarshalArray | UC-2 |
| JSONArray getJSONArray() |  | Marshaler: MarshalArray | UC-2 |
| Void setJSONArray(JSONArray) |  | Marshaler: MarshalArray | UC-2 |
| Void addLayers(int) |  | Marshaler: MarshalArray | UC-2 |
| Void createCompleteReturnType(Object[]) |  | Marshaler: MarshalArray | UC-2 |
| Int countDimensions(Object[]) |  | Marshaler: MarshalArray | UC-2 |
| ReponseObject unmarshalClientStub(String) |  | Unmarshaler | UC-2 |
| RequestObject unmarshalServerStub(String) |  | Unmarshaler | UC-2 |
| Object unmarshalObject(String, String) |  | Unmarshaler | UC-2 |
| Object[] unmarshalArray(String, JSONArray) |  | Unmarshaler | UC-2 |
| Map unmarshalMap(JSONObject, String) |  | Unmarshaler | UC-2 |
| String marshalQueryRequest(String) |  | NameServerMarshaler | UC-2 |
| String marshalRegisterRequest(String) |  | NameServerMarshaler | UC-2 |
| String marshalQueryResponse(List<List<String>>) |  | NameServerMarshaler | UC-2 |
| NameServerRequestObject unmarshalRequest(String) |  | NameServerMarshaler | UC-2 |
| List<NameServerResponseObject> unmarshalResponse(String) |  | NameServerMarshaler | UC-2 |
| String marshalNameServerRequest(String, String) |  | NameServerMarshaler | UC-2 |
| String getNameServerIP() | * Gibt die IP Adresse vom NameServer zurueck | MiddlewareConfig | UC-1 |
| Int getNameServerPort() | * Gibt den Port vom NameServer zurueck | 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 * Ruft marshaling auf * Sendet mit TCP | ClientStub | UC-3 |
| Void invokeAsynchronously(String, TrnasportType, Object…) | * Ruft query auf * Ruft marshaling auf * 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) | * ??? | 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 zurueck, an der sich die jeweilige Methode befindet | ClientStub | UC-1 |
| Void sendUDPPacket(byte[], int, String, int, DatagramSocket) | * Versendet ueber den UDP Socket ein Packet | ClientStub | UC-3 |
| String receiveUDPPacket(DatagramSocket) | * Liest ueber den UDP Socket ein empfangenes Packet ein und gibt dieses als String zurueck | ClientStub | UC-3 |
| Void sendTCPPacket(byte[], DataoutputStream) | * Versendet ueber TCP ein Packet | ClientStub | UC-3 |
| String receiveResponseTCPPacket(BufferedReader) | * Liest ueber 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 fuer TCP mit einem Thread | ServerStub | UC-3 |
| Void startUDP() | * Startet die Interne Klasse fuer UDP mit einem Thread | ServerStub | UC-3 |
| String receivePacket() | * Empfaengt Packete ueber TCP | ServerStub: RunnableTCPWorker | UC-3 |
| Void sendPacket(String) | * Versendet ein Packet ueber TCP | ServerStub: RunnableTCPWorker | UC-3 |
| String processUDPPacket() | * Erstellt einen String aus den Daten vom DatagramPacket und der Laenge des Packetes | ServerStub: RunnableUDPWorker | UC-3 |
| Void sendUDPPACKPacket(byte[], int) | * Versendet ein UDP packet ueber einen DatagramSocket | ServerStub: RunnableUDPWorker | UC-3 |
| Object callImplementation(String, Object…) | * Wendet die argumente auf die jweilige Methode an und ruft diese auf | ImplManager | UC-3 |
| Bool isPrefixedArg(String, int) | * Prueft ob eine Methode schon geprefexid ist | ImplManager | UC-3 |
| Bool isAsyncMethod(String) | * Prueft ob eine Methode Asynchron ist | ImplManager | UC-3 |
| Void registerSyncImplementation(String, Function<Object[], Object>, bool[]) | * Registiert die jeweilige Methode auf dem NameServer ueber den ServerStub * Nur fuer Synchrone Methoden | ImplManager | UC-1 |
| Void registerAsyncImplementation(String, Function<Object[], Object>, bool[]) | * Registiert die jeweilige Methode auf dem NameServer ueber den ServerStub * Nur fuer Asynchrone Methoden | ImplManager | UC-1 |
| Object onKeyPressWrapper(Object..) | * Wrapper Methode fuer die onKeyPress methode aus dem Tron Controller | IPlayerInputManagerImplWrapper | UC-3 |
| Object getValidKeysWrapper(Object…) | * Wrapper Methode fuer die getValidKeys methode aus dem Tron Controller | IPlayerInputManagerImplWrapper | UC-3 |
| Void registerWrapperMethods() | * Ruft registerSyncImplementation auf und regisitriert somit die Wrapper Methoden dieser Klasse | IPlayerInputManager | UC-1/UC-3 |
| Object commitAndChangeToNextSceneWrapper(Object…) | * Wrapper Methode fuer die commitAndChangeToNextScene Methode aus dem Tron Controller | ISceneChangerImplWrapper | UC-3 |
| Object changeToPreviousSceneWrapper(Object…) | * Wrapper Methode fuer die changeToPreviousScene Methode aus dem Tron Controller | ISceneChangerImplWrapper | UC-3 |
| Object changeToNextSceneWrapper(Object…) | * Wrapper Methode fuer 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 fuer die showLobbyScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Object showGameScreenWrapper(Object…) | * Wrapper Methode fuer die showGameScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Object showStartScreenWrapper(Object…) | * Wrapper Methode fuer die showStartScreen Methode aus der Tron View | IScreenHandlerImplWrapper | UC-3 |
| Void registerWrapperMethods | * Ruft registerSyncImplementation auf und regisitriert somit die Wrapper Methoden dieser Klasse | IScreenHandlerImplWrapper | UC-1/UC-3 |

# Bausteinsicht

**Komponenten Diagramm auf oberster Ebene:**

Graphical user interface, application, Teams

Description automatically generated

**Klassendiagramme**

**Graphical user interface

Description automatically generated**

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

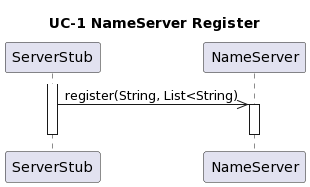
Graphical user interface, website

Description automatically generated

# Graphical user interface, text Description automatically generated

# Laufzeitschicht

**Sequenzdiagramme**



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, installiert.

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

### Zusammenspiel Gruppen intern:

* RPC Calls
* Code Generator fuer Wrapper (Applicationstub) ueber Annotations

### Zusammenspiel Beta 5 – Gamma 8

* Außenstehender REST Server, welcher input aufnimmt
  + Model / controller nehmen sich diesen input und berechnen jeweils die veraenderungen auf dem Board und verteilen die normal gruppen intern ueber RPC Calls
  + Tick mechanik