

Testat: Verteiltes Blackboard

Projektarbeit zur Vorlesung Verteilte Systeme

an der Fakultät für Technik
im Studiengang Informatik

an der
DHBW Ravensburg

Verfasser: Dirk Hattemer (2226744)
Tim Fassbender (7267367)
Manuel Kreft (8384857)
Tim Heckenberger (3111136)
Alexander Müller (2101133)
Kai Weinmann (9919522)

Dozent: Thomas Vogt

Abgabedatum: 19.06.2022

Selbständigkeitserklärung

gemäß Ziffer 1.1.13 der Anlage 1 zu §§ 3, 4 und 5 der Studien- und Prüfungsordnung für die Bachelorstudiengänge im Studienbereich Technik der Dualen Hochschule Baden-Württemberg vom 29.09.2017 in der Fassung vom 25.07.2018.

Wir versichern hiermit, dass wir diese Seminararbeit mit dem Thema

Testat: Verteiltes Blackboard

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Dirk Hattemer

Tim Fassbender

Manuel Kreft

Tim Heckenberger

Alexander Müller

Kai Weinmann

Friedrichshafen, 19.06.2022

Inhaltsverzeichnis

Abbildungsverzeichnis	I
1 Anforderungsanalyse	1
2 Architekturbeschreibung	2
2.1 Modellierung des Client-Server Modells	3
2.1.1 Systemdiagramm	3
2.1.2 Komponentendiagramm	3
2.2 Schnittstellen	5
2.2.1 CREATE_BLACKBOARD	6
2.2.2 DISPLAY_BLACKBOARD	7
2.2.3 CLEAR_BLACKBOARD	8
2.2.4 READ_BLACKBOARD	9
2.2.5 GET_BLACKBOARD_STATUS	10
2.2.6 LIST_BLACKBOARDS	11
2.2.7 DELETE_BLACKBOARD	12
2.2.8 DELETE_ALL_BLACKBOARDS	13
3 Technologieentscheidung	14
3.1 Programmiersprache	14
3.2 RPC-Bibliothek	14
3.2.1 Gründe für die Wahl der Bibliothek	14
3.2.2 Funktionsweise von RPyC	15
3.3 Nebenläufigkeitstransparenz	16
4 Anwendung	18
4.1 Voraussetzungen	18
4.2 Anleitung Server	18
4.3 Anleitung Client	19
4.4 Timeouts	22
5 Testing	23

Abbildungsverzeichnis

2.1	Systemdiagramm „Verteiltes Blackboard“	3
2.2	Komponentendiagramm „Verteiltes Blackboard“	4
4.1	Client-Interface	19

1 Anforderungsanalyse

Ziel dieser Arbeit ist es, ein verteiltes Blackboard zu Konzeptionieren und als Prototyp umzusetzen. Hierfür sollen über einen Server Dienste zur Erstellung und Verwaltung von einzelnen Blackboards realisiert werden. Dabei soll es Nutzern möglich sein, n verschiedene Blackboards zu erstellen, welche einzeln ansprechbar und verwaltbar sind. Hierbei soll die Validität der im Blackboard befindlichen Informationen zeitlich limitiert sein. Der Zugriff auf die Dienste des Servers soll über einen Client erfolgen.

Besonderer Fokus liegt hierbei auf der Verteilungstransparenz eines verteilten Systems, speziell drei Arten von Transparenz: Zugriffstransparenz, Nebenläufigkeitstransparenz und Fehlertransparenz.

Ein weiterer Fokus besteht auf dem Schutz der Konsistenz der im Blackboard gespeicherten Daten. So soll es n verschiedenen Nutzern gleichzeitig möglich sein, lesend wie schreibend auf den Blackboard-Server zuzugreifen. Hierbei muss sichergestellt werden, dass keine illegalen Operationen erfolgen, die das Blackboard in einen unsicheren Zustand überführen.

Als Sicherheitseinrichtung sollen die Anfragen mit Quelladresse in einem Log abgespeichert werden. Weitere Sicherheitsanforderungen z. B. Verschlüsselung sind nicht zu beachten.

2 Architekturbeschreibung

Ziel dieses Kapitels ist die Auswahl eines Architekturmusters und einem Design für die Blackboard-Applikation. Als Grundlage dient hierbei die Aufgabenstellung des Testats, welche bereits in Kapitel *Anforderungsanalyse* (1) herausgearbeitet wurde.

Die Anforderungen des Testats beinhalten einen Server als Einzelinstanz, welcher beliebig vielen Clients Ressourcen zur Verfügung stellen kann. Konkret soll der Server Funktionen bieten, um Blackboards zu verwalten. Der Server stellt hierbei eine zentrale Instanz dar, auf die die Clients zugreifen können, um die Blackboards zu bearbeiten und zu lesen.

Ein nahe liegendes Architekturmuster, welches die zuvor genannten Anforderungen erfüllt, ist das Client-Server-Modell. Dieses gehört der Kategorie der Schichtenarchitekturen an, da es sich um einen zweischichtigen Architekturansatz handelt. Hierbei wird auf dem Server ein Prozess implementiert, der zentral Dienstleistungen bereitstellt. Dahingegen ist der Client in der Lage, die implementierten Dienstleistungen des Servers anzufordern und somit dem User zur Verfügung zu stellen. Das Client-Server-Modell basiert somit auf einem Anforderungs-/Antwort-Verhalten und wird über die bereitgestellte Infrastruktur durch ein verbindungsloses oder verbindungsorientiertes Protokoll realisiert.

Die Auswahl dieser Architektur bringt für die im Rahmen dieses Testats geforderten Anforderungen mehrere Vorteile mit sich.

Erstens ermöglicht eine Client-Server-Architektur eine beliebige horizontale wie vertikale Skalierung. Hierdurch können die Clients beliebig skaliert und mit den implementierten Services versorgt werden. Zur Verbindung neuer Clients wird nur die entsprechende Client-Software (z. B. eine Konsolenanwendung) auf dem Client-Gerät und die IP-Adresse des Servers sowie der genutzte Port benötigt.

Ein weiterer Vorteil ist, dass die Anforderungen des Clients bezüglich dessen Rechenleistung gering sind, da dieser nur für simple Anfragen an den Server und die Visualisierung dessen Antworten eingesetzt wird. Jegliche Verarbeitung findet auf dem Server statt.

Risiken der Client-Server-Architektur wie Single-point-of-failure-Zwischenfälle können durch einen redundanten Einsatz der Server-Hardware und -Konfiguration ausgeschlossen werden. Da im Rahmen dieser Projektarbeit eine Redundanz nicht vorausgesetzt wird, soll hiermit nur aufzeigt werden, dass eine Erweiterung für ein Produktivsystem mit erhöhter Zuverlässigkeit gegeben sein können.

Als Implementierung des Client-Server-Modells wird Remote Procedure Call (RPC) verwendet. Dies ermöglicht den Clients, Funktionen des Servers aufzurufen, als wären diese ein Bestandteil des lokalen Systems. Wie bei einer lokalen Funktion können bei einem RPC-Funktionsaufruf Argumente übergeben werden. Ebenso sind Rückgabewerte möglich.

2.1 Modellierung des Client-Server Modells

Das folgende Kapitel soll das umgesetzten Architekturmuster der Blackboard Applikation aufzeigen. Die aufgezeigten Modelle sollen der Übersicht über das Gesamtsystem sowie der Darstellung der enthaltenen Komponenten dienen.

2.1.1 Systemdiagramm

Das Systemdiagramm gibt einen Überblick über den Aufbau der verteilten Applikation und die Implementierung des Client-Server Modells. Beliebige viele Clients können auf die durch den Server bereitgestellten Services zugreifen und diese in Anspruch nehmen. Die einzelnen bereitgestellten Services werden erst in Kapitel *Komponentendiagramm* (2.1.2) in der Grafik 2.2 aufgezeigt.

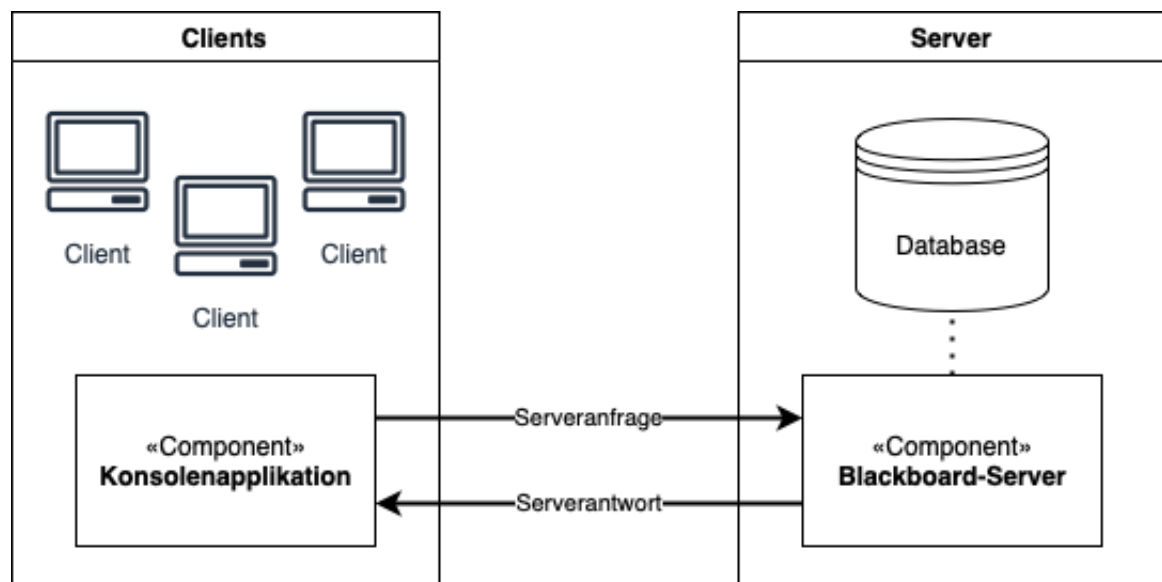


Abb. 2.1: Systemdiagramm „Verteiltes Blackboard“

2.1.2 Komponentendiagramm

Die Applikation besteht aus drei Komponenten, der Konsolenapplikation auf den Clients sowie der Blackboardanwendung und der Speicherkomponente auf dem Server. Die Umsetzung einer datenerhaltenden Schicht stellte keine Anforderung des Testats dar, wurde jedoch zusätzlich implementiert. Derzeit erfolgt die Speicherung als JSON-Datei, welche alle Inhalte und Statusinformationen eines Blackboards beinhaltet. Alternativ könnte dies ebenfalls über eine relationale oder nicht-relationale Datenbank erfolgen.

Die Serveranwendung lässt sich in weitere Subkomponenten unterteilen, welche die einzelnen Funktionalitäten der Blackboard-Applikation implementieren.

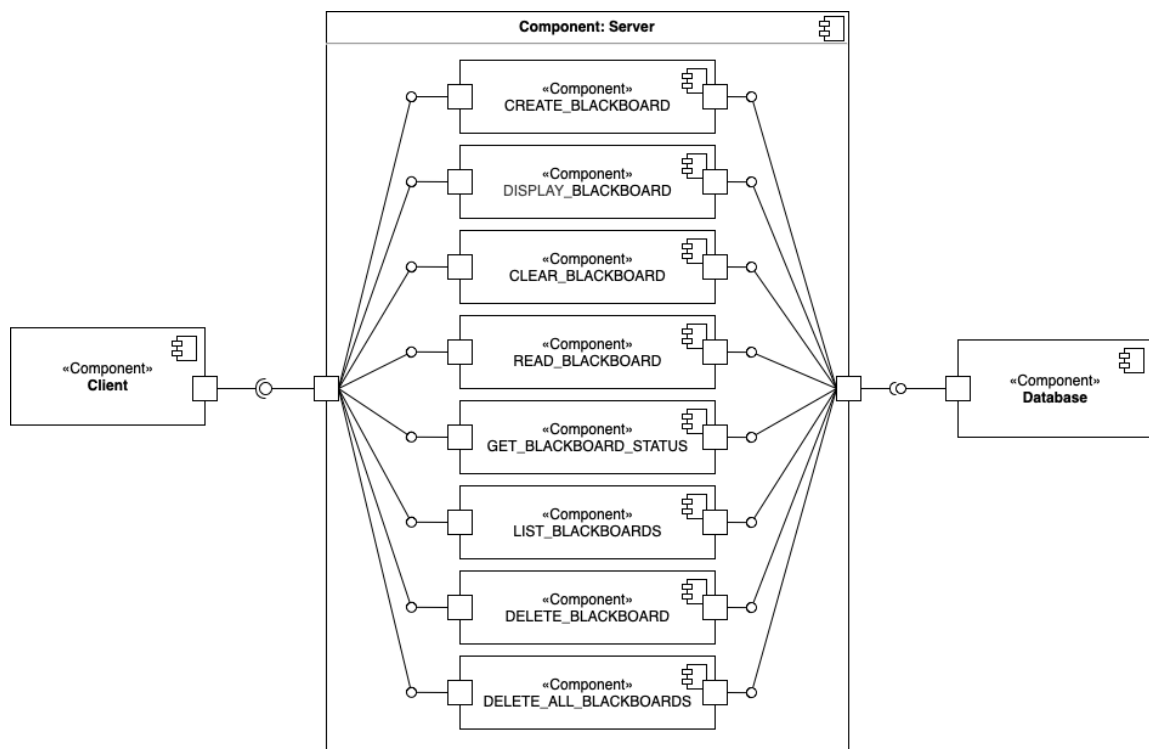


Abb. 2.2: Komponentendiagramm „Verteiltes Blackboard“

2.2 Schnittstellen

Die Umsetzung der Schnittstellen für die Anwendung entspricht größtenteils den vorgeschlagenen Schnittstellen der Aufgabenstellung des Testats. Eine detaillierte Beschreibung Schnittstellen ist im Folgenden dargestellt. Die Wertebereiche der nachfolgend gelisteten Parameter sind, wenn nachfolgend nicht explizit anders beschrieben, nicht manuell eingeschränkt. Somit gelten die standardmäßigen Einschränkungen der Programmiersprache Python.

2.2.1 CREATE_BLACKBOARD

Funktionsname	CREATE_BLACKBOARD
Kurzbeschreibung	Erstellt ein neues leeres Blackboard auf dem Server.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert noch nicht
Übergabeparameter	<p>(<i>String</i>) name - Name des neuen Blackboards</p> <p>(<i>Float, Integer oder String</i>) valid_sec - Dauer der Gültigkeit des Blackboards (0 bzw. „inf“ = unbegrenzte Gültigkeit), muss von Python in einen Float konvertiert werden können</p>
Rückgabeparameter	<p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über den Verlauf der Erstellung</p>
Post-Conditions	Blackboard wurde erstellt
Fehlerverhalten	<p>name already exists - Erstellung wird verhindert, da bereits ein Blackboard mit dem gleichen Namen existiert.</p> <p>invalid parameters - Erstellung wird wegen Angabe von ungültigen Parameter(typen) unterbunden.</p> <p>valid_sec < 0 - Erstellung wird aufgrund eines ungültigen Parameterwertes verhindert.</p> <p>timeout - Erstellung konnte aufgrund eines Timeouts nicht durchgeführt werden.</p>

2.2.2 DISPLAY_BLACKBOARD

Funktionsname	DISPLAY_BLACKBOARD
Kurzbeschreibung	Dient der Aktualisierung des Inhalts eines existierenden Blackboardes. Zusätzlich wird die Aktualitätsinformation (Zeitstempel) aktualisiert.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert
Übergabeparameter	<p>(<i>String</i>) name - Name des existierenden Blackboards</p> <p>(<i>String</i>) data - Die neue Information die auf das Blackboard geschrieben werden soll</p>
Rückgabeparameter	<p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über das Blackboard</p>
Post-Conditions	Blackboard-Inhalt wurde aktualisiert; Zeitstempel wurde aktualisiert
Fehlerverhalten	<p>board does not exist - Aktualisierung kann nicht durchgeführt werden, da kein Blackboard mit dem angegeben Namen existiert.</p> <p>timeout - Aktualisierung konnte aufgrund eines Timeouts nicht durchgeführt werden.</p>

2.2.3 CLEAR_BLACKBOARD

Funktionsname	CLEAR_BLACKBOARD
Kurzbeschreibung	Löscht den Inhalt des Blackboards. Blackboard wird dadurch ungültig.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert
Übergabeparameter	<i>(String)</i> name - Name des existierenden Blackboards
Rückgabeparameter	<i>(Boolean)</i> success - Information über Erfolg oder Misserfolg <i>(String)</i> message - Log-Nachricht mit Informationen über die Löschung der Inhalte des Blackboards
Post-Conditions	Inhalt des Blackboards wurde gelöscht
Fehlerverhalten	board does not exist - Entfernen des Inhaltes kann nicht durchgeführt werden, da kein Blackboard mit dem angegeben Namen existiert. timeout - Entfernen des Inhaltes konnte aufgrund eines Timeouts nicht durchgeführt werden.

2.2.4 READ_BLACKBOARD

Funktionsname	READ_BLACKBOARD
Kurzbeschreibung	Inhalt des Blackboards wird ausgelesen. Gültigkeit der Nachricht wird ebenfalls zurückgegeben.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert
Übergabeparameter	<p>(<i>String</i>) name - Name des existierenden Blackboards</p>
Rückgabeparameter	<p>Bei Erfolg (hierzu zählen auch Warnungen):</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) data - Inhalt des Blackboards</p> <p>(<i>Boolean</i>) is_valid - Gültigkeit des Boards</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über das Auslesen des Blackboards</p> <p>Bei Misserfolg:</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über das Auslesen des Blackboards</p>
Post-Conditions	
Fehlerverhalten	<p>board does not exist - Auslesen kann nicht durchgeführt werden, da kein Blackboard mit dem angegebenen Namen existiert.</p> <p>timeout - Auslesen konnte aufgrund eines Timeouts nicht durchgeführt werden.</p> <p>data invalid (<i>Warnung</i>) - Daten können ausgelesen werden, diese sind jedoch ungültig.</p> <p>is empty (<i>Warnung</i>) - Daten können ausgelesen werden, das Board besitzt jedoch keinen Inhalt.</p>

2.2.5 GET_BLACKBOARD_STATUS

Funktionsname	GET_BLACKBOARD_STATUS
Kurzbeschreibung	Gibt den Status des Blackboards zurück.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert
Übergabeparameter	<p>(<i>String</i>) name - Name des existierenden Blackboards</p>
Rückgabeparameter	<p>Bei Erfolg (hierzu zählen auch Warnungen):</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>Boolean</i>) is_empty - Information ob das Blackboard leer oder beschrieben ist</p> <p>(<i>Float</i>) entry_time - Zeitstempel der Aktualisierung</p> <p>(<i>Boolean</i>) is_valid - Gültigkeit des Boards</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über den Status des Blackboards</p> <p>Bei Misserfolg:</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über den Status des Blackboards</p>
Post-Conditions	
Fehlerverhalten	<p>board does not exist - Status des Blackboards kann nicht abgefragt werden, da kein Blackboard mit dem angegeben Namen existiert.</p> <p>timeout - Status des Blackboards konnte aufgrund eines Timeouts nicht abgefragt werden.</p>

2.2.6 LIST_BLACKBOARDS

Funktionsname	LIST_BLACKBOARDS
Kurzbeschreibung	Listet alle vorhandenen Blackboards auf.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden
Übergabeparameter	-
Rückgabeparameter	<p>Bei Erfolg (hierzu zählen auch Warnungen):</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>List of String</i>) list_of_boards - Liste mit allen Namen der Blackboards</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über die Auflistung der Blackboards</p> <p>Bei Misserfolg:</p> <p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über die Auflistung der Blackboards</p>
Post-Conditions	
Fehlerverhalten	<p>timeout - Liste aller Blackboards kann aufgrund eines Timeouts nicht erstellt werden.</p> <p>no boards found (<i>Warnung</i>) - Liste aller Blackboards kann erstellt werden, es gibt jedoch keine Blackboards.</p>

2.2.7 DELETE_BLACKBOARD

Funktionsname	DELETE_BLACKBOARD
Kurzbeschreibung	Löscht ein vorhandenes Blackboard und dessen Inhalt.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden; Angegebenes Blackboard existiert
Übergabeparameter	<i>(String)</i> name - Name des existierenden Blackboards
Rückgabeparameter	<i>(Boolean)</i> success - Information über Erfolg oder Misserfolg <i>(String)</i> message - Log-Nachricht mit Informationen über die Löschung des Blackboards
Post-Conditions	Blackboard wurde entfernt
Fehlerverhalten	board does not exist - Löschung des Blackboards kann nicht durchgeführt werden, da kein Blackboard mit dem angegebenen Namen existiert. timeout - Löschung des Blackboards konnte aufgrund eines Timeouts nicht durchgeführt werden.

2.2.8 DELETE_ALL_BLACKBOARDS

Funktionsname	DELETE_ALL_BLACKBOARDS
Kurzbeschreibung	Löscht alle vorhandenen Blackboards und deren Inhalte.
Pre-Conditions	Server verfügbar; Client ist mit Server verbunden
Übergabeparameter	
Rückgabeparameter	<p>(<i>Boolean</i>) success - Information über Erfolg oder Misserfolg</p> <p>(<i>String</i>) message - Log-Nachricht mit Informationen über die Löschung der Blackboards</p>
Post-Conditions	Alle Blackboards wurden entfernt
Fehlerverhalten	<p>timeout - Löschung der Blackboards konnte aufgrund eines Timeouts nicht durchgeführt werden.</p>

3 Technologieentscheidung

Nachfolgend werden die grundlegend Technologieentscheidungen der Projektgruppe und deren Hintergründe vorgestellt.

3.1 Programmiersprache

Primär wurde sich durch die umfassend vorhandene Kompetenz der Teilnehmer der Projektgruppe in der Programmiersprache „Python“ für diese Technologie entschieden. Weiterführend bietet Python die Vorteile des simplen Prototypings. Dies erlaubte schnell und unkompliziert erste Versionen des verteilten Blackboards zu konzipieren und half entsprechend bei der Auswahl des architektonischen Ansatzes. Ebenso wird durch Python eine hohe Lesbarkeit des Codes und somit eine bestmögliche Wartbarkeit erreicht. Weiterführend überzeugt Python durch die einfache Portabilität auf diverse Plattformen und Geräte. Ebenso erleichtert die Auswahl und Anwendbarkeit der zahlreichen Bibliotheken den Umgang mit komplexen Problemen, wie beispielsweise RPCs.

Pythons größte Schwäche, die verhältnismäßig langsame Ausführungszeit, spielt nach Einschätzung des Umfangs des Projekts keine signifikante Rolle und ist entsprechend kein Problem für das Projekt.

3.2 RPC-Bibliothek

Als Technologie zur Umsetzung der RPCs wird die Bibliothek „RPyC“ verwendet. Im Folgenden wird beschrieben, warum gerade diese Bibliothek gewählt wurde. Anschließend wird die grundlegende Funktionsweise der Bibliothek erläutert. Hierbei werden nur diejenigen Aspekte beleuchtet, welche bei der Umsetzung des Projektes beachtet wurden.

3.2.1 Gründe für die Wahl der Bibliothek

RPyC wurde gewählt, da diese Bibliothek viele Vorteile bietet. Nachfolgend werden die wichtigsten dieser Vorteile genannt und kurz beschrieben.

Erstens ermöglicht die Bibliothek eine transparente Verwendung von RPCs. Dies bedeutet, dass abgesehen von der Initialisierung Remoteobjekte so verwendet werden können, als wären sie lokal vorhanden.

Des Weiteren ist die Bibliothek plattformunabhängig. So kann die Bibliothek sowohl auf 32-Bit als auch 64-Bit-Systemen und im Little-Endian- sowie Big-Endian-Format verwendet werden. Auch werden alle gängigen Betriebssysteme (Windows, Linux, MAC und Solaris) unterstützt.

Außerdem arbeitet RPyC sehr effizient, was daran liegt, dass ein kompaktes binäres Protokoll zur Kommunikation genutzt wird, dass nur einen geringen Overhead besitzt.

Vorhanden sind auch mehrere Sicherheitsmerkmale, die optional genutzt werden können. Hierunter fallen eine sichere Kommunikation via SSH oder TLS sowie berechtigungsbaasierte Sicherheitsrichtlinien. Da solche Sicherheitsmerkmale jedoch nicht gefordert sind, wurden diese im Rahmen des Testats nicht verwendet, könnten jedoch im Nachhinein ohne Komplikationen hinzugefügt werden.

Neben den genannten Vorteilen bietet RPyC noch weitere Vorzüge, die jedoch aufgrund ihrer geringen Relevanz für das Projekt näher erläutert werden.

3.2.2 Funktionsweise von RPyC

RPyC verfolgt zur Realisierung der RPC-Funktionalität eine objektorientierte Strategie. Dies bedeutet, dass anstelle von Funktionen Methoden zum Aufrufen angeboten werden können. Dabei ist die RPyC Klasse namens „Service“ von Bedeutung. Von dieser Klasse muss jene serverseitigen Klassen erben, die die Methoden zum Aufrufen bereitstellt. Für jeden Client, der eine Verbindung zum Server aufbaut, wird ein Objekt dieser Klasse erstellt.

Direkt nach der Erstellung des Objektes wird die von der Klasse Server vererbte Methode „on_connect“ aufgerufen. Diese bekommt als Parameter eine Referenz auf ein Objekt, welches die Verbindung zum Client repräsentiert. Durch ein Überschreiben der Methode kann aus diesem Verbindungsobjekt die IP-Adresse und der Port des verbundenen Clients ausgelesen werden. Ein Speichern dieser Informationen in einem Attribut des Objekts ermöglicht es, jeden folgenden Methodenaufruf einem Client zuzuweisen. Hierdurch wird ein effektives Loggen möglich. Ähnlich zur „on_connect“-Methode gibt es eine „on_disconnect“-Methode, welchem beim Abbruch der Verbindung aufgerufen wird.

Generell werden nicht alle Methoden, die innerhalb der Klasse definiert sind, für einen Remoteaufruf angeboten, sondern nur jene mit dem Präfix „exposed_“. Alle anderen Methoden sind für einen Client nicht sichtbar.

Um die Klasse bzw. die enthaltenen Methoden bereitzustellen, muss ein Server-Objekt erstellt werden. Hierbei sind von RPyC mehrere verschiedene Server-Klassen definiert. Im Rahmen dieses Testates wird die Klasse „ThrededServer“ verwendet, welche durch Threads mehrere Verbindungen gleichzeitig erlaubt. Zum Erstellen eines zugehörigen Objektes muss lediglich die Klasse, welche die aufrufbaren Methoden enthält, sowie der zu verwendende Port angegeben werden. Nach dem Erstellen des Objektes kann der Server durch einen Methodenaufruf gestartet werden. Hierdurch ist der Server für Clients erreichbar.

Auf der Clientseite wird zum Verbindungsaufbau die Funktion „connect“ der RPyC-Bibliothek verwendet. Diese Funktion gibt anhand einer übergebenen IP-Adresse und eines Ports ein Verbindungsobjekt zurück, welches im Anschluss verwendet werden kann, um die bereitgestellten Methoden des Servers zu nutzen. Hierbei können die Remoteme-

thoden wie lokal definierte Methoden aufgerufen werden.

Trotz der hohen Transparenz der Bibliothek ist zu beachten, dass neben den Fehler, die auch bei lokalen Methodenaufrufen auftreten können (beispielsweise unerwartete Argumente oder Rückgabewerte), ebenso Verbindungsprobleme wie Timeouts oder Verbindungsabbrüche berücksichtigt und abgefangen werden müssen.

3.3 Nebenläufigkeitstransparenz

Zur Realisierung der Synchronisierung mehrerer Zugriffe auf die Ressourcen des Servers und damit zur Lösung des Nebenläufigkeitsproblems wird auf einen Lock-Mechanismus zurückgegriffen. Dieser sperrt die geteilten Ressourcen für den Zugriff mehrerer Teilnehmer und erlaubt lediglich einen einzelnen zeitgleichen Zugriff, während die anderen Teilnehmer darauf warten müssen, dass die angeforderte Ressource wieder freigegeben wird.

Die Implementierung des Mechanismus erfolgt unter Verwendung des „threading“-Moduls der Python-Standardbibliothek. Dieses stellt abstrakte Schnittstellen zur Verfügung, welche eine einfache Implementierung verschiedener Synchronisierungsmechanismen erlauben. So beinhaltet das Modul bereits eine verwendbare Lock-Funktionalität, welche die Verwaltung zeitgleicher Ressourcenzugriffe synchronisiert. Zusätzlich integriert die Bibliothek bereits automatisch einen Mechanismus, welcher die Verwaltung mehrerer wartender Teilnehmer organisiert. Dieser Mechanismus sorgt dafür, dass bei Freigabe des Locks nur einer der wartenden Teilnehmer den Lock „erhält“ und auf die Ressource zugreifen kann.

Die Bedienung des Lock-Objektes erfolgt über die Methoden „acquire“ und „release“, welche jeweils den maßgebenden Zustand („locked“ oder „unlocked“) steuern. Das Lock-Objekt kann durch einen Thread über die Methode „acquire“ vom „unlocked“- in den „locked“-Zustand gebracht werden und nur von diesem durch die „release“-Methode wieder freigegeben und in den „unlocked“ Zustand gebracht werden. Solange es sich in diesem gesperrten Zustand befindet, ist die Anforderungsmethode blockiert, wodurch kein anderer Thread das Lock-Objekt anfordern und Zugriff auf die geschützte, zu synchronisierende Ressource erhalten kann.

Durch eine Kombination des Lock-Mechanismus der python-Standardbibliothek und dem Context-Manager-Protokoll, kann eine vereinfachte Implementierung der Synchronisierung realisiert werden. So kann durch eine sogenannten „with“-Anweisung der Zugriff auf eine Ressource verwaltet werden. So muss die Ressource nicht mehr explizit angefordert und freigegeben werden. Dies erfolgt automatisch beim betreten bzw. verlassen des „with“-Blocks. Das genannte Prinzip ist im Folgenden anhand vereinfachter Quellcode-Ausschnitte dargestellt. Hierbei sind beide Ausschnitt äquivalent zueinander.

```
1 with some_lock:  
2     #do something
```

```
1 some_lock.acquire()  
2 try:  
3     # do something  
4 finally:  
5     some_lock.release()
```

Im Rahmen des Projekts wurde anstatt des dargestellten standardmäßige Context-Manager-Verhaltens eine individuelle Funktion verwendet, die es erlaubt, bei der Verwendung des Context-Managers einen Timeout-Wert für den „acquire“-Versuch anzugeben. Dieser Timeout ist wichtig, um selbst bei einer hohen Serverauslastung ein Zurückkehren der aufgerufenen Funktion innerhalb eine maximale Zeit zu gewährleisten.

4 Anwendung

Nachfolgend wird eine ausführliche Anleitung zum Erstellen und Ausführen der Anwendung im Terminal gegeben. Diese setzt sich nicht weiter mit den verwendeten Technologien auseinander, sondern soll ein reines Handbuch für Nutzer ohne technisches Verständnis darstellen.

4.1 Voraussetzungen

Um die Anwendung ausführen und nutzen zu können, muss auf den genutzten Maschinen, also server- und clientseitig, Python installiert sein. Hierbei sollte die Version 3.10 genutzt werden, da unter dieser Version die Entwicklung der Anwendung erfolgte und somit eine reibungslose Ausführung gewährleistet wird.

Weiter muss die Python Bibliothek „RPyC“ installiert werden. Sofern Python mit Paket-Manager „pip“ installiert ist, erfolgt die Installation der Bibliothek mit folgendem Befehl:

```
python -m pip install rpyc
```

4.2 Anleitung Server

Der Server lässt sich mit folgendem Befehl starten:

```
python blackboard_server.py -p <port_number>
```

Bzw.:

```
python blackboard_server.py --port <port_number>
```

Dabei kann dem Server der gewünschte Port, auf welchem dieser antworten soll, übergeben werden. Sollte kein Port angegeben werden, wird als Default-Port „8080“ genutzt.

Grundlegend sollte der Server vor dem Client gestartet werden. Ist dies nicht der Fall, hat der Client keine Möglichkeit, sich mit dem Server zu verbinden und beendet nach entsprechend fehlgeschlagenem Versuch das Programm.

Sollte nun durch einen Client eine Anfrage an den Server gestellt und nach Prüfung auf Korrektheit ausgeführt werden, so erhält der Betreiber des Servers jegliche Informationen zur Anfrage im Terminal des Servers angezeigt. Zusätzlich wird eine Logdatei angelegt, welche jeglichen Aktionen und Anfragen an den Server loggt und nachvollziehbar macht. Diese Datei ist direkt im Projektordner zu finden unter dem Namen „log.csv“.

Im Log werden jegliche Aktionen des Servers sowie die erhaltenen Anfragen und ausgesendeten Antworten mit zugehörigen Parametern gespeichert. Dabei wird jeder Eintrag mit einem Zeitstempel versehen und chronologisch der Datei angehängt.

Ebenso wird nach der ersten Ausführung eine Datei „boards.json“ erstellt, welche den Zustand der Boards nach jeder Aktion speichert und bei Start des Servers entsprechend wieder geladen wird. Dies erlaubt eine Speicherung der Daten auch über einen Reboot, Crash oder Shutdowns hinweg.

4.3 Anleitung Client

Der Client kann durch das Starten der Datei „client.py“ ausgeführt werden. Hierbei muss der Port und die IP-Adresse spezifiziert werden, unter welcher der gewünschte Server zu erreichen ist. Dabei muss die Server-Adresse, wie unten dargestellt, zusammen mit dem Port als ein Parameter angegeben werden.

Sollten keine Angaben getroffen werden, ist der Standardwert „localhost:8080“.

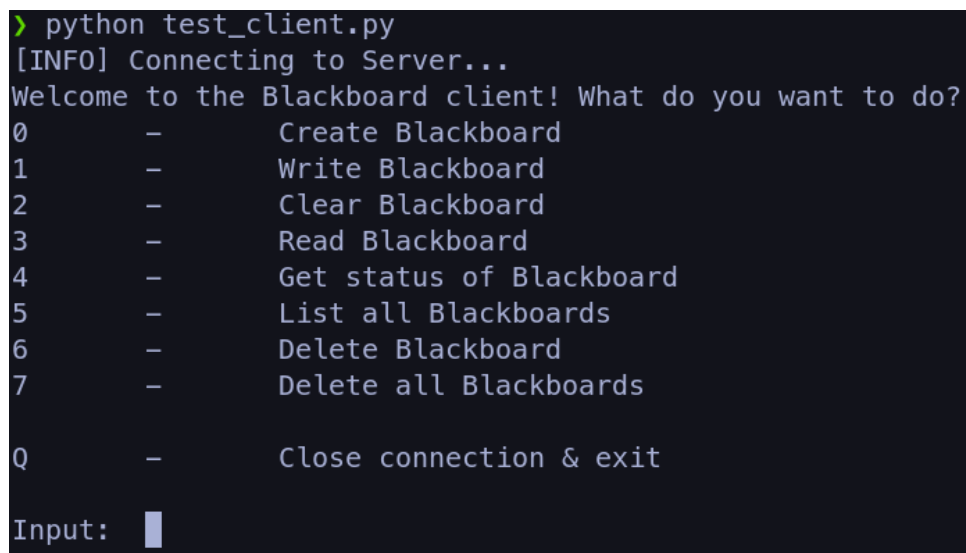
```
python client.py -s <server_address:port>
```

Bzw.:

```
python client.py --server <server_address:port>
```

Der Client versucht zuerst eine Verbindung zum Server aufzubauen. Sollte der Server noch nicht gestartet sein, so meldet der Client diesen Fehler und bricht anschließend ab.

Nachdem der Client erfolgreich eine Verbindung zum Server aufbauen konnte, wird der Anwender mit folgendem Interface begrüßt:



```
> python test_client.py
[INFO] Connecting to Server...
Welcome to the Blackboard client! What do you want to do?
0      -      Create Blackboard
1      -      Write Blackboard
2      -      Clear Blackboard
3      -      Read Blackboard
4      -      Get status of Blackboard
5      -      List all Blackboards
6      -      Delete Blackboard
7      -      Delete all Blackboards

Q      -      Close connection & exit

Input: █
```

Abb. 4.1: Client-Interface

Hier kann der Anwender durch Eingabe des gewünschten Befehls (der Aktionsnummer) die Aktion im Client starten. Anschließend werden für jeden einzelnen Befehl bei Bedarf weitere Eingaben des Anwenders angefordert, wie beispielsweise der Name des gewünschten Blackboards oder die zu schreibenden Informationen.

Durch die Eingabe „Q“ kann der Client beendet werden und die Verbindung zum Server wird geschlossen. Alternativ ist ein Beenden per Tastenkombination „CTRL + C“ möglich.

Nachfolgend werden die einzelnen Befehle im Detail erläutert und deren Parameter erklärt. Grundlegend erhält der Nutzer bei jedem dieser Befehle aussagekräftige Fehlermeldungen bei möglichen Falscheingaben. Fehler, die nicht direkt mit der Clientanfrage in Verbindung stehen, z. B. ein Fehler beim Schreiben der Server-Log-Datei werden dem Client nicht mitgeteilt.

CREATE_BLACKBOARD

Um ein neues Blackboard zu erstellen, muss im Client die Option „0“ gewählt werden. Anschließend folgen Fragen bezüglich des zu erstellenden Blackboards.

So muss zunächst der gewünschte Name des Boards angegeben werden. Dieser Name muss einzigartig sein und darf entsprechend nicht mit dem Namen eines bereits existierenden Boards übereinstimmen. Sollte bereits ein Board mit dem entsprechenden Namen existieren, wird nach dem Absenden des Befehls eine entsprechende Fehlermeldung zurückgegeben.

Weiterführend wird eine Zahl verlangt, die angibt, wie viele Sekunden die Informationen des Blackboards nach Aktualisierung valide bleiben. Diese muss größer oder gleich null gewählt werden und darf Dezimalstellen für die Angabe in Millisekunden besitzen. Sollte der Wert gleich null sein, so besitzen die Daten des Blackboard eine dauerhafte Gültigkeit, sofern Informationen vorhanden sind. Alternativ zum Wert null kann auch der String „inf“ verwendet werden.

Anschließend erhält der Nutzer eine Meldung über die Bestätigung der Erstellung des Blackboards oder eine entsprechende Fehlermeldung. Jedes neu erstellte und damit leere Blackboard wird mit dem Zustand „ungültig“ initialisiert, dies gilt auch für Blackboards mit unbegrenzter Gültigkeit.

Bezüglich der maximalen Anzahl an Blackboards und des Blackboardnamens gibt es keine Limitierungen. Für das Speichern des Blackboardnamens wird Unicode verwendet.

DISPLAY_BLACKBOARD

Um Informationen im Blackboard zu speichern, muss im Client die Option „1“ gewählt werden. Diese erlaubt, über den spezifischen Namen eines Blackboard Informationen zu diesem hinzuzufügen bzw. diese zu überschreiben, sofern das Blackboard existiert. Sollte dies nicht der Fall sein, wird der Nutzer darauf hingewiesen und die Aktion beendet. Die zu übergebenden Informationen werden als String behandelt und sind in der Länge nicht begrenzt. Der verwendete Zeichensatz ist wie beim Blackboardnamen Unicode und stellt somit keine Einschränkung dar.

Mit dem Schreiben der Informationen auf das Blackboard wird der Zeitstempel der Aktualisierung gesetzt und der Gültigkeitsstatus des Blackboards auf „gültig“ gesetzt.

CLEAR_BLACKBOARD

Um den Inhalt eines Blackboards zu löschen und das Blackboard somit zurückzusetzen, muss im Client die Option „2“ gewählt werden. Hiernach muss wiederum der Name des gewünschten Blackboards eingegeben werden.

Anschließend wird der Inhalt des Blackboards zurückgesetzt und der Gültigkeitsstatus des Blackboards auf „ungültig“ gesetzt. Dies gilt auch für Blackboards mit unbegrenzter Gültigkeit. Sollte kein Blackboard mit dem gewünschten Namen existieren, wird der Nutzer darauf hingewiesen.

READ_BLACKBOARD

Um die Daten eines Blackboards auszulesen, muss im Client die Option „3“ gewählt werden. Zusätzlich wird hierbei die Gültigkeit der Daten mit abgefragt. Hierbei wird die Eingabe des Namens des Boards verlangt. Sollte das gewünschte Board nicht existieren, wird die Aufgabe beendet und der Nutzer informiert.

Im Normalfall erhält der Client die Informationen des Boards sowie eine Aussage, ob die Informationen im zeitlichen Gültigkeitsbereich liegen. Also seit letzter Aktualisierung der Informationen die Grenzzeit, welche bei Erstellung des Boards definiert wurde, überschritten wurde oder nicht.

GET_BLACKBOARD_STATUS

Um einen vollständigen Status eines Blackboards zu erhalten, muss im Client die Option „4“ gewählt werden. Diese Aktion erlaubt die Abfrage, ob das Blackboard aktuell leer oder mit Informationen gefüllt ist. Zusätzlich erhält der Nutzer die Information über die Gültigkeit des Blackboards, sowie den Zeitstempel der letzten Aktualisierung.

LIST_BLACKBOARDS

Um die Namen aller Blackboards zu erhalten, muss im Client die Option „5“ gewählt werden. Diese werden dem Client als Liste übergeben und anschließend dargestellt.

DELETE_BLACKBOARD

Um ein einzelnes Blackboard zu löschen, muss im Client die Option „6“ gewählt werden. Hierbei muss der Nutzer nach Eingabe des Namens des zu löschenden Blackboards, die Operation durch eine Sicherheitsfrage bestätigen. Sollte diese Abfrage bestätigt werden,

wird das Blackboard vollständig gelöscht. Existiert kein Blackboard mit dem angegebenen Namen, wird dem Nutzer dies angezeigt.

DELETE _ALL_ BLACKBOARDS

Um alle existierenden Blackboards zu löschen, muss im Client die Option „7“ gewählt werden. Anschließend muss der Nutzer die Operation durch Bestätigung einer Sicherheitsfrage autorisieren. Bei positiver Antwort auf die Sicherheitsfrage werden alle Blackboards gelöscht.

4.4 Timeouts

Sollte der Server bereits durch andere Anfragen ausgelastet sein und somit kein Zugriff auf die Blackboards nötig sein, kehrt jede Funktion nach maximal 20 Sekunden mit einem Timeout-Fehler zurück. Liefert der Server innerhalb von 30 Sekunden gar keine Antwort, wird clientseitig ein Timeout geworfen, die anfrage abgebrochen und das Programm beendet. In beiden Fällen muss die Anfrage zu einem späteren Zeitpunkt wiederholt werden.

5 Testing

Um die Funktionsfähigkeit der Anwendung zu garantieren, wurden verschiedene Tests durchgeführt. Hierbei wurde sich auf den Server konzentriert, da hier die Konsistenz der Daten unabhängig von clientseitigen Prüfungen sichergestellt werden sollte. Diese Konsistenz wurde sowohl mit Unittests für alle Rückgabewerte der geforderten Server-Funktionalitäten geprüft als auch mittels separater Tests, welche die Konsistenz des globalen Zustandes aller Blackboards kontrollieren. Ebenso wurde das Erhalten der Datenkonsistenz nach dem Eintreten paralleler RPC-Anfragen geprüft. Hierfür wurde der Server mit Delays modifiziert, um das einfache stellen von Anfragen per Hand zu ermöglichen. Insgesamt wurde der in den Anforderungen geforderte Ablauf des Erstellens und Verwaltens von Blackboards mehrfach während der Entwicklung durchlaufen und schlussendlich auch im Kontext der fertiggestellten Anwendung teils über Unittests und teils über manuelle Plausibilitätsprüfungen verifiziert, wobei stets auf das Abdecken aller Code-Pfade geachtet wurde.

Das Durchführen der Unittests erfolgt durch das Ausführen des Konsolen-Befehls

```
python -m unittest discover -s tests -p '*test.py'
```

innerhalb des Wurzelverzeichnisses des Projektes und damit parallel zur Datei „blackboard_server.py“. Die Parameter „tests“ und „*test.py“ geben an, dass sich die Unittests im Unterverzeichnis „tests“ befinden und auf „test.py“ enden.