Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Oliver Behncke

Das Messen der Leistung von verteilten virtuellen Umgebungen

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Oliver Behncke

Das Messen der Leistung von verteilten virtuellen Umgebungen

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Christoph Klauck
Zweitgutachter: Lutz Behnke

Eingereicht am: 30. August 2013

Oliver Behncke

Thema der Arbeit
Das Messen der Leistung von verteilten virtuellen Umgebungen

Stichworte
Leistung, verteilte Systeme, verteilte virtuelle Umgebungen, Messen von Leistung

Kurzzusammenfassung
Diese Arbeit beschäftigt sich mit dem Aspekt der Leistung von verteilten virtuellen Umgebungen. Dabei steht im Mittelpunkt, wie diese Leistung gemessen werden kann. Durch die Analyse der Fachliteratur zur Bewertung der Leistung von verteilten virtuellen Umgebungen wird ein hierfür geeignetes Kriterium ausgewählt. Mit Hilfe dieses Kriteriums wird anhand einer Beispielumgebung ein Verfahren entwickelt, Leistung zu messen. Ziel ist es dabei, die Probleme, die beim Messen der Leistung auftreten, zu identifizieren.

Oliver Behncke

Title of the paper
Measuring the Performance of Distributed Virtual Environments

Keywords
performance, distributed systems, distributed virtual environments, measuring performance

Abstract
This thesis is concerned with the aspect of performance in distributed virtual environments. It evaluates how performance can be measured. An adequate criterion for the performance is identified by evaluating literature about performance in distributed virtual environments. Using this criterion, a model for measuring is derived from analysing an example environment. The goal of this study is to point out issues that come along with measuring performance.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Verteilte virtuelle Umgebungen (Englisch: distributed virtual environments, DVEs) sind in den letzten Jahren populär geworden. Allerdings sind sie nicht unter dieser Bezeichnung bekannt. Online-Spiele wie World of Warcraft sind DVEs und begeistern Millionen von Spielern. Sie tauchen in eine gemeinsame virtuelle Welt ab und interagieren miteinander.

Ermöglicht wird diese gemeinsame Erfahrung durch verteilte Systeme. Sie bieten die technische Grundlage, damit Spieler miteinander in Kontakt treten können und die Illusion einer gemeinsamen Welt entsteht. Verteilte Systeme bieten zwar die notwendigen technischen Möglichkeiten, sie bedeuten aber zudem Komplexität. Es bedarf der Kommunikation der beteiligten Parteien, um die Interaktionen in der virtuellen Welt zusammenzubringen. Ein gemeinsamer Zustand muss aus den Aktionen der einzelnen Nutzer erst geschaffen werden. Hiermit ist ein enormer Rechen- und Kommunikationsaufwand verbunden. Hinzu kommt, dass diese Berechnungen und die Kommunikation schnell erledigt werden müssen und so den Nutzern verborgen bleiben sollen, um die Illusion der gemeinsamen Welt nicht zu gefährden.

Es wird deutlich, dass eine DVE eine gewisse Leistungsfähigkeit mitbringen muss, um die an sie gestellten Anforderungen zu erfüllen. Es sollen möglichst viele Nutzer die virtuelle Welt erfahren können – und zwar ohne Einbußen im Spielvergnügen. Die virtuelle Welt soll skalierbar, d.h. um möglichst viele Nutzer erweiterbar, sein.

Doch wie ist diese Leistung einer DVE zu messen? In der Fachliteratur wird diese Frage selten alleine behandelt. Häufig stehen Verfahren zur Verbesserung der Skalierbarkeit im Mittelpunkt. Das Messen der Leistung dient dann nur zum Bewerten der Skalierbarkeit. Es wird aber nicht als eigenständiges Thema behandelt.

Aus diesem Umstand und der anfangs erwähnten Popularität von DVEs entstand die Idee, sich in einer Arbeit vollständig dem Messen der Leistung von DVEs zu widmen.

1.2 Aufgabenstellung

In dieser Arbeit steht dabei der gesamte Vorgang des Messens der Leistung im Mittelpunkt. Beim Messen benötigt man Kriterien zur Bewertung. Deshalb soll in einem Überblick über Kriterien zur Bewertung der Leistung von DVEs ein geeignetes Kriterium identifiziert werden.

Ein Bewertungskriterium reicht allein jedoch nicht aus, um die Leistung zu Messen. Für ein Kriterium müssen entsprechende Daten erhoben werden. Deshalb soll beispielhaft anhand einer DVE ein Modell zum Erheben solcher Daten, also dem tatsächlichen Messvorgang, erstellt werden. Indem das Modell experimentell evaluiert wird, sollen allgemeine Probleme beim Erheben geeigneter Daten herausgearbeitet werden.

Ziel sowohl des Überblicks über die Bewertungskriterien als auch des Modells und seiner Evaluation ist es, allgemeine Schwierigkeiten und Herausforderungen beim Messen der Leistung von DVEs aufzuzeigen. Dabei liegt der Schwerpunkt nicht auf der konkret untersuchten DVE, sondern auf den Aussagen, die über diese DVE hinaus getroffen werden können.

1.3 Aufbau der Arbeit

Der Aufbau der Arbeit stellt sich wie folgt dar. Zuerst wird ein allgemeiner Blick auf verteilte virtuelle Umgebungen geworfen. Dabei werden ihre Eigenschaften und der theoretische Hintergrund betrachtet. Es wird dargelegt, was DVEs ausmacht und welche theoretischen Konzepte mit ihnen einhergehen. Daraufhin wird anhand mehrerer Texte zur Skalierbarkeit von DVEs herausgearbeitet, wie geeignete Bewertungskriterien zum Messen der Leistung aussehen und wie diese erhoben werden können.

Das Wissen aus den theoretischen Betrachtungen und dem Literaturüberblick zum Thema Leistung wird im Anschluss verwendet, um ein Modell zu entwickeln und an einem konkreten Beispiel, dem massively multiplayer online role-playing game (MMORPG) Timadorus, die Leistung einer DVE zu messen. Eine auf Grundlage dieses Modells entworfene Software soll helfen, das Modell in der Praxis auszuprobieren und Schwierigkeiten beim Messen der Leistung herauszuarbeiten. Das Erheben von empirischen Daten über Timadorus ist dabei nur Mittel zur Evaluierung des Modells.

Die aus diesem Vorgehen gewonnenen Erkenntnisse sollen helfen, das Problem, Leistung in DVEs zu messen, verständlicher zu machen.

2 Grundlagen

Dieses Kapitel beschäftigt sich mit den grundlegenden Eigenschaften und dem theoretischen Hintergrund von DVEs. Im Folgenden sollen zuerst die Eigenschaften von DVEs herausgearbeitet werden. Es werden dabei Unterschiede u.a. zwischen verschiedenen DVEs aufgezeigt. Darüber hinaus werden die Anforderungen an DVEs dargelegt.

Ziel von DVEs ist es, dem Nutzer die Illusion einer kohärenten Welt zu geben. Eine kohärente Welt bedeutet, anders formuliert, dass der Weltzustand konsistent ist. Deshalb wird die Rolle der Konsistenz in DVEs in dem zweiten Abschnitt dieses Kapitels betrachtet. Dabei wird untersucht, wie von einem theoretischen Standpunkt Konsistenz aufgefasst werden kann. Aus diesen unterschiedlichen Auffassungen ergeben sich Modelle der Konsistenz. Es wird ein Überblick über die Modelle gegeben, die in DVEs zum Einsatz kommen. Da die Änderungen des Weltzustandes an die Nutzer weitergeleitet werden müssen, werden abschließend Verfahren zur Steuerung der Verteilung dieser Zustandsaktualisierungen vorgestellt.

2.1 Grundlegendes zu verteilten virtuellen Umgebungen

Virtuelle Umgebungen sind häufig Gesprächsthema – nicht nur in der Wissenschaft, sondern auch in nicht wissenschaftlichen Gebieten wie der Popkultur. Häufig wird dabei ausgelassen, was genau eine virtuelle Umgebung ausmacht. Miller (2011) definiert sie als *„computer simulation typically involving space and time“* (Miller, 2011, 13). Verteilte virtuelle Umgebungen unterscheiden sich von zentralisierten virtuellen Umgebungen hinsichtlich des Netzwerkaspektes. Die Verwaltung des Zustandes wird meist von mehreren Knoten bzw. Computern der virtuellen Umgebung übernommen. (Miller, 2011, 13)

Was unterscheidet aber nun eine DVE von einem verteilten System? Nach Tanenbaum and Steen (2006) lässt sich ein verteiltes System wie folgt definieren: *„A distributed system is a collection of independent computers that appears to its users as a single coherent system.“* (Tanenbaum and Steen, 2006, 2) Sowohl in einer DVE als auch in einem allgemeinen verteilten System interagieren Computer miteinander. Miller (2011) betont, dass es das Ziel einer DVE ist, für den Nutzer wie eine kohärente Welt zu erscheinen. (Miller, 2011, 13) Versteht man diese

Anforderung etwas allgemeiner, geht es also auch bei DVEs darum, dass mehrere miteinander interagierende Computer wie eine einzelnes kohärentes System erscheinen.

Der Aspekt, eine Simulation zu sein, die Raum und Zeit mit einbezieht, scheint jedoch ein besonderes Kriterium von DVEs zu sein, das in der Definition des verteilten Systems keine Erwähnung findet. DVEs scheinen diesen Aspekt nicht mit jedem verteilten System zu teilen. Ein solches System zur Simulation mit Zeit- und Raumbezug ist demnach eine konkrete Ausformung eines verteilten Systems. DVEs können dann als eine Teilmenge der verteilten Systeme verstanden werden. Zusammenfassend lässt sich eine DVE als ein System zur raum- und zeitbezogenen Computer-Simulation begreifen, das aus einer Menge von Rechnern besteht und für den Nutzer wie ein einzelnes kohärentes System erscheint.

2.1.1 Unterscheidungen

Populär sind DVEs aktuell vor allem durch Online-Spiele. Diese Spiele, welche große Zahlen von Benutzer online miteinander in Verbindung bringen, werden massively multiplayer online games (MMOGs) genannt. Unter den populären Spielen lassen sich laut Miller (2011) drei verschiedene Genres unterscheiden (Miller, 2011, 13):

- Ego-Shooter (Englisch: first-person shooter, FPS)
- Echtzeitstrategie (Englisch: real-time strategy, RTS)
- Rollenspiel (Englisch: role-playing game, RPG)

DVEs kommen aber nicht nur als Spiele zum Einsatz, sondern werden auch bei militärischen Simulationen oder als eine Art soziales Netz (z.B. Second Life) verwendet. (Miller, 2011, 13)

Nicht nur die Einsatzgebiete von DVEs sind vielfältig, sondern auch die verwendeten Architekturen. Prominente Beispiele, welche bei DVEs zum Einsatz kommen, sind Client-Server-Architekturen, Peer-to-Peer-Architekturen sowie die Verwendung einer Cloud als Ersatz für einen Server.

Ist ein Server Bestandteil der Architektur einer DVE, so ist die Aufgabe dieses Servers (oder auch mehrerer Server) in der virtuellen Welt laut Waldo (2008) eine zweifache. Zum einen ermöglicht der Server den Teilnehmern, miteinander zu interagieren. Zum anderen tritt er als Verwalter des wahren Zustands der virtuellen Welt auf. (Waldo, 2008, 12f)

2.1.2 Anforderungen an verteilte virtuelle Umgebungen

Die Anforderungen an die Leistung einer DVE unterscheiden sich stark nach ihrem Typ. Ein FPS verlangt deutlich schnellere Aktualisierungen des Spielzustandes als z.B. ein RPG, da Millise-

kunden über den Ausgang eines Kampfes entscheiden. (Barri et al., 2010, 341) Barri et al. (2010) weisen z.B. darauf hin, dass bei FPS-Spielen die einzelnen Sitzungen zwischen ein paar Minuten und mehreren Stunden dauern können, wohingegen bei RPGs und RTS-Spielen diese Sitzungen Tage bis Monate dauern können. Dementsprechend muss das zugrundeliegende System auch längere Sitzungen unterstützen, was unter anderem Herausforderungen bei der Implementierung von Persistenz mit sich bringt.

Ein Kernproblem teilen dabei alle Typen von DVEs: Kann die Leistung aufrechterhalten werden, wenn das System skaliert wird? Nach Tanenbaum and Steen (2006) ist Skalierbarkeit eines der Ziele von verteilten Systemen. Die beiden Autoren stellen folgende Dimensionen zum Messen der Skalierbarkeit fest (Tanenbaum and Steen, 2006, 9):

1. Skalierbarkeit hinsichtlich der Größe: Lassen sich mehr Ressourcen und/oder Nutzer einfach so zum System hinzufügen?

2. Geographische Skalierbarkeit: Funktioniert das System auch, wenn die Nutzer geographisch weit auseinander angeordnet sind?

3. Administrative Skalierbarkeit: Lässt sich das System über mehrere administrative Organisationseinheiten skalieren?

Tanenbaum and Steen (2006) stellen dabei zu diesen Dimensionen Folgendes fest: *„Unfortunately, a system that is scalable in one or more of these dimensions often exhibits a loss of performance as the system scales up.“* (Tanenbaum and Steen, 2006, 9)

Wenn Skalierbarkeit bei bestimmten DVEs betrachtet wird, so geht es meist um den ersten Punkt, Skalierbarkeit hinsichtlich der Größe. Gelegentlich ist auch die Entfernung der Teilnehmer Teil der Diskussion. Im Folgenden wird sich jedoch die Skalierbarkeit auf die Skalierbarkeit hinsichtlich der Größe beziehen.

2.2 Das Problem der Konsistenz

Bei DVEs stehen vor allem die Geschwindigkeitseinbußen bei der Hinzunahme von immer mehr Nutzern im Mittelpunkt der Skalierbarkeitsforschung. Gupta et al. (2009) kommen zu folgendem Schluss hinsichtlich der Probleme, welche durch Skalierung einer DVE entstehen: *„These scalability problems arise in part because of the need to maintain consistency between all players.“* (Gupta et al., 2009, 1) Zum Erhalten der Konsistenz ist es notwendig, dass aus den Aktionen der unterschiedlichen Klienten auf dem Systemzustand ein gemeinsamer Zustand entsteht. Dieser Vorgang wird state melding genannt. Liu et al. (2012) gehen soweit zu sagen,

dass *„[s]tate melding is the core of creating this illusion of a shared reality“* (Liu et al., 2012, 1).[1] State melding besteht aus zwei Hauptteilen:

1. consistency maintenance
2. state update dissemination

Ersteres befasst sich mit dem Zusammenführen der unterschiedlichen Änderungen des Systemzustands durch die teilnehmenden Prozesse bzw. Klienten. Letzteres befasst sich mit der Aufgabe, unter effizienten Einsatz von Ressourcen die Zustandsaktualisierungen an die teilnehmenden Prozesse zu verteilen. Der entstehende Verkehr stellt dabei nicht nur die Knoten der virtuellen Umgebung vor große Herausforderungen, auch die Bandbreite des Netzwerks wird immer stärker in Anspruch genommen.

Nach Liu et al. (2012) lassen sich folgende Eigenschaften der Konsistenz und Interaktivität von DVEs herausstellen:

- Kausalität
- Nebenläufigkeit
- Simultanität
- Fähigkeit zu Echtzeitreaktionszeiten (real-time responsiveness)

Kausalität und Nebenläufigkeit sind Kriterien, welche an viele verteilte Anwendungen angelegt werden. Kausalität ist dabei im Sinne der happened before Relation von Lamport zu verstehen. (Lamport, 1978, 558) Simulatanität und real-time responsiveness sind hingegen Kriterien, welche DVEs nicht mit allen verteilten Systemen teilen. Simultanität bezieht sich darauf, dass, wenn zwei Ereignisse von einem Nutzer simultan wahrgenommen werden, dies auch für die anderen Nutzer gelten muss. Das Kriterium der real-time responsiveness führt dazu, dass, um den Eindruck einer sofortigen Antwort des Systems aufrechtzuerhalten, in einigen DVEs kurzfristige Inkonsistenzen auf der Seite des Klienten in Kauf genommen werden. (Liu et al., 2012, 3)

2.2.1 Konsistenzmodelle

Um diese Eigenschaften ganz oder teilweise zu erreichen, verfolgen DVEs unterschiedliche Konsistenzmodelle. Tanenbaum and Steen (2006) fassen die Bedeutung eines Konsistenzmodells für ein verteiltes System wie folgt zusammen: *„A consistency model is essentially a contract*

[1] Die Autoren beschäftigen sich mit virtual worlds, verwenden den Begriff jedoch so, dass er synonym zu dem der DVE verstanden werden kann.

between processes and the data store" (Tanenbaum and Steen, 2006, 277). Das System garantiert, dass es korrekt funktioniert, so lange sich die Prozesse an die Regeln halten, welche im Vertrag festgehalten werden. Wichtig für die Wahl des Konsistenzmodells ist der Kontext der Anwendung. Je nach Art der Anwendung sind einige der oben vorgestellten Eigenschaften wichtiger als andere. (Bouillot and Gressier-Soudan, 2004, 21)

Liu et al. (2012) unterscheiden zwei verschiedene Typen von Konsistenzmodellen, welche in DVEs zum Einsatz kommen:

- ultimate consistency
- deadline-based consistency

Modelle des Typs ultimate consistency beschäftigen sich nicht mit einer globalen Zeit, stattdessen werden Regeln und Reihenfolgen für die Ausführung der einzelnen Operationen festgelegt. Es geht also um eine Art logische Zeit. Deadline-based consistency bedeutet, dass Nachrichten mit physikalischen Zeitstempeln versehen werden. Die Ausführungsreihenfolge der Operationen richtet sich demnach nach Echtzeitbedingungen. Dieses Modell bedingt, dass die Uhren der einzelnen Rechner zeitlich synchronisiert sind.

Die Modelle der ultimate consistency lassen sich nach Liu et al. (2012) wie folgt unterscheiden:

- Kausale Konsistenz: Alle kausal in Beziehung stehenden Ereignisse werden von allen teilnehmenden Prozessen in der derselben Reihenfolge gesehen.
- Sequentielle Konsistenz: *„The result of any execution is the same as if the (read and write) operations by all processes on the data store were executed in some sequential order and the operations of each individual process appear in this sequence in the order specified by its program."* (Tanenbaum and Steen, 2006, 282)
- Serialisierbarkeit: Serialisierbarkeit bedeutet, dass alle Transaktionen in derselben Reihenfolge von allen teilnehmenden Prozesse gesehen werden. Wenn Transaktionen als einzelne Schreib- oder Leseoperationen definiert sind, dann sind Serialisierbarkeit und sequentielle Konsistenz synonym zu verstehen.[2]

Es ist festzuhalten, dass nach Tanenbaum and Steen (2006) all diese Modelle als datenzentrierte Konsistenzmodelle zu verstehen sind. (Tanenbaum and Steen, 2006, 276ff) Während datenzentrierte Modelle Konsistenz mit dem Blick auf einen geteilten Datenspeicher betrachten, konzentrieren

[2] Tanenbaum and Steen (2006) betrachten das Thema der Serialisierbarkeit im Unterschied zu Liu et al. (2012) nicht als Problem der Konsistenz, sondern behandeln es als Teil des Themenbereichs Transaktionen. (Tanenbaum and Steen, 2006, 22)

sich klientenzentrierte Modelle auf die Perspektive des Klienten. (Tanenbaum and Steen, 2006, 273)

Zu den Modellen der ultimate consistency ist zudem anzumerken, dass keines sich mit einer Reduzierung der Antwortzeiten beschäftigt. Wenn sie in einer DVE implementiert werden, müssen zusätzliche Mechanismen eingesetzt werden, um kurze Antwortzeiten zu erreichen.

Die Modelle der deadline-based consistency lassen sich nach Liu et al. (2012) wie folgt unterscheiden:

- Wahrnehmende Konsistenz (auch: Absolute Konsistenz): Wenn ein Ereignis bei einem Prozess vor einem anderen Ereignis ausgeführt wird, dann muss dies auch bei allen anderen Prozessen der Fall sein.
- Verzögerte Konsistenz: Im Gegensatz zur absoluten Konsistenz wird eine Verzögerungszeit für die Ausführung von Ereignissen festgelegt und weder über- noch unterschritten.
- Zeitlich abgepasste Konsistenz: Die Ausführung einer Operation wird innerhalb einer bestimmten Verzögerungszeit auf einem anderen Prozess wahrgenommen.

2.2.2 Verteilung der Zustandsaktualisierungen

Neben der Definition, was als konsistent gilt, steht die Frage, wie der Zustand konsistent gehalten wird. Daraus ergibt sich die Aufgabe zu definieren, wie und wann die beteiligten Parteien über Änderungen des Zustands informiert werden. Zu häufige Aktualisierungen können zu einer Überlastung des Netzwerks führen oder die den Zustand verwaltenden Komponenten überfordern. Dabei ist vor allem zu hinterfragen, welcher Klient über welche Zustandsänderungen informiert werden muss. Denn obwohl alle Klienten nicht unterschiedliche, konfligierende Zustände der gleichen Welt sehen sollen, müssen sie gegebenenfalls nicht den gleichen Ausschnitt dieser Welt sehen. Je nach Position in der Welt können andere Zustandsaktualisierungen notwendig sein.

Gerade diese Erkenntnis ist relevant, um die Leistung einer DVE zu steigern bzw. sie überhaupt erst möglich zu machen. Im Folgenden werden mit Interessenmanagement und Detaillierungsgrad zwei unterschiedliche Ansätze vorgestellt, die Verteilung der Zustandsaktualisierungen zu organisieren.

Interessenmanagement

Unter Interessenmanagement werden Techniken verstanden, die für den jeweiligen Nutzer notwendigen Aktualisierungen zu identifizieren. Die vorhandenen Ansätze hierzu lassen sich in zwei Kategorien einteilen: Raumbasierte und klassenbasierte Ansätze. Raumbasierte Ansätze

bestimmen das Interesse an Zustandsaktualisierungen anhand der relativen Position der Objekte in der virtuellen Welt. Klassenbasierte Ansätze bestimmen das Interesse anhand von Objekt-Variablen (z.B. dem Typ eines Objekts). (Liu et al., 2012, 10)

Detaillierungsgrad

Bei den Techniken, die unter dem Oberbegriff Detaillierungsgrad zusammengefasst werden, werden Beschränkungen der menschlichen Wahrnehmungen ausgenutzt wie z.B. die Unfähigkeit, auf großer Distanz Details zu erkennen. So lassen sich bestimmte Aktualisierungen in Anzahl und Qualität reduzieren, wenn ein Objekt sich in größerer Entfernung befindet. (Liu et al., 2012, 13)

2.3 Fazit

In diesem Kapitel wurde festgehalten, dass DVEs spezielle verteilte Systeme sind, in denen die Simulation von Raum und Zeit eine besondere Rolle spielt. DVEs unterscheiden sich jedoch untereinander. Es kann sich bei Ihnen um Spiele unterschiedlicher Genres handeln oder gar um Simulationen ohne Spielaspekt. Je nach konkreter Anwendung unterscheiden sich auch die Anforderungen an die Leistungsfähigkeit. Alle DVEs teilen sich aber das Problem der Skalierbarkeit hinsichtlich der Größe, d.h. der Anzahl der beteiligten Nutzer. Diese Form der Skalierbarkeit ist für die restliche Arbeit als die entscheidende Form der Skalierbarkeit identifiziert worden.

Das Zusammenführen der Zustandsänderungen in einem konsistenten Weltzustand, das state melding, ist der entscheidende Teil einer DVE, in dem Skalierbarkeit und Leistung wichtig werden. Dauert der Prozess des state meldings zu lange, geht die Illusion der kohärenten Welt verloren. Es ist damit der Bereich, auf den sich das Messen der Leistung konzentrieren sollte.

State melding besteht aus den beiden Teilaspekten consistency maintenance und state dissemination. Die consistency maintenance orientiert sich an theoretischen Konsistenzmodellen. Je nach DVE kommen andere Konsistenzmodelle zum Einsatz. Für die weitere Bearbeitung der Aufgabenstellung dienen diese Modelle als Grundlage, um die consistency maintenance konkreter DVEs einzuordnen. Je nach Konsistenzmodell können bestimmte Annahmen über den Zustand auf Seite der Klienten und über den Zustand auf Seite des Servers getroffen werden.

State dissemination handhabt die Verbreitung der Zustandsaktualisierungen. Ziel ist eine Reduktion der Kommunikation der Klienten auf die Informationen, die für sie notwendig sind. Das jeweilige Verfahren ist beim Messen der Leistungsfähigkeit deshalb relevant, da es festlegt, wann Kommunikation erfolgt und wann nicht. Diese Information ist beim Generieren von vergleichbarer Last bzw. beim eventuellen Messen von Kommunikation entscheidend.

3 Kriterien zur Bewertung der Leistung von verteilten virtuellen Umgebungen

Wenn eine bestehende DVE skaliert werden soll, besteht der Anreiz, diese Skalierung mittels Hinzunahme weiterer, leistungsfähigerer Ressourcen zu realisieren. Dies ist zum einen nicht immer möglich, da nicht jede Anwendung durch mehr Ressourcen skaliert werden kann. Zum anderen kosten mehr und leistungsfähigere Ressourcen auch mehr Geld. Deshalb besteht der Bedarf, Algorithmen und Technologien einzusetzen, welche helfen, die Anforderungen wie Skalierbarkeit und Leistung zu erfüllen. Die Entwicklung dieser Algorithmen an sich ist schon keine triviale Aufgabe. Doch die Evaluation der Algorithmen wirft eine weitere Frage auf: Wie lässt sich die Leistung einer DVE bewerten? Es findet sich wenig Literatur, die sich explizit mit der Leistungsfähigkeit von verteilten virtuellen Umgebungen beschäftigt. Stattdessen steht die Skalierbarkeit häufig im Mittelpunkt. Hinter der Frage der Skalierbarkeit steht jedoch die allgemeinere der Leistung einer DVE und die Auswertung der Algorithmen zur Verbesserung der Skalierbarkeit läuft auf eine Auswertung der Leistung der DVE hinaus.

In diesem Kapitel sollen unterschiedliche Bewertungskriterien vorgestellt und anschließend evaluiert werden. Ziel ist es, ein geeignetes Kriterium zu identifizieren, welches dabei hilft, die Leistung von DVEs zu bewerten.

3.1 Systemspezifische Auswertungskriterien

In der Fachliteratur zu Skalierbarkeit und Leistung von virtuellen Umgebungen werden häufig konkrete Umgebungen untersucht. Dabei werden unter anderem Kriterien verwendet, die sich auf spezielle Eigenschaften der Leistung entweder hinsichtlich der Nutzererfahrung oder auf spezifische Aspekte der Software der Umgebung beziehen. Ich bezeichne diese Kriterien deswegen als systemspezifisch, da sie sich nicht ohne weiteres auf andere Systeme anwenden lassen. Dies kann daran liegen, dass z.B. bestimmte Kriterien eines FPS nicht ohne weiteres auf ein Rollenspiel übertragen werden können. Die Auswertung der Leistung bestimmter Aspekte eines implementierten Algorithmus bezieht sich z.B. auf Ereignisse im Spielgeschehen, die in anderen Systemen nicht vorkommen. Es ist wichtig, dabei festzuhalten, dass die als nicht systemspezifisch

kategorisierten Ansätze im weitesten Sinne als nicht systemspezifisch bezeichnet werden. Es kann immer DVEs geben, auf welche bestimmte Annahmen, die den verwendeten Leistungskriterien zugrundeliegen, nicht anwendbar sind.

3.1.1 Nutzererfahrung

Die zentrale Frage ist hier: Welchen Einfluss hat das Skalieren des Systems auf die durch den Nutzer wahrgenommene Leistung? Fritsch et al. (2005) beschäftigen sich mit den Auswirkungen der Latenz auf die Erfahrung der Nutzer eines MMORPG. Bei dieser Bewertung ist die Latenz die Stellgröße und die quantifizierten Nutzererfahrungen die Messwerte. In dem konkreten Anwendungsfall, dem MMORPG Everquest2, stellen sie fest, dass das Spiel mit bis zu 1250 ms Latenz noch spielbar ist. Kriterien für die Bewertung sind die Dauer der Kämpfe, die Ressourcen, die nach einem Kampf übrig sind, sowie Gesundheit und Mana nach einem Kampf. (Fritsch et al., 2005, 5) Diese Kriterien beziehen sich also auf Aspekte der Spiellogik einer DVE.

3.1.2 Softwarespezifisch

Es existieren unterschiedliche Möglichkeiten in der Architektur, wie auf höhere Last reagiert werden kann oder wie von vornherein eine zu hohe Last verhindert werden kann. Die Technik des zonings bezieht sich auf eine Einteilung der virtuellen Welt in Partitionen, die klein genug sind, dass ein Server sie verwalten kann. Beim sogenannten sharding werden völlig getrennte Instanzen der gleichen Welt, shards, erstellt. Dies ermöglicht z.B., hohe Latenzen zu vermeiden, indem Spieler aus geographisch weit entfernten Regionen der Erde nicht in derselben Instanz spielen. Eine weitere Technik wird als instancing bezeichnet. Hier werden kleine Teile der virtuellen Welt einer bestimmten Anzahl Spieler vorbehalten, um gemeinsam und ungestört spielen zu können. Dabei wird diese Technik meist nicht aufgrund ihrer Effekte auf die Skalierbarkeit eingesetzt, sondern aus Gründen des Spieledesigns. (Gupta et al., 2009, 1312)

Barri et al. (2010) verwenden Bewertungskriterien für eine DVE mit zoning. Sie messen die Anzahl der erstellten Zonen, die Lebensdauer einer Zone sowie die Anzahl der Spieler pro Zone. (Barri et al., 2010, 346f) Als verstellbare Größen wählen sie sowohl die Anzahl der Spieler als auch die Wartezeit für einen neuen Spieler, um einem Spiel beizutreten. Jeweils eine der Größen wird dabei fix gesetzt und die andere verändert. Das automatische Erstellen von Zonen ist Teil des von ihnen untersuchten Algorithmus und dient zum Umgang mit verstärkter Nutzung der Umgebung (in diesem Fall ein FPS). Damit lässt sich dieses Kriterium allerdings nur auf Umgebungen verallgemeinern, die ähnlich bei steigender Last reagieren.

3.2 Nicht systemspezifische Auswertungskriterien

3.2.1 Ressourcenbezogen

Natürlich gibt es auch Kriterien, welche sich leicht von einer DVE auf eine andere übertragen lassen. Offensichtlich ist dies möglich, wenn die Kriterien sich mit der Verwendung von Ressourcen beschäftigen.

Ein solches ressourcenbezogenes Kriterium kommt bei Morillo et al. (2006) zum Einsatz. Es handelt sich um die durchschnittliche CPU-Auslastung in Prozent. Dieses Kriterium ist relativ einfach zu erheben. Das Problem ist seine beschränkte Aussagekraft. Denn auch unter hoher Last können Systeme durchaus performant arbeiten. Eine hohe CPU-Auslastung von bis 100 Prozent kann zwar darauf hindeuten, dass das System nicht mehr performant arbeitet, aber auch ohne hohe CPU-Auslastung kann das System langsam antworten. Dies kann z.B. an einer schlechten Implementierung der Verarbeitung von Nachrichten liegen.

Der Verbrauch der Bandbreite ist ein ähnlich gelagertes Kriterium. Es wird u.a. bei Ahmad et al. (2008) und Najaran and Krasic (2010) als Kriterium angegeben. Erstere betrachten den Bandbreitenverbrauch in kb/s hinsichtlich der Anzahl der verwendeten Peers (sie untersuchen eine Peer-to-Peer-DVE). Letztere erwähnen zwar das Kriterium der Bandbreite, verzichten dann aber darauf, Werte für dieses Kriterium zu erheben. Gupta et al. (2009) betrachten die Menge der übertragenen Daten in Abhängigkeit von der Anzahl der Klienten. Die Einwände gegen die Verwendung dieser Kriterien können ähnlich erhoben werden, wie bei dem Kriterium der CPU-Auslastung. Ein hoher Verbrauch an Bandbreite ist ein Indiz für eine schlechte Leistung der DVE und besonders beim Einsatz im Internet eine relevante Größe. Dieser Wert muss aber nicht zwangsläufig mit einer schlechten Leistung einhergehen.

3.2.2 Nachrichtenbezogen

Von dem Verbrauch der Bandbreite ist es nicht weit zu Kriterien, welche sich mit den zwischen den einzelnen Computern einer DVE ausgetauschten Nachrichten beschäftigen. Ahmad et al. (2008) betrachten neben der Bandbreite als nachrichtenbezogenes Kriterium die durchschnittliche Anzahl der ausgetauschten Nachrichten in Bezug auf die Anzahl der verwendeten Peers. Morillo et al. (2006) vergleichen zwei Algorithmen hinsichtlich der Anzahl der Aktualisierungsnachrichten, die bei unterschiedlicher Skalierung verschickt werden. (Morillo et al., 2006, 5)

Bei diesem Kriterium muss berücksichtigt werden, dass die Aktualisierungsnachrichten nicht immer von gleicher Größe sind und eine bestimmte Anzahl an Aktualisierungen notwendig sind, um einen konsistenten Spielzustand zu erhalten. Je nach DVE kann aber auch eine hohe oder eine geringe Anzahl von verschickten Zustandsaktualisierungen sowohl ein Indiz für gute als auch

für schlechte Leistung sein. Ein Server könnte sowohl zu viele Nachrichten verschicken, was das Netzwerk negativ beeinflusst, als auch zu wenige Nachrichten verschicken, da er überlastet ist und nicht mehr in der Lage ist, genug Nachrichten an die Klienten zu verschicken, um die Illusion einer gemeinsamen Welt aufrechtzuerhalten.

Bharambe et al. (2008) verwenden zur Evaluation der Leistung ihrer Architektur einen anderen Ansatz. Sie messen den Anteil der notwendigen Aktualisierungsnachrichten, welche der Spieler rechtzeitig erhalten hat. Notwendig heißt bei ihnen, dass die Nachrichten aufgrund der Spielelogik für den Spieler relevant sind.[1] Rechtzeitig heißt, dass eine bestimmte Deadline für die Dauer von einem Spieler zum anderen nicht überschritten wird. Für den konkreten Anwendungsfall, ein FPS-Spiel, setzen sie diesen Wert auf 150 ms fest. (Bharambe et al., 2008, 8)

Ähnlich zu diesem Ansatz beschäftigt sich ein Großteil der Artikel zum Thema Skalierbarkeit mit der Zeit, in der die Anfrage verarbeitet und an andere Klienten weitergeleitet wird. Najaran and Krasic (2010) messen die Verarbeitungszeit (processing time) von Aktualiserungsnachrichten, welche durch die Differenz zwischen dem Zeitpunkt der Ankunft und des Verlassens des verwaltenden Systems bestimmt wird. (Najaran and Krasic, 2010, 4) Als konkrete Werte werden dann die 99,9. Perzentile der schlechtesten Verarbeitungszeiten aller Spieler sowie die mittleren Verarbeitungszeiten der Aktualisierungen verwendet und in Abhängigkeit von der Anzahl der teilnehmenden Spieler dargestellt. (Najaran and Krasic, 2010, 5) Später nennen die Autoren diese Verarbeitungszeit in der konkreten, von ihnen vorgenommenen Auswertung Latenz.

Dieses Beispiel weist auf ein Problem hin, was in vielen Artikeln zum Thema Skalierbarkeit auftaucht. Es wird der Begriff der Latenz verwendet, aber nicht genau festgehalten, wie er verstanden wird. Welche Ereignisse die Messung der Latenz eingrenzen, wird meist nicht erwähnt.

Miller definiert die Latenz in einer DVE wie folgt: *„The lag between proposing a state change and that state change being communicated to or accepted by other elements of the DVE is called latency.“* (Miller, 2011, 15) Selbst diese Definition lässt als Grenze der Latenz die Auswahl zwischen dem Kommunizieren der Änderung und dem Akzeptieren der Änderung offen. Bei der Erfassung der Latenz muss zudem in Betracht gezogen werden, dass die Übertragung einer Nachricht im Netzwerk von der Netzwerkinfrastruktur abhängt. Dieser Effekt ist vor allem dann nicht zu vernachlässigen, wenn es um verteilte virtuelle Umgebungen im Internet geht. Der Umgang mit diesem Effekt ist somit durchaus relevant in der Praxis. Je nach Ziel der Auswertung ist dieser Effekt jedoch zu kontrollieren, um die Ergebnisse vergleichbar zu halten.

Barri et al. (2010) messen die Latenz eines von ihnen ausgewählten Systems. Wie genau diese Latenz definiert ist, bleibt aber unklar. Bei ihren Versuchen verwenden sie die Anzahl der Spieler

[1] Dabei ist festzuhalten, dass dieses Kriterium zwar an der Logik eines konkreten Systems ansetzt, das Verfahren jedoch im Prinzip auf beliebige DVEs übertragbar ist und somit nicht systemspezifisch ist. Je nach DVE wären bestimmte Aktualisierungen als notwendig zu definieren.

sowie die Wartezeit eines Spielers, um einem Spiel beizutreten, als Stellgrößen. Sie werten dabei die durchschnittliche Latenz hinsichtlich zweier Bestandteile des von ihnen untersuchten System aus: einem master server sowie dynamisch erstellten Zonen. Als Grenzwert für die akzeptable Latenz geben sie 95 ms an. (Barri et al., 2010, 342)

Ahmed and Shirmohammadi (2011) definieren die zu messende Latenz konkreter: Sie beschäftigen sich mit der paarweisen Latenz zwischen zwei Klienten. D.h. sie betrachten die Zeit, die die Zustandsänderung eines Klienten braucht, um bei einem anderen Klienten anzukommen. Die maximale Latenz aller Spielerpaare wird hinsichtlich einer sich ändernden Gruppengröße gemessen. Die Gruppe bezieht sich auf die Spieler, die voneinander Aktualisierungen erhalten. Darüber hinaus werten sie bei dem Vergleich zweier Algorithmen in Abhängigkeit der Gruppengröße aus, für wie viele Spieler es zu einer Verbesserung der Latenzen durch den neuen Algorithmus kommt. (Ahmed and Shirmohammadi, 2011, 9)

Gupta et al. (2009) hingegen verwenden in ihrer Testumgebung eine voreingestellte durchschnittliche Latenz von 238 ms. (Gupta et al., 2009, 9) Sie messen dann die durch die Klienten beobachtete Antwortzeit des Systems in Abhängigkeit von der Anzahl der Klienten, der Anzahl der sichtbaren Avatare als auch der Komplexität der durchgeführten Aktionen. Die Komplexität wird dabei anhand der Dauer der Bearbeitung einer Aktion errechnet.

Morillo et al. (2006) messen eine average system response time – also eine durchschnittliche Antwortzeit des Systems – in Abhängigkeit von der Anzahl der Spieler. Diese verstehen sie als durchschnittliche Zeit vom Senden einer Nachricht eines Avatars an den Server über dessen Weiterleiten der Nachricht an alle notwendigen Avatare bis zum Erhalt einer ACK-Nachricht von den anderen Avataren, welche diese nach dem Erhalt der ursprünglichen Nachricht an den sendenden Avatar schicken. Ein durchaus interessanter Aspekt an ihrer Untersuchung ist, dass sie die Leistung in Abhängigkeit von bestimmten Avatarverteilungen (gleichmäßig / uniform, ungleichmäßig / skewed, gebündelt / clustered, siehe auch Abbildung 3.1) messen. (Morillo et al., 2006, 4) Dies ist durchaus ein relevanter Punkt. Denn wenn die Avatare in der Spielewelt so verteilt sind, dass sie aufgrund von Interessenmanagement keine Aktualiserungsnachrichten voneinander erhalten müssen, wirkt das System durchaus leistungsfähiger, als wenn eine große Anzahl von Avataren nebeneinander in der Welt vorkommt und alle voneinander die Aktualisierungen erhalten müssen.

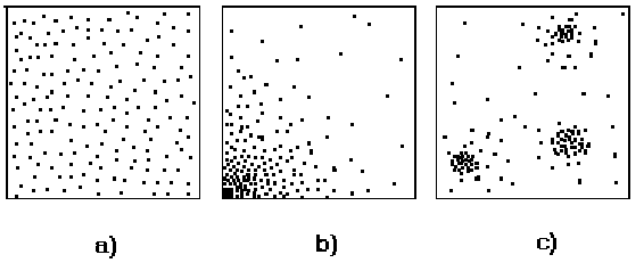

Abbildung 3.1: Unterschiedliche Verteilung von Klienten im zweidimensionalen Raum: a) gleichmäßig, b) ungleichmäßig, c) gebündelt (Morillo et al., 2006, 18)

3.3 Bewertung und Ausblick

Die vorgestellten Bewertungskriterien sind zur Übersicht in der Tabelle 3.1 festgehalten. Für die Bearbeitung der Aufgabenstellung ist aus diesen Kriterien ein geeignetes auszuwählen. Dieses soll dann in der Entwicklung eines Modells zum Messen der Leistung verwendet werden.

Das Kriterium soll dabei helfen, die Leistung von verteilten virtuellen Umgebungen möglichst generell zu betrachten. Es ist trotzdem festzuhalten, dass die systemspezifischen Kriterien eine Bedeutung haben, die über die Systeme, für die sie geschaffen wurden, hinaus geht. Durch die Beschäftigung mit diesen Kriterien wird deutlich, wie durch die Eingrenzung auf bestimmte Systeme spezielle Leistungskriterien entwickelt werden können, aber auch was die Besonderheiten bestimmter DVEs sind.

Um das Thema der Leistung von DVEs möglichst verallgemeinerbar zu erfassen, taugen sie jedoch nicht. Auch scheinen die ressourcenbezogenen Kriterien wenig geeignet, da sie die Leistung nur oberflächlich erfassen. Ihre begrenzte Aussagekraft wurde oben bereits erwähnt.

Vielmehr scheinen die Verarbeitungs- und Antwortzeiten, die in einigen nachrichtenbezogenen Leistungskriterien herangezogen werden, besser geeignet, da sie einen elementaren Aspekt von DVEs erfassen: Das Aufrechterhalten einer konsistenten Welt, so dass die verteilte, technische Implementierung dem Benutzer verborgen bleibt, die Verteilungstransparenz.

Wichtig erscheint dabei auch der von Ahmed and Shirmohammadi (2011) berücksichtigte Aspekt, dass vor allem die Dauer, die eine Änderung am Weltzustand von einem Nutzer zum anderen benötigt, relevant für die Illusion einer gemeinsamen Welt ist. Dabei wird Leistung nicht nur auf Seite des Servers betrachtet wie bei der processing time von Najaran and Krasic (2010). Vielmehr wird die potentielle Nutzerperspektive samt des Aspekts der Übertragung der Zustandsaktualisierungen im Netzwerk mit berücksichtigt. Im Folgenden soll deshalb das Kriterium einer paarweisen Latenz verstanden als die Dauer einer Aktualisierung von einem

– den Zustand ändernden – Klienten zu einem weiteren – die Änderung wahrnehmenden – Klienten betrachtet werden.

Ein weiterer Aspekt ist die Frage, welche Latenzen als tolerierbar gelten können. Ahmed and Shirmohammadi (2011) fassen zusammen, dass je nach Umgebung die Latenzen zwischen 100 und über 1000 ms schwanken können. So wird für das RPG Everquest 2 eine Latenz von 1250 ms als noch spielbar angegeben, wohingegen für das RPG Diablo 2 120 ms als Grenzwert gilt und 80 ms als optimaler Latenzwert. Dieser Aspekt scheint relevant für die Betrachtung einer einzelnen DVE. Für jede einzelne müsste ein solcher Wert festgelegt werden, damit gemessene Werte an Aussagekraft gewinnen. Dann könnte eine Auswertung auch um den Aspekt von rechtzeitig erhaltenen Nachrichten wie bei Bharambe et al. (2008) erweitert werden. Da in dieser Arbeit aber ein allgemeiner, auf mehrere DVEs übertragbarer Ansatz gefragt ist, wird die Bewertung der Zeiten als rechtzeitig ausgelassen.

Kategorie	Kriterium	Quelle
Systemspezifisch		
Nutzererfahrung	Auswirkung der Latenz auf Nutzererfahrung	Fritsch et al. (2005)
Softwarespezifisch	Anzahl erstellter Zonen	Barri et al. (2010)
	Lebensdauer der Zonen	Barri et al. (2010)
	Spieler pro Zone	Barri et al. (2010)
Nicht systemspezifisch		
Ressourcenbezogen	CPU-Auslastung	Morillo et al. (2006)
	Menge der übertragenen Daten	Gupta et al. (2009)
	Verbrauch der Bandbreite	Ahmad et al. (2008), Najaran and Krasic (2010)
Nachrichtenbezogen	Antwortzeit des Systems (konstante Netzwerklatenz)	Gupta et al. (2009)
	Antwortzeit des Systems (mit ACK-Nachrichten)	Morillo et al. (2006)
	Anzahl der Nachrichten (Aktualisierungen)	Morillo et al. (2006)
	Anzahl der Nachrichten (allgemein)	Ahmad et al. (2008)
	Latenz (ohne Definition)	Barri et al. (2010)
	Paarweise Latenz	Ahmed and Shirmohammadi (2011)
	Rechtzeitig erhaltene notwendige Aktualisierungen	Bharambe et al. (2008)
	Verarbeitungszeit	Najaran and Krasic (2010)

Tabelle 3.1: Übersicht über die Kriterien zur Bewertung der Leistung von DVEs

4 Modell zum Messen der Leistung

4.1 Ziele

In dem vorigen Kapitel wurde die paarweise Latenz als geeignetes Bewertungskriterium für die Leistung von DVEs identifiziert. Die paarweise Latenz ist dadurch bestimmt, wie lange es dauert, bis die Zustandsänderung eines Klienten bei einem anderen Klienten bekannt gemacht wird. Dies bedeutet, dass der Zeitpunkt der Zustandsänderung eines Klienten sowie der Zeitpunkt, wenn diese Änderung bei einem weiteren Klienten eintrifft, erfasst werden müssen. In DVEs werden diese Änderungen mit Nachrichten bekannt gemacht. D.h. die vom einen Klienten ausgehende Nachricht sowie die bei einem zweiten Klienten eintreffende Nachricht müssen in Verbindung gebracht werden – die ausgehende Zustandsänderung und die eingehende Zustandsaktualisierung müssen einander zugeordnet werden. Diese Zuordnung von Nachrichten ermöglicht somit erst das Messen der paarweisen Latenz.

Im Folgenden soll exemplarisch, d.h. anhand einer DVE, ein Modell zur Zuordnung und damit zum Messen entworfen werden. Um dieses Modell entwickeln zu können, wird zuerst die zu testende Software und das in ihr verwendete Protokoll untersucht. Hierbei soll deutlich werden, welche Nachrichten zur Zuordnung geeignet sind und wie diese Zuordnung erfolgen kann. Abschließend werden die Überlegungen zum Modell in zwei Algorithmen festgehalten. Diese sollen das automatisierte Messen der paarweisen Latenzen durch die Zuordnung von Nachrichten ermöglichen.

4.2 Vorgehen

Wie können für eine konkrete DVE die entsprechenden Daten gemessen werden, auf deren Grundlage die paarweise Latenz ausgewertet werden kann? Im Folgenden will ich analog zum Testen von Software drei unterschiedliche Betrachtungsweisen eines zu untersuchenden Systems unterscheiden, um die Optionen für das weitere Vorgehen beim Messen zu verdeutlichen.

Beim Testen von Software wird ein System als black box verstanden, wenn die Interna des Systems unbekannt sind und nur dessen Ausgabe bei einer bestimmten Eingabe betrachtet werden soll. Die Betrachtung als grey box ergänzt dieses Vorgehen, um limitiertes Wissen bezüglich der

Interna. Bei dem Testen eines Systems als white box werden die Interna selbst getestet. (Jalote, 2005, 535)

Übertragen auf das Messen der Leistung einer DVE lässt sich dies wie folgt verstehen. Als black box sind keine Interna des Systems bekannt, z.B. über die Art der Nachrichten oder die verwendete Konsistenz, und trotzdem ist die Leistung zu messen. Dies lässt sich nur schwer umsetzen und schon simple DVEs haben komplizierte Mechanismen der Verteilung der Nachrichten. Ansätze wie das Messen der CPU-Auslastung, wie es Morillo et al. (2006) vorschlagen, oder der Netzwerklast würden einem solchen Vorgehen entsprechen.

Als grey box wären Interna begrenzt bekannt, aber trotzdem werden nur äußere Charakteristika wie z.B. Nachrichten gemessen. Eingriffe in die DVE unterbleiben. Hingegen wäre dann bei einem white box Ansatz eine Untersuchung der Interna der einzelnen Komponente möglich. Dies könnte weitergehend zu einem Eingriff in die Software führen, um die Interna besser messen zu können. Z.B. wäre es möglich, bestimmte Ereignisse im Klienten messbar zu machen, indem der Code der DVE um entsprechende Messfunktionalitäten ergänzt wird.

Der black box Ansatz scheint eher beschränkt in seiner Aussagekraft. Der grey box Ansatz bezieht sich hauptsächlich auf externe Messgegenstände, während der white box Ansatz einen Eingriff in die DVE ermöglicht, vielleicht sogar notwendig macht. Offensichtlich ist, dass sich beim grey box Ansatz die programmatischen Anteile in einer externen Software kapseln lassen und so bei einer Veränderung der Software, diese nicht wieder komplett überarbeitet werden muss. Der white box Ansatz bietet mehr Flexibilität hinsichtlich dessen, was gemessen werden kann, schafft aber durch Eingriffe in die Software Abhängigkeiten und erhöht damit die Komplexität. Außerdem scheint die Übertragbarkeit auf andere DVEs eher bei einem grey box Ansatz gegeben zu sein. Sollten z.B. Nachrichten als Gegenstand des Messens dienen, müsste nur die Dekodierung und Verarbeitung der Nachrichten je nach DVE angepasst werden. Die grundsätzliche Infrastruktur könnte jedoch erhalten bleiben.

Im Folgenden wird der Ansatz einer grey box verwendet, um das Modell im Rahmen der Möglichkeiten auf andere DVEs übertragbar zu halten. Es wird versucht, wenig in die zu testende Software einzugreifen und das verwendete Protokoll nicht anzupassen.

4.3 Die zu testende Software: Timadorus

Die zu testende DVE, Timadorus, wird im Rahmen eines Projekts an der HAW Hamburg entwickelt. Es handelt sich um die Implementation eines MMORPG in Java. Die Kommunikation zwischen Klienten und Server erfolgt dabei per TCP. Bisher ist die Entwicklung soweit voran-

geschritten, dass als Klienten Bots zur Verfügung stehten, die sich durch eine virtuelle Welt bewegen. Ein von Menschen nutzbarer Klient existiert nicht.

Timadorus basiert auf Red Dwarf Server (RDS), einem Fork des Project Darkstar von Sun. Für den Entwurf eines Modells ist dabei relevant, wie in dieser Software das state melding organisiert wird. Denn das state melding ist der entscheidende Aspekt für die Leistung einer DVE (siehe Abschnitt 2.2).

4.3.1 Red Dwarf Server und Project Darkstar

RDS bietet eine Server-Infrastruktur, mit Hilfe derer unterschiedliche Spiele entwickelt werden können. Ziel von Darkstar und RDS ist es, die Illusion einer Umgebung mit einem einzelnen Server-Thread auf einer einzelnen Maschine aufrechtzuerhalten, während im Hintergrund das System mit mehreren Threads und Maschinen arbeitet, um so Skalierbarkeit zu gewährleisten. (Waldo, 2008, 14)[1] RDS funktioniert dabei als Container, in dem der eigentliche Spiele-Server läuft.

Bei RDS wird der Empfang der Eingabe des Klienten vom Server als Ereignis verstanden, was einen Thread, eine sog. Aufgabe, zur Folge hat. Diese Aufgaben können den Zustand der Welt ändern und initiieren Kommunikation zu den Klienten. (Waldo, 2008, 14) Der Zustand der Welt wird mittels eines sog. data services in einem data store persistiert. Letzterer ist durch ein Berkley Database System implementiert. (Liu et al., 2012, 18) Dabei ist jede Aufgabe in eine Transaktion eingehüllt, um konfligierende Zugriffe auf den Zustand zu vermeiden.

Die zugrundeliegende Architektur ist in Abbildung 4.1 dargestellt. Der Kommunikationsdienst ist die Brücke zwischen den Klienten und den Spieleservern. Die Aufgaben, welche auf diesen laufen, persistieren die Änderungen über den data service in einem oder mehreren Datenservern, dem data store. In diesem data store wird der Weltzustand festgehalten.

RDS kennt zwei Ebenen der Konsistenzkontrolle. Zum einen muss der Zustand der Klienten mit dem der autoritativen Kopie des Servers im Einklang gehalten werden. Dabei setzt der Server eine Reihenfolge für die Eingaben der Klienten fest. Unzulässige Zustände werden anschließend auf Klientenseite überschrieben. Durch dieses Vorgehen stellt der Server sequentielle Konsistenz her. (Liu et al., 2012, 18)

Zum anderen muss der Zustand zwischen den Servern synchronisiert werden. Eigentlich verwenden die Aufgaben der unterschiedlichen Server denselben data service. Jedoch werden Kopien der Objekte im Cache gehalten, um Verzögerungen zu verhindern. Durch die Transaktionen, in welche Aufgaben gehüllt werden, sollen die dadurch entstehenden Konflikte reduziert werden

[1] Die Anmerkungen von Waldo (2008), einem mehrjährigen Entwickler von Project Darkstar, beziehen sich zwar nur auf Project Darkstar sind aber so grundsätzlich gehalten, dass sie auf RDS übertragbar sind.

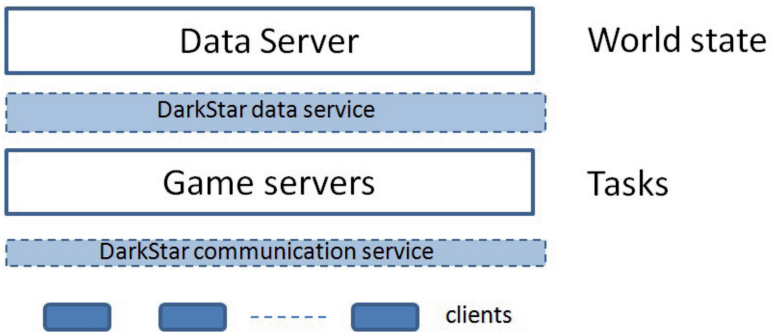

Abbildung 4.1: Server-Architektur von Project Darkstar (Liu et al., 2012, 18)

und für konfligierende Aufgaben ein neuer Termin zur Ausführung festgelegt werden. Liu et al. (2012) halten fest, dass damit Serialisierbarkeit der Operationen über mehrere Spielserver erreicht wird. (Liu et al., 2012, 18)

Momentan legt RDS kein Protokoll zur Verteilung der Zustandsaktualisierungen fest. Dies liegt in der Hand der Entwickler der tatsächlichen Anwendungen, welche mit RDS implementiert werden sollen.

4.3.2 Timadorus[2]

Bei Timadorus werden die Aktualisierungen nach dem Radius gefiltert. Wenn sich die Zustandsänderung innerhalb eines bestimmten Radius um die Position eines Klienten der Spielewelt zuträgt, werden Aktualisierungen an den Klienten weitergeleitet. Dies entspricht dem in Kapitel 2.2.2 vorgestellten Interessenmanagement.

Als klientenseitiger Bot existiert momentan einzig die Implementierung eines Rehs. Dieses läuft durch die virtuelle Welt und isst Kohlköpfe. Dabei folgt es dem SARSA(Lambda) Algorithmus für die künstliche Intelligenz. Die Aktionen des Rehs sind beschränkt auf Bewegung sowie Konsumieren von Objekten.

4.3.3 Fazit

Die zwei Aspekte des state meldings lassen sich auch in Timadorus bzw. dem zugrundeliegenden RDS wiederfinden. Für die consistency maintenance wird auf Serverseite die sequentielle Konsistenz verwendet. Die Klientenseite hat sich dieser Konsistenz unterzuordnen. Temporär

[2] Zu Timadorus wurde eine Bachelorarbeit geschrieben (Willatowski (2013)), welche jedoch nicht öffentlich einsehbar ist und bei der Bearbeitung der Aufgabenstellung nicht zur Verfügung stand. Darüber hinaus wurden bisher keine wissenschaftlichen Artikel zu Timadorus publiziert. Allerdings liegt der vollständige Quellcode vor, der bei auftauchenden Fragen zur Implementierung konsultiert werden kann.

können auf Seite einzelner Klienten inkonsistente Zustände entstehen, um anschließend vom Server überschrieben zu werden.

Die state dissemination, also die Verteilung der Zustandsaktualisierungen, ist in RDS nicht festgelegt und damit Timadorus überlassen. Dieses verwendet Interessenmanagement anhand der Distanz zwischen Objekten. Darüber hinaus ist festzuhalten, dass die Aktualisierungen nicht direkt weitergeleitet werden. Stattdessen wird bei einer Änderung des Weltzustands auf der Seite des Servers eine Benachrichtigungsnachricht über diese Änderung an alle interessierten Klienten verschickt.

4.4 Analyse des Protokolls

Um zu verstehen, wie das Ziel, die paarweise Latenz zu messen, umgesetzt werden kann, muss das Protokoll analysiert werden. Es muss überprüft werden, wie die vom Klienten verschickte Zustandsänderung bei einem anderen Klienten ankommt. In dem vorigen Abschnitt wurde deutlich, dass die Zustandsaktualisierungen nicht direkt weitergeleitet werden. Daraus ergibt sich als zentrale Frage für die Analyse des Protokolls, wie die Zuordnung der unterschiedlichen Nachrichten als zusammengehörig erfolgen kann.

Als erster Schritt der Analyse wurde der Verkehr zwischen Klienten und Server mitgeschnitten und aufgezeichnet. Da es sich dabei erst einmal nur um TCP-Pakete mit kodierten Daten handelt, wurde der Code zum Dekodieren der Nachrichten aus Timadorus übernommen.

Das Protokoll unterscheidet zwischen Server-Nachrichten und Klienten-Nachrichten.[3] Server-Nachrichten sind Nachrichten, welche die Klienten an den Server schicken. Klienten-Nachrichten sind hingegen Nachrichten an die Klienten.

4.4.1 Zuordnung von Zustandsaktualisierungen

Bei der Analyse des Protokolls ist ein hauptsächliches Ziel, Zustandsänderungen eines Klienten und die Nachrichten zu identifizieren, welche die Zustandsaktualisierungen bei einem anderen Klienten bekannt machen. Der Lastgenerator vollzieht nur zwei Tätigkeiten, welche Einfluss auf die Spielewelt haben. Das Reh bewegt sich und isst Kohlköpfe. Die folgende Analyse beschränkt sich auf die Bewegung des Klienten, um die Komplexität des Ansatzes zu reduzieren.

Auffällig bei der Analyse des Protokolls ist, dass die Nachrichten keine eindeutige ID, keinen Zeitstempel o.ä. besitzen. Ihr Aufbau und Inhalt sind rein an dem Transport des spielerelevanten Inhalts orientiert. Die Nachricht eines Klienten, dass er sich jetzt in eine bestimmte Richtung bewegen möchte, und die Bekanntgabe dieser Änderung bei einem anderen Klienten können nur

[3] Eine Übersicht über alle im Protokoll definierten Nachrichten kann aus Tabelle 1 im Anhang entnommen werden.

über den Inhalt, d.h. anhand der Daten zur Bewegung, als zusammengehörig identifiziert werden. Dies liegt vor allem auch an der oben diskutierten Architektur von RDS. Zustandsänderungen werden nicht einfach an andere Klienten weitergeleitet. Sie werden in einer Datenbank, dem data store, gespeichert und diese Änderung der Daten hat ein Benachrichtigung aller interessierten Klienten zur Folge.

Aus dem Ablauf der initialen Klienten-Server-Kommunikation kann man entnehmen, dass bei Bewegungen der Klient eine MOVE-Anfrage an den Server schickt, welche die Bewegung und die Rotation als Werte enthält (siehe Abbildung 4.2).[4] Eine entsprechende MOVEOBJECT-Aktualisierung geht vom Server aus an die anderen Klienten, welche im entsprechenden Radius stehen.

Diese MOVEOBJECT-Nachricht enthält die ID des Klienten, continuous (gibt an, ob die Bewegung kontinuierlich ist oder spontan), die Position, die Richtungsänderung/Bewegung, die Orientierung und die Rotation des Klienten (siehe Abbildung 4.3). Somit ist es möglich anhand der Kombination aus Bewegung und Rotation des Klienten, MOVEOBJECT-Nachrichten zu identifizieren, welche aus einer MOVE-Anfrage mit gleicher Kombination aus Bewegung und Rotation resultieren.[5]

Aus der Analyse der Inhalte beider Nachrichtentypen ergibt sich ebenfalls, dass die MOVE-Nachrichten nichts über die konkreten Koordinaten des sich bewegenden Klienten aussagen. Vielmehr werden diese auf Seiten des Servers berechnet und in den MOVEOBJECT-Nachrichten verschickt. In einem gewissen zeitlichen Abstand kann dann eine weitere Nachricht mit neuen Koordinaten erfolgen. Dies bedeutet, dass sich aus einer MOVE-Nachricht mehrere MOVEOBJECT-Nachrichten ergeben können, die sich hinsichtlich der berechneten Koordinaten unterscheiden.

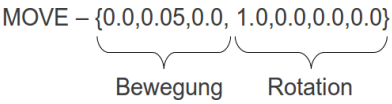

Abbildung 4.2: Aufbau einer MOVE-Nachricht

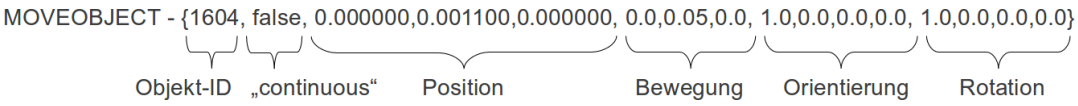

Abbildung 4.3: Aufbau einer MOVEOBJECT-Nachricht

[4] In Tabelle 2 des Anhangs ist das Beispiel einer initialen Klienten-Server-Kommunikation dargestellt.

[5] Im Beispiel der Tabelle 2 im Anhang sendet der Klient, dessen ID 1602 ist, mit der Nachricht 31 eine MOVE-Anfrage an den Server. Eine entsprechende Benachrichtigung über die veränderten Bewegungsdaten und die aktuelle Position des Klienten erfolgt in Nachricht 37.

4.4.2 Identifizieren der ID eines Klienten

Ein weiteres Problem besteht darin, die ID des Klienten zu ermitteln, denn diese wird nicht mit der MOVE-Anfrage mitgesendet. Somit besteht als zweites Ziel der Analyse des Protokolls, die ID des Klienten herauszufinden, der über eine TCP-Verbindung eine MOVE-Anfrage verschickt hat.

Hierfür eignen sich die ENTERING-Nachrichten. Diese Nachrichten haben den Zweck, dem Klienten die eigene ID mitzuteilen. Für das Messen ist diese Nachricht durchaus relevant: Hiermit kann einer Kombination aus TCP-Port und IP-Adresse eine Klienten-ID zugeordnet werden und damit sämtliche Kommunikation über den Port an einem bestimmten Rechner dieser ID zugeordnet werden.[6]

4.4.3 Fazit

Es lässt sich zusammenfassen, dass anhand des Mitschneidens der Kommunikation zwischen Klienten und Server Nachrichten eindeutig Klienten zugeordnet werden können. Dies geschieht mit den ENTERING-Nachrichten.

Aufgrund des bloß inhaltlichen Charakters der untersuchten Bewegungsnachrichten (MOVE und MOVEOBJECT) lassen sich die Zustandsänderung eines Klienten und die darauf folgende Zustandsaktualisierungen an die anderen Klienten durch den Server nicht exakt zuordnen. Die einzige Zuordnung, welche erfolgen kann, ist mit Hilfe des Inhalts und entspricht einer vermeintlichen Zugehörigkeit. Damit diese Zuordnung automatisch erfolgen kann und damit das Messen der paarweisen Latenzen auf Basis dieser Zuordnung möglich wird, muss ein Algorithmus entworfen werden, der diese Zuordnung von Zustandsaktualisierungen vornimmt.

4.5 Zuordnung von Nachrichten

4.5.1 Überlegungen zur Konstruktion von Algorithmen

Das generelle Problem lässt sich auf die Zuordnung von MOVE- zu MOVEOBJECT-Nachrichten reduzieren. Da eine exakte Zuordnung nicht möglich ist, können unterschiedlich aufwändige Versuche zum Entwurf von Algorithmen unternommen werden, um diese Zuordnung möglichst präzise durchzuführen.

Bei der Konstruktion eines Algorithmus ist jedoch die generelle Aufgabenstellung der Arbeit zu beachten. Ziel ist es, das Messen der Leistung zu untersuchen und nicht möglichst exakte

[6] In Tabelle 2 des Anhangs ist zum Beispiel zu sehen, dass der Server dem Klienten in Nachricht 3 mitteilt, dass er die ID 1602 hat.

Daten über die zu untersuchende DVE zu erheben. Deswegen scheint der Entwurf eines Greedy-Algorithmus angebracht.

Greedy-Algorithmen zeichnen sich dadurch aus, dass sie schnell passable Ergebnisse erzielen, aber nicht unbedingt optimale. Bei einem Algorithmus wird in einzelnen Berechnungsschritten nach und nach ein Gesamtergebnis entwickelt. Die Grundidee hinter Greedy-Algorithmen ist dabei, dass immer aus der lokalen Situation der beste Folgeschritt berechnet wird. Das schon errechnete Teilergebnis wird dabei nicht mit betrachtet. (Schöning, 2008, 18f)

Übertragen auf die Aufgabe der Zuordnung von Nachrichten zueinander bedeutet dies, dass ohne Berücksichtigung der gesamten bisher getroffenen Zuordnungen über die beste Folgezuordnung entschieden wird. Doch diese Grundentscheidung über die Konstruktion des Algorithmus sagt noch nicht aus, wie genau ein Algorithmus implementiert werden soll.

Die Idee, welche ich bei meinem Entwurf verfolgt habe, ist, dass die Nachrichten an einem Rechner in eine zeitliche Reihenfolge gebracht werden – der Reihenfolge, in der sie an einer bestimmten Netzwerkschnittstelle abgefangen wurden. Für jede Bewegungsnachricht wird dann in einem sinnigen Ausschnitt der restlichen Nachrichten nach einem inhaltlich passenden Partner gesucht.

Das hier zu wählende Vorgehen ist ein iteratives. Die Liste der Nachrichten wird Schritt für Schritt durchgegangen. Für jede ausgewählte Nachricht wird dann – ebenfalls iterativ – in einem Ausschnitt der übrigen Nachrichten nach einer Nachricht gesucht, die zu der ausgewählten Nachricht inhaltlich passt.

Um die Komplexität zu reduzieren, wird diese Zuordnung für jedes Paar von Klienten einzeln in eine Richtung betrachtet. Die Liste der Nachrichten für Klientenpaar (A,B) enthält dementsprechend nur MOVE-Nachrichten von A und MOVEOBJECT-Nachrichten an B, welche die ID von A betreffen. Diese Liste ist zeitlich aufsteigend nach dem Aufzeichenzeitpunkt der Nachrichten sortiert. Für eine Betrachtung von (B,A) würden dann die MOVE-Nachrichten von B und die MOVEOBJECT-Nachrichten an A betrachtet werden.

Der zu entwickelnde Algorithmus wird dann für jede mögliche Kombination von Klienten angewendet und dabei werden die Nachrichtenpaare ermittelt. Ein Beispiel soll dies verdeutlichen:

$$Klienten = \{A, B, C\}$$
$$Zu\,untersuchende\,Klienten{-}Paare = \{(A,B),(A,C),(B,A),(B,C),(C,A),(C,B)\}$$

Dabei werden die Kombinationen (A,A) (B,B) (C,C) ausgelassen, da es Ziel des Algorithmus ist, herauszufinden, wie lange Aktualisierungen von einem Nutzer zum anderen Nutzer brauchen. Nicht jedoch, wie schnell das System auf eigene Änderungen des Zustands reagiert.

Nun gibt es zwei Ansatzpunkte, um ein Paar von Nachrichten ausfindig zu machen. Man kann bei der Ursache, der MOVE-Nachricht, ansetzen und vorwärts nach der Wirkung suchen oder man startet bei der Wirkung, der MOVEOBJECT-Nachricht, und sucht rückwärts nach der Ursache. Da beide Ansatzpunkte betrachtenswert erscheinen, wird im Folgenden für jeden Ansatzpunkt ein Algorithmus entworfen. Anschließend werden die Eigenschaften der Algorithmen diskutiert.

Hauptziel der Arbeit ist nicht die Entwicklung von Algorithmen. Die Konstruktion der Algorithmen ist vielmehr Mittel zum Zweck. Die Algorithmen sollen helfen, das Bewertungskriterium der paarweisen Latenz für die zu untersuchende DVE Timadorus überhaupt messbar zu machen. Deswegen wird bei dem Entwurf der Algorithmen darauf verzichtet, deren Korrektheit zu beweisen – dies würde den Rahmen der Arbeit sprengen. Stattdessen wurden die Java-Implementierungen der Algorithmen ausführlich auf die Abwesenheit bestimmter Fehler überprüft. Die Java-Implementierungen können in Abschnitt 2 des Anhangs betrachtet werden. Die Unit-Tests der Implementierungen der beiden Algorithmen sind in den Abschnitten 3 und 4 des Anhangs dokumentiert.

4.5.2 Algorithmus ausgehend vom Senden der Zustandsänderung

Bei dem ersten von mir entworfenen Algorithmus wird davon ausgegangen, dass jede Ursache eine Wirkung hat. D.h. auf jede MOVE-Nachricht von Klient A erhält ein weiterer Klient B eine Aktualisierung der Bewegung von B. Ausgehend von der MOVE-Nachricht wird in den zeitlich geordneten Nachrichten vorwärts die darauf folgende passende MOVEOBJECT-Nachricht an B gesucht.

In Algorithmus 1 wird die Liste der Nachrichten durch den Parameter messages dargestellt. Die äußere Schleife (Zeile 5) iteriert über die gesamte Liste der Nachrichten. Für jede Nachricht wird überprüft, ob es es sich um ein MOVE-Nachricht handelt (Zeile 7). Sollte diese der Fall sein, wird ab der darauf folgenden Nachricht in der inneren Schleife (Zeile 10) nach einer passenden MOVEOBJECT-Nachricht gesucht. Sollte dies der Fall sein, wird das Nachrichtenpaar dem Ergebnis hinzugefügt (Zeile 13) und die innere Schleife beendet (Zeile 15). Der Index zum Iterieren der MOVE-Nachrichten wird um eins erhöht (Zeile 20) und die äußere Schleife fortgesetzt.

Es muss dabei berücksichtigt werden, dass wenn ein Klient zwei Mal hintereinander eine MOVE-Nachricht mit den gleichen Bewegungsinformationen versendet, keine Unterscheidung mehr getroffen werden kann, auf welche der beiden MOVE-Nachrichten sich eine entsprechende MOVEOBJECT-Nachricht bezieht. Deswegen werden nach der erfolgreichen Zuordnung von einem Nachrichtenpaar (siehe Zeile 14 in Algorithmus 1), die nächsten MOVE-Nachrichten so lange ignoriert, bis sie einen anderen Inhalt haben (siehe Zeile 7 in Algorithmus 1).

Algorithmus 1 Algorithmus zur Suche von Nachrichtenpaaren ausgehend von MOVE-Nachrichten

```
1:  function calculateMessagePairs(messages)
2:      moveIndex ← 0
3:      oldMovement ← empty
4:      result ← emptyList
5:      while moveIndex < messages.size do
6:          move ← messages[moveIndex]
7:          if move is Move AND move.movement ≠ oldMovement then
8:              newMovement ← move.movement
9:              moveobjectIndex ← moveIndex + 1
10:             while moveobjectIndex < messages.size do
11:                 moveobject ← messages[moveobjectIndex]
12:                 if moveobject is Moveobject AND moveobject.movement ==
    newMovement then
13:                     result.add((move, moveobject))
14:                     oldMovement ← newMovement
15:                     break
16:                 end if
17:                 moveobjectIndex ← moveobjectIndex + 1
18:             end while
19:         end if
20:         moveIndex ← moveIndex + 1
21:     end while
22:     return result
23: end function
```

Dieses Vorgehen steht immer noch im Einklang mit dem Greedy-Vorgehen. Nach Schöning (2008) ist die lokale Situation ausschlaggebend über die Entscheidung, welche Folgeschritt als bester zu bewerten ist. Was einen Greedy-Algorithmus auszeichnet ist, dass er auf eine globale Sicht der bisher errechneten Ergebnisse verzichtet. In dem vorgestellten Algorithmus wird keine Betrachtung der globalen Situation vorgenommen. Stattdessen wird die lokale Situation um das Wissen über die vorangehende Zuordnung erweitert. Somit kann der Algorithmus weiterhin als Greedy-Algorithmus verstanden werden.

4.5.3 Algorithmus ausgehend vom Empfangen der Zustandsaktualisierung

Es wird davon ausgegangen, dass jede Wirkung eine Ursache hat. D.h. für jede MOVEOBJECT-Nachricht an Klient B zur Bewegung eines Klienten A existiert eine MOVE-Nachricht des

Klienten A. Ausgehend von der MOVEOBJECT-Nachricht wird die Liste der zeitlich geordneten Nachrichten rückwärts nach einer entsprechenden MOVE-Nachricht durchsucht.

Der grundlegenden Unterschiede von Algorithmus 2 zu Algorithmus 1 sind die Wahl des Ausgangspunktes der Zuordnung und daraus folgend eine andere Richtung der Suche eines passenden Partners. Dies spiegelt sich darin wieder, dass die innere Schleife von der MOVEOBJECT-Nachricht ausgehend rückwärts über die Liste der Nachrichten iteriert. Der Index für die innere Schleife wird auf den um eins verringerten äußeren Index gesetzt (Zeile 9) und bei jedem Durchlauf um eins verringert (Zeile 17). Das Abbruchkriterium der Schleife ist folglich auch nicht die Länge der Liste der Nachrichten, sondern deren Beginn (Zeile 10).

Ebenfalls ist hier zu berücksichtigen, dass auch bei dieser Betrachtung zwei aufeinander folgende MOVEOBJECT-Nachrichten die gleiche Bewegung zur Grundlage haben können. Auch hier müssen deshalb aufeinander folgende Nachrichten mit gleicher Bewegung ignoriert werden (siehe Zeile 7 in Algorithmus 2).

4.5.4 Diskussion der Algorithmen

Konsistenz

Der Ansatz der Zuordnung von Nachrichten anhand des Inhalts hat einen Nachteil, der auch schon in den vorigen Abschnitten Erwähnung fand: Die Zuordnung ist nicht eindeutig, sie ist nur eine Vermutung. Dadurch ist die oben erwähnte sequentielle Konsistenz auf Serverseite durch keinen der beiden Algorithmen rekonstruierbar. Für die einzelnen Klientenpaare können durchaus unterschiedliche Reihenfolgen der Zustandsaktualisierungen erkannt werden.

Alternative Zuordnungsmöglichkeiten

Dadurch dass die Zuordnungen bloß einer Vermutung entsprechen, gibt es unterschiedliche Möglichkeiten, sie vorzunehmen. Es besteht die Möglichkeit dabei optimistisch oder pessimistisch vorzugehen. Wenn z.B. mehrere MOVE-Nachrichten kurz hintereinander mit gleichen Inhalt versandt wurden, besteht die Möglichkeit die erste oder die letzte Nachricht auszuwählen. Die Auswahl der ersten Nachricht wäre pessimistisch zu nennen, da ein größerer Zeitraum zwischen Ursache und Wirkung liegt. Bei der Auswahl der letzten Nachricht wird also von einen geringerem Zeitraum ausgegangen.

In den eben vorgestellten Zuordnungsalgorithmen wird in den inneren Schleifen eine optimistische Zuordnung vorgenommen. Eine pessimistische Zuordnung würde eine temporäre Zuordnung von Nachrichten-Paaren notwendig machen. Bei Algorithmus 1 würde dann eine darauf folgende MOVEOBJECT-Nachricht mit gleicher Bewegung die vorige Zuordnung

Algorithmus 2 Algorithmus zur Suche von Nachrichtenpaaren ausgehend von MOVEOBJECT-Nachrichten

```
1:  function calculateMessagePairs(messages)
2:      moveobjectIndex ← 0
3:      oldMovement ← empty
4:      result ← emptyList
5:      while moveobjectIndex < messages.size() do
6:          moveobject ← messages[moveobjectIndex]
7:          if moveobject is MoveObject AND moveobject.movement ≠ oldMovement
    then
8:              newMovement ← clientMessage.movement
9:              moveIndex ← moveobjectIndex − 1
10:             while moveIndex >= 0 do
11:                 move ← messages[moveIndex]
12:                 if move is Move AND move.movement == newMovement then
13:                     result.add((move, moveobject))
14:                     oldMovement ← newMovement
15:                     break
16:                 end if
17:                 moveIndex ← moveIndex − 1
18:             end while
19:         end if
20:         moveobjectIndex ← moveobjectIndex + 1
21:     end while
22:     return result
23: end function
```

ändern. Die endgültige Zuordnung würde erst dann erfolgen, wenn eine weitere MOVEOBJECT-Nachricht mit einer anderen Bewegung gefunden würde. Bei Algorithmus 2 würde dies analog mit MOVE-Nachrichten geschehen.

Eine dementsprechende Änderung der Zuordnung hin zu einer pessimistischen würde jedoch nichts an dem prinzipiellen Vorgehen ändern. Diese Änderung würde vor allem zu Zuordnungen von weiter auseinander liegenden Nachrichten führen und damit zu höheren Werten für die paarweise Latenz. Da aber die empirischen Daten über die zu untersuchende DVE nur als Mittel zur Identifizierung von Schwierigkeiten beim Messen dienen, ist deren absolute Größe nicht relevant. Deswegen wird diese alternative Zuordnungsweise nicht weiter untersucht.

Mehrfachzuordnung von Nachrichten

Das Protokoll sieht vor, dass ein Klient den Server höchstens dann über seine Bewegung informiert, wenn er seine Richtung und Rotation ändert. Der Server hingegen informiert die anderen Klienten regelmäßig über die aktuellen Position des Klienten. Eine MOVE-Nachricht kann, wie oben erwähnt, mehrere MOVEOBJECT-Nachrichten nach sich ziehen, die sich nur in den aktuellen Koordinaten unterscheiden. Hier ist nach einer erfolgten Zuordnung sicherzustellen, dass die nächste Zuordnung einen anderen Inhalt hat, so dass nicht eine MOVE-Nachricht mehreren Ursache-Wirkung-Paaren zugeordnet wird.

In dem hier vorgestellten Ansatz wird auch dieses Problem nicht vollkommen ausgeschlossen. Zwar werden durch das Ignorieren von kurz hintereinander folgenden gleichen Bewegungen gewisse Fälle ausgeschlossen. Trotzdem können in Algorithmus 2 einer MOVE-Nachricht mehrere MOVEOBJECT-Nachrichten zugewiesen werden und in Algorithmus 1 einer MOVEOBJECT-Nachricht mehrere MOVE-Nachrichten. Dies liegt daran, dass beide Algorithmen nicht überprüfen, ob eine Wirkung (Algorithmus 1) bzw. eine Ursache (Algorithmus 2) schon einmal in einer Zuordnung verwendet wurde. Es ist auch fraglich, ob eine Ergänzung der Algorithmen um solch ein Gedächtnis sinnvoll wäre. Sollte eine falsche Zuordnung vorliegen, würde diese fortan eine darauf folgende richtige Zuordnung verhindern, da die beteiligten Nachrichten von einer erneuten Zuordnung zu ihrem richtigen Partner ausgeschlossen wären.

Weitere Probleme können bei der Zuordnung entstehen, sollte der Server die Bewegung ohne Nachricht des Klienten ändern oder der Server eine Bewegungsänderung des Klienten ignorieren oder aber beim Aufzeichnen eine Nachricht verloren gehen. Dann kann es sein, dass einer Wirkung eine falsche Ursache zugewiesen wird, da beide Algorithmen über die vollständige Liste der Nachrichten iterieren und so lange suchen, bis sie ein vom Inhalt her passendes Paar gefunden haben. Dies ist im Fall von Algorithmus 1 zwar eingeschränkt auf alle zeitlich nach der MOVE-Nachricht liegenden MOVEOBJECT-Nachrichten und im Fall von Algorithmus 2 auf alle zeitlich vor der MOVEOBJECT-Nachricht liegenden MOVE-Nachricht. Nichtsdestotrotz ist hier das grundsätzliche Problem zu erkennen, dass die Algorithmen eine geringe Fehlertoleranz haben.

Fazit

Beide Algorithmen stellen einen Versuch dar, eine Zuordnung von Nachrichtenpaaren vorzunehmen und damit die paarweise Latenz messbar zu machen. Dabei sind ihre diskutierten Schwächen zu berücksichtigen. Trotzdem scheint das grundlegende Vorgehen der inhaltlichen Zuordnung zeitlich aufeinander folgender Nachrichten geeignet, eine grobe Zuordnung vorzunehmen. Eine

exakte Zuordnung würde das Messen zwar präziser machen, muss aber nicht zwangsläufig notwendig sein, um Tendenzen in den paarweisen Latenzen zu erkennen. Dies kann auch mit abgeschätzten Zuordnungen möglich sein.

5 Das Modell in der Praxis

5.1 Versuchsziele

Ziel des im folgenden dargestellten Versuchs ist es, die Überlegungen zum Modell einem praktischen Test zu unterziehen. In einem Versuch soll die Aussagekraft des Ansatzes der inhaltlichen Zuordnung von Nachrichten zur Berechnung der paarweisen Latenz beispielhaft evaluiert werden. Ziel ist es nicht, die Leistungsfähigkeit des vorgestellten Systems Timadorus zu untersuchen. Vielmehr wird anhand eines Systems untersucht, inwieweit mit den vorgestellten Überlegungen die Leistung dieser zu untersuchenden DVE bewertet werden kann.

In dem Versuch werden beide vorgestellten Algorithmen einem praktischen Test unterzogen. Um zu schauen, wie aussagekräftig die errechneten Zeiten sind, muss jeder Algorithmus auf ein System mit unterschiedlicher Skalierung angewandt werden. Entsprechend der Skalierbarkeit hinsichtlich der Größe bedeutet dies, dass das System bei einer variierenden Anzahl der Klienten untersucht wird.

5.2 Versuchsanordnung

Um die vorigen Überlegungen praktisch zu überprüfen, wird folgender Aufbau verwendet: Es wird ein Server auf einem Computer verwendet. Auf weiteren Rechnern werden je mindestens zwei Klienten gestartet. Dort wird jeweils der Netzwerkverkehr mitgeschnitten. Die Mitschnitte können dann dekodiert und anschließend mit Hilfe der Algorithmen ausgewertet werden. Abbildung 5.1 zeigt den generellen Aufbau mit nur einem Computer, auf dem Klienten laufen. Diese Anordnung kann jedoch um weitere Rechner mit Klienten erweitert werden.

Sowohl für die Klienten-Rechner als auch für den Server-Rechner wurden Computer im Labor für Allgemeine Informatik an der HAW Hamburg verwendet. Diese sind mit einem Linux-Betriebssystem ausgestattet und befinden sich in einem gemeinsamen Gigabit-Ethernet.

Durch das Mitschneiden auf Klienten-Seite werden zwei Dinge erreicht. Zum einen wird das Zeitsynchronisierungsproblem umgangen. Die Zeitstempel beim Mitschneiden der ausgehenden und der eingehenden Nachrichten werden mit Hilfe der Uhr desselben Rechners erstellt. Dies

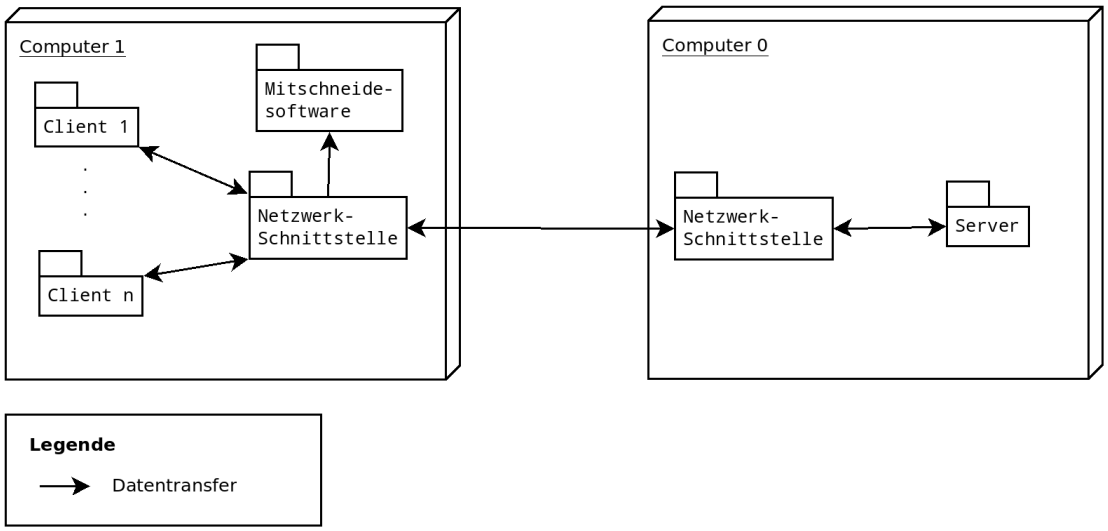

Abbildung 5.1: Aufbau des Versuchs. Exemplarisch mit einem Server und einem weiteren Computer mit Klienten

bedeutet aber auch, dass die einzelnen Mitschnitte separat den Zuordnungsalgorithmen unterworfen werden müssen. Denn wenn die Dauer der Zustandsaktualisierungen von dem einem Klienten auf einem Rechner zu dem anderen Klienten auf einem weiteren Rechner betrachtet würde, wäre das Zeitsynchronisierungsproblem wieder vorhanden. Beide Rechner könnten unterschiedlich synchronisierte Uhren haben.

Ein weiterer Vorteil des Mitschneidens des Netzwerkverkehrs auf Klienten-Seite ist, dass die Auslastung des Netzwerks mit in die Messung einbezogen wird. Gerade durch die Skalierung der DVE werden mehr Klienten den Zustand ändern und mehr Klienten werden Zustandsaktualisierungen benötigen, damit jeder einen kohärenten Blick auf die virtuelle Welt hat. Die entstehende Netzwerklast in Form von verzögerten Nachrichtenübermittlungen wird mit berücksichtigt, da der Weg übers Netzwerk vom Klienten-Rechner zum Server wieder zurück zum gleichen Klienten-Rechner mit in der Messung enthalten ist.

In der Definition von verteilten virtuellen Umgebungen in Kapitel 2.1 wurde festgehalten, dass DVEs auch auf mehreren Knoten laufen können. Im Versuchsaufbau mit nur einem Server ist dies nicht berücksichtigt. Dies ist der aktuellen Implementation des Servers von Timadorus geschuldet, die es momentan nicht unterstützt, auf mehreren Rechnern zum Einsatz zu kommen. Trotzdem beeinträchtigt dies die Auswertung der Algorithmen nicht. Denn die Algorithmen verwenden als Grundlage die auf Klienten-Seite mitgeschnittenen Nachrichten. Für die Anwendung der

Algorithmen ist es irrelevant zu oder von welchem Server die Nachrichten geschickt wurden. Somit ist die Anordnung prinzipiell um weitere Server erweiterbar.

Das tatsächliche Durchführen eines Versuches besteht aus vier Schritten:

- dem Starten des Servers
- dem Starten der Software zum Mitschneiden des Verkehrs
- dem Starten der Klienten
- dem Auswerten der gesammelten Mitschnitte

Dabei sind folgende Bedingungen einzuhalten: Erstens muss der Server vor den Klienten gestartet werden, da die Klienten sich sonst nicht beim Server anmelden können. Zweitens muss die Software zum Mitschneiden vor den Klienten gestartet werden. Die letzte Bedingung stellt sicher, dass die Anmeldung der Klienten bei dem Server mitgeschnitten wird. Dies ist notwendig, um die ENTERING-Nachrichten mitzulesen, die erst eine Zuordnung des Klienten zu einer bestimmten ID rekonstruierbar machen.

Ein einzelner Versuchslauf besteht aus folgenden Konfigurationsparametern: Anzahl der Klienten pro Rechner, Anzahl der Rechner, Dauer der Messung. Die Ausführung erfolgt mit Hilfe eines Bash-Skriptes. Dieses startet den Server, loggt sich per SSH auf den Rechnern ein und startet vor Ort entsprechend viele Klienten.

5.3 Implementierung

5.3.1 Abfangen

Zum Abfangen und Speichern der Pakete wird das Programm tcpdump verwendet. Es handelt sich um ein Werkzeug zum sog. network sniffing – ein Programm zum Mitschneiden von Netzwerkverkehr. Das Programm verfolgt dabei drei Prinzipien: Erstens ist das Abfangen der Pakete passiv und beeinflusst den Netzwerkverkehr nicht. Zweitens können nur Pakete abgefangen werden, welche das eigene System physikalisch empfängt. Drittens können nur Pakete abgefangen werden, die an das eigene System adressiert sind. Letzteres lässt sich durch den sog. promiscuous mode der eigenen Netzwerkschnittstelle umgehen. (Styn, 2011, 1ff)

Das Programm tpcdump verwendet dabei die Bibliothek LIBPCAP, wobei PCAP für packet capture steht. Die Bibliothek wurde entwickelt, um als Rückgrat von tcpdump zu wirken. Sie definiert ein Format zum Speichern und Lesen mitgeschnittener Pakete. In den durchgeführten

Versuchen wurde die Funktionalität von tcpdump zum Speichern der Pakete in diesem Format genutzt, um sie einer anschließenden Auswertung zugänglich zu machen.

Um tcpdump nur für eine bestimmte Dauer laufen zu lassen, wird es mit einem Bash-Skript gestartet. In dem Skript wird tcpdump als Hintergrundprozess gestartet, die Prozess-ID des Hintergrundprozesses wird gespeichert und das Skript für die Dauer der Messung angehalten. Anschließend wird die Prozess-ID verwendet, um den Hintergrundprozess zu beenden.

Während der Vorbereitung des Versuchs wurde evaluiert, das Framework JNetPCAP zu verwenden. Dabei handelt es sich um Open Source Java-Framework, welches ebenfalls einen Zugriff auf die Funktionalitäten von LIBPCAP ermöglicht. Das Speichern der Pakete in einer Datei ist damit ebenso möglich wie deren Vorhalten im Speicher. Der Nachteil besteht jedoch darin, dass auf den Versuchsrechnern Berechtigungen notwendig sind, damit das entsprechende Java-Programm die Pakete auf der Netzwerkschnittstelle mithören kann. Da Java-Programme jedoch über das Ausführen in der Java Virtual Machine (JVM) gestartet werden, hätten der JVM und damit allen auf der JVM laufenden Programmen diese Rechte zugewiesen werden müssen, was aus Sicherheitsgründen nicht möglich war. Darüber hinaus bestehen Abhängigkeiten zu einer speziellen Entwicklerversion von LIBPCAP, welche auf den Versuchsrechnern nicht zur Verfügung stand.

5.3.2 Dekodieren und Evaluieren

Die zum Dekodieren und Evaluieren notwendige Software wurde in Java implementiert. Sie besteht dabei aus vier unterschiedlichen Funktionseinheiten:

- Auslesen der Paketinformationen aus Dateien
- Dekodieren der Pakete zu Nachrichten
- Zuordnung von Nachrichtenpaaren mit Hilfe von Algorithmen
- Auswertung der Nachrichtenpaare hinsichtlich statistischer Werte

Dazu kommt eine graphische Benutzeroberfläche, über welche das Auswerten der einzelnen Versuchsläufe vorgenommen werden kann. Außerdem bietet die Oberfläche die Option, die jeweils mit einem Algorithmus ermittelten Nachrichtenpaare als comma-separated values (CSV) zu exportieren, wodurch die Ergebnisse auch in Tabellenkalkulationsprogrammen wie Excel verwendet werden können.

Der Ablauf der Auswertung ist wie folgt: Es wird jede Datei einzeln ausgelesen. Eine Datei enthält dabei die auf einem Rechner mitgeschnittenen Paketinformationen. Dann werden die

Nachrichten dekodiert und anschließend für die einzelnen Klientenpaare die Zuordnung mit den entsprechenden Algorithmen durchgeführt. Wenn alle Dateien ausgelesen und die Zuordnungen durchgeführt wurden, wird eine Liste mit allen Paaren erstellt. Diese kann dann exportiert werden. Außerdem werden kurze statistische Auswertungen vorgenommen.

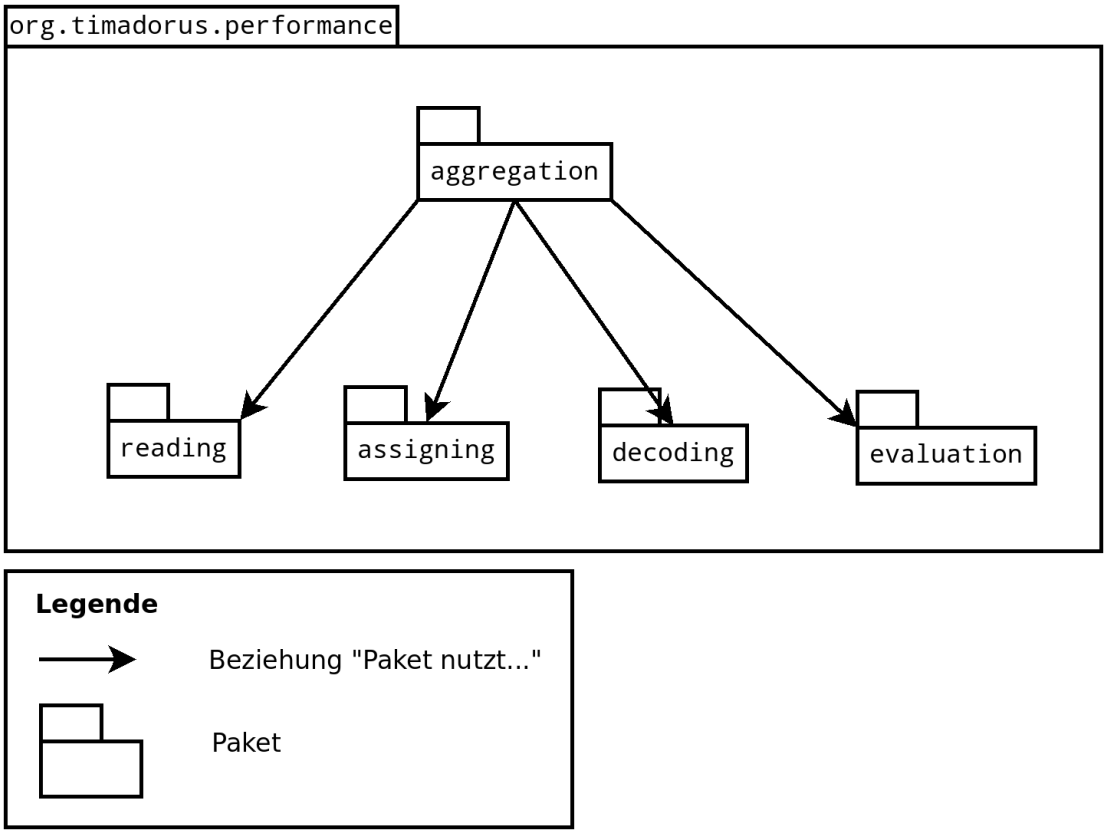

Abbildung 5.2: Vereinfachte Architektur des Auswertungsprogrammes

Die einzelnen Komponenten kennen einander dabei nicht. Vielmehr definieren sie einen notwendigen Input und einen Output. Eine weitere Klasse muss diese Komponenten instanzieren und den Input und Output zwischen den Komponenten weiterleiten. Hierdurch wird eine lose Kopplung erreicht und die Komponenten bleiben austauschbar. Zwar teilen sie sich ähnliche Klassen, aber diese sind in dem Paket common zusammengefasst. Common hat wiederum keinerlei Abhängigkeiten zu anderen Paketen.

Der Aufbau ist in Abbildung 5.2 veranschaulicht. Das Paket reading übernimmt das Auslesen der Paketinformationen aus Dateien, das Paket decoding das Dekodieren der Pakete zu Nachrichten, das Paket assigning übernimmt die Zuordnung von Nachrichtenpaaren mit Hilfe

von Algorithmen und das Paket evaluation die Auswertung der Nachrichtenpaare hinsichtlich statistischer Werte. Die Abbildung lässt zur besseren Verständlichkeit das Paket common weg. Jedes der dargestellten Pakete kann eine oder mehrere Abhängigkeiten zu common haben.

Ein weiterer Punkt ist in dieser Abbildung nicht erfasst. Jedes der vier Pakete, die eine der funktionalen Komponenten des Programms repräsentieren, wird nach außen durch ein Interface sowie eine Klasse vertreten, in der mittels Factory-Methoden Objekte vom Typ des jeweiligen Interfaces erstellt werden können. Alle anderen Klassen und Interfaces sind package private und von außen nicht sichtbar.

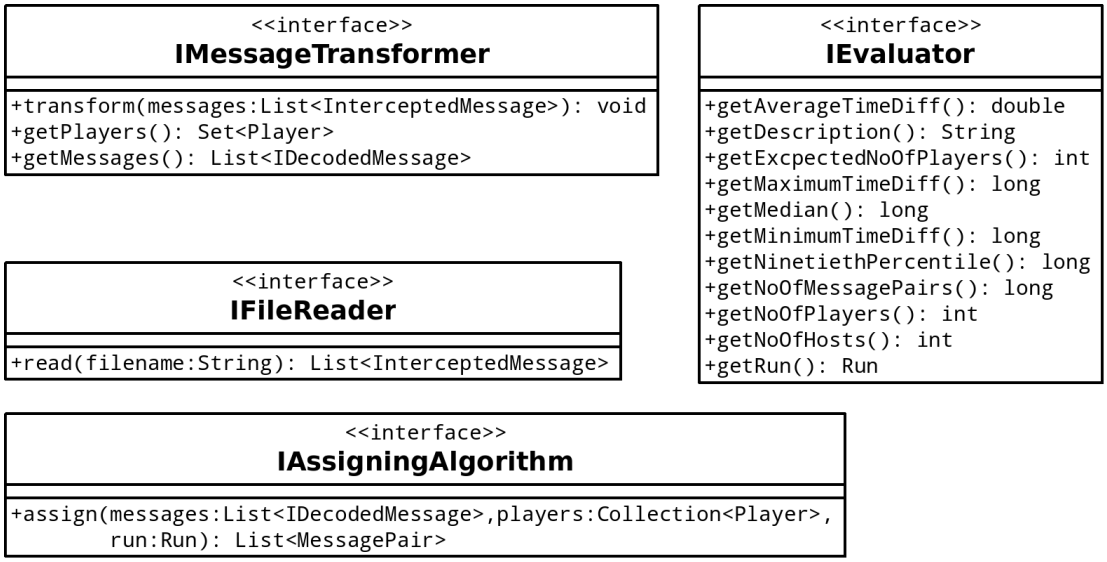

Abbildung 5.3: Öffentliche Interfaces der funktionalen Komponenten in UML-Notation

Abbildung 5.3 zeigt die öffentlichen Interfaces der Pakete der funktionalen Komponenten. IMessageTransformer ist das Interface für das Paket decoding, IFileReader für das Paket reading, IAssigningAlgorithm für das Paket assigning und IEvaluator für das Paket evaluation. Aus der kurzen Darstellung wird schon sichtbar, dass in der Definition der Interfaces jeweils keine Interfaces der Pakete der anderen funktionalen Komponenten verwendet werden.

Doch wie funktionieren diese Pakete intern? Zum Auslesen der Paketinformationen aus den mitgeschnittenen Dateien wird das oben erwähnte Framework JNetPCAP verwendet. Da hier nur Dateien ausgelesen werden, bestehen die Berechtigungsprobleme nicht. Ebenfalls muss die Auswertung nicht auf den Versuchsrechnern vorgenommen werden, so dass die Probleme mit den Abhängigkeiten leicht auf einem anderen Computer behoben werden können.

Timadorus verwendet eine Bibliothek names timadorus-util. Diese wurde eigens für Timadorus entwickelt und enthält sämtliche Funktionalitäten zum Parsen der Timadorus-Nachrichten. Empfangene Daten müssen dabei nicht unbedingt eine vollständige und damit parsbare Nachricht repräsentieren. Stattdessen werden die erhaltenen Daten zuerst gepuffert und dann dekodiert, wenn sie vollständig sind. Um dies zu berücksichtigen, muss das Dekodieren der Nachrichten für den Server sowie für jeden einzelnen Klienten mit einem separaten Puffer vorgenommen werden. Es wird somit beim nachträglichen Dekodieren der Pakete aus der Datei das Verhalten für jeweils Server und Klienten nachgebildet.

Beim Dekodieren werden ebenfalls die ENTERING-Nachrichten ausgewertet, so dass die Spieler auf der jeweiligen Maschine, den entsprechenden Ports zugeordnet werden können. Diese sowie andere für die weitere Auswertung irrelevanten Nachrichten können dann sofort verworfen werden. Über die Methoden getPlayers und getMessages des Interfaces IMessageTransformer können im Anschluss die Informationen über die Spieler sowie alle Nachrichten entnommen werden.

Die dekodierten Nachrichten werden in einer Liste in eine zeitliche Reihenfolge gebracht. Dabei ist der Zeitpunkt des Abfangens das entscheidende Kriterium. Für jede Datei werden die Klientenpaare gebildet. Daraufhin werden die Nachrichten so gefiltert, dass immer nur MOVE-Nachrichten eines Klienten A sowie MOVEOBJECT-Nachrichten mit der ID von A an den Klienten B in der gefilterten Liste enthalten sind. Eine Java-Implementierung der oben erwähnten Algorithmen nimmt dann die Zuordnung vor. Für jeden der beiden Algorithmen existiert eine konkrete Klasse, welche das Interface IAssigningAlgorithm implementiert.

Die zugeordneten Nachrichtenpaare aller Klientenpaare sowie aller Dateien werden dann in einer Liste pro Algorithmus zusammengeführt und zeitlich sortiert. Die Liste der Nachrichtenpaare kann dann für den Export als CSV verwendet werden. Ebenfalls dient sie als Grundlage für die sofortige statistische Auswertung und die Anzeige dieser Werte in der grafischen Oberfläche.

Aus der bisherigen Beschreibung der Funktionsweise und des Aufbaus sollte deutlich geworden sein, dass Abhängigkeiten zu anderen Programmen existieren. Um den Abhängigkeiten bei der Entwicklung Herr zu werden, wurde das Programm als Maven-Projekt aufgesetzt. Hierdurch lassen sich die Abhängigkeiten sowie der Build-Prozess der Software verwalten. Ebenfalls bietet Maven die Optionen, den Release der Software sowie die Versionsnummer zu verwalten.

5.3.3 Die verteilte virtuelle Umgebung

Um Last zu generieren, wird in Timadorus ein Bot-Klient verwendet, welcher mit Hilfe eines Algorithmus zur künstlichen Intelligenz Kohlköpfe in der virtuellen Welt konsumiert.

Die Rehe müssen sich dabei nicht immer sehen. Das in Timadorus implementierte Interessenmanagement hat zum Ziel, dass sich nur Klienten, die sich innerhalb einer bestimmten Entfernung zueinander befinden, gegenseitig sehen (siehe 4.3). Die künstliche Intelligenz des Rehs hat nicht zur Aufgabe, in der Sichtweite der anderen Rehe zu bleiben. Dies hat Auswirkungen auf die Verwendung der Algorithmen zur Zuordnung der Nachrichten.

Diese Algorithmen gehen in ihren grundsätzlichen Ansätzen („Es gibt für jede Wirkung eine Ursache“ bzw. „Jede Ursache hat eine Wirkung“) davon aus, dass alle Bewegungsänderungen eines Klienten bei dem anderen Klienten ankommen. Der Lastgenerator muss also sicherstellen, dass sich die zu betrachtenden Klientenpaare in dem gleichen Sichtbarkeitsbereich befinden. Sonst könnte es dazu kommen, dass z.B. ein Klient eine bestimmte Richtung einschlägt und nach längerer Zeit in die Sichtweite eines weiteren Klienten läuft. Die gemessene Zeit wäre dann von der Richtungsänderung bis hin zum Eintreten in die Sichtweite, was nicht unbedingt etwas über die paarweise Latenz und damit über die Leistung der DVE aussagt.

Um den Lastgenerator dennoch verwenden zu können, wurde das Interessenmanagement des Servers manipuliert. Fortan gelten folgende Kriterien, um einen zweiten Klienten über die Zustandsaktualisierung zu informieren: Entweder beide befinden sich in Sichtweite oder beide Klienten laufen auf dem gleichen Rechner.

So kann gewährleistet werden, dass alle interessanten Zustandsaktualisierungen, nämlich die von Klienten untereinander auf einer Maschine, immer weitergeleitet werden. Die gegenseitige Sichtbarkeit ist damit gewährleistet. Jedoch muss berücksichtigt werden, dass hierdurch nicht zwangsläufig ein realistischeres Szenario für die Last des Servers entsteht. Was entsteht, ist eine Welt, in der die Klienten, die vom gleichen Rechner kommen, bezüglich der Zustandsaktualisierungen so behandelt werden, als wenn sie immer in gegenseitiger Sichtweite wären.

5.4 Durchführung

Auf den Rechnern der HAW Hamburg wurden mehrere Versuchtsdurchläufe mit einer jeweiligen Dauer von fünf Minuten durchgeführt. Dabei wurden pro Computer zwei Klienten gestartet. Zu Beginn wurden drei Läufe mit nur einem Computer durchgeführt. Die Anzahl der Computer wurde schrittweise um einen Computer erhöht und jeweils drei Läufe mit einer Computerzahl ausgeführt. Dies wurde bis inklusive einer Computeranzahl von acht fortgeführt. Durch die schrittweise Erhöhung sollten unterschiedliche Grade der Belastung der DVE simuliert werden.

Die Dauer wurde auf fünf Minuten festgelegt, da stichprobenhafte Versuche mit unterschiedlichen Werten für der Dauer ergeben haben, dass deutlich größere Werte nicht zu wesentlich mehr

Nachrichtenpaaren geführt haben. Andererseits ist eine gewisse Mindestdauer notwendig, um überhaupt etwas zu messen. Fünf Minuten erwiesen sich dabei als guter Wert.

Pro Computer sind mindestens zwei Klienten notwendig, um die Auswertung vorzunehmen. Bei weniger Klienten pro Computer müsste eine Auswertung der Nachrichten von paarweisen Latenzen über mehrere Computer hinweg erfolgen, wodurch es Probleme mit nicht synchronen Uhren geben kann. Denn in verteilten Systemen ist nicht gewährleistet, dass die Uhren der Computer synchron laufen. Die Erhöhung auf mehr als zwei Klienten wäre eine Option gewesen, wenn mit der bestehenden Anzahl an Computern das System nicht hätte an seine Skalierungsgrenzen gebracht werden können. Andererseits ist eine geringe Zahl von Klienten pro Rechner wünschenswert, da so sichergestellt werden kann, dass die erbrachte Leistung der Klienten nicht durch die Belastung des einzelnen Computers durch zu viele Klientenprozesse beeinträchtigt ist.

5.5 Ergebnisse

Die Mitschnitte wurden jeweils mit beiden Algorithmen evaluiert und für jeden Lauf und angewandten Algorithmus das arithmetische Mittel (Durchschnitt), der Median, die 90. Perzentile, das Maximum und Minimum errechnet. Daraufhin wurde für jede identische Konfiguration das arithmetische Mittel der statistischen Werte ermittelt. D.h. dass z.B. das arithmetische Mittel der 90. Perzentilen aller Läufe mit 2 Klienten pro Rechner, einem Rechner insgesamt und einer Dauer von 5 Minuten ermittelt wurde.

Im Folgenden werden nun die ermittelten statistischen Werte dieser Läufe genauer betrachtet. Dabei stehen sowohl die Frage nach den Unterschieden zwischen den beiden Algorithmen im Mittelpunkt als auch nach deren Gemeinsamkeiten. Ein deutliches Ergebnis für die Werte von Durchschnitt, Median und 90. Perzentile zeigen die beiden Grafiken 5.4 und 5.5. Mit steigender Klientenzahl nehmen alle drei betrachteten Werte zu – und dies unabhängig vom Algorithmus. Auffällig ist bei beiden Algorithmen, dass die paarweisen Latenzen sehr stark mit der Anzahl der Klienten zunehmen.

Der Median bezeichnet den Wert, der in einer aufsteigenden Sortierung aller gemessenen Werte genau in der Mitte liegt. Er gilt deshalb als robust gegenüber statistischen Ausreißern. Betrachtet man die mit beiden Algorithmen ermittelten Mediane (siehe Tabelle 5.1) genauer, zeigt sich, dass dieser vergleichsweise robuste Werte mit der Anzahl der Klienten ebenfalls zunimmt. Von zwölf zu 14 Klienten erfolgt dann ein Sprung, der im Vergleich zu den vorigen Anstiegen größer ist. Hier nimmt der Median der paarweisen Latenz mehr als das sechsfache des vorigen Wertes an. Zudem liegen die Werte beider Algorithmen dicht beieinander. Algorithmus 2 ist

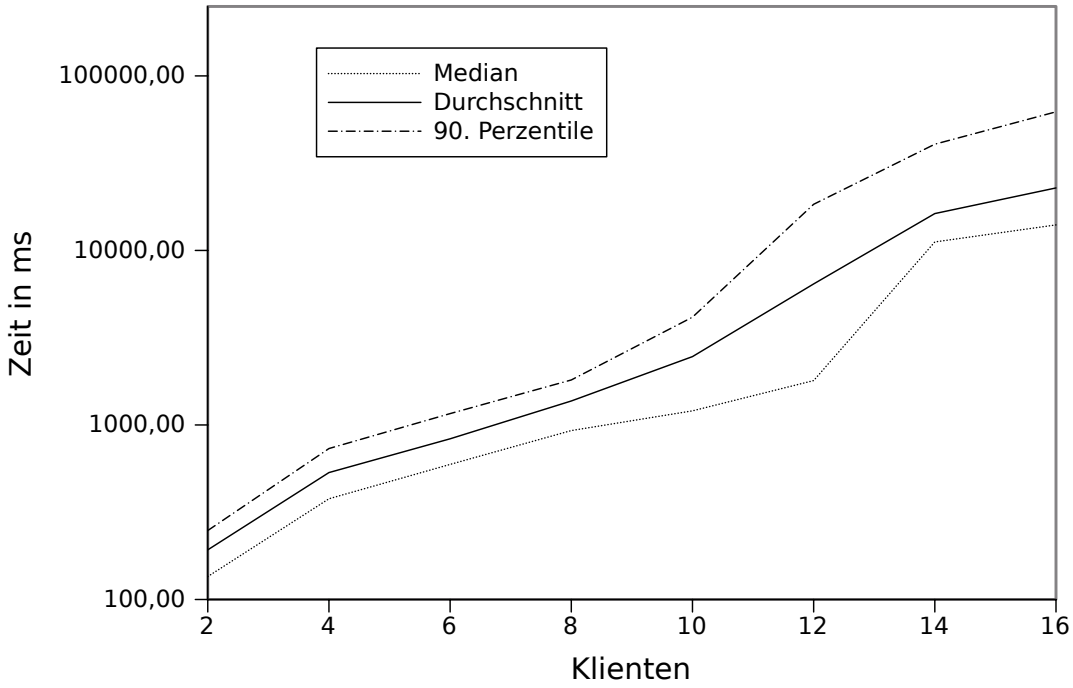

Abbildung 5.4: Algorithmus 1: Gemittelte Werte über drei Läufe mit einer Dauer von je 5 Minuten

meist niedriger, außer bei 14 Klienten. Hier wären sicherlich weitere Läufe notwendig, um diese Tendenz bezüglich der Algorithmen zu bestätigen.[1]

Durchschnittswerte sind als Mittel aller Werte weniger robust gegen Ausreißer als der Median. Bei der Betrachtung der gemittelten Durchschnittswerte (siehe Tabelle 5.2) sind die Unterschiede zwischen den Algorithmen größer als beim Median, was auf unterschiedlich Anzahl an Ausreißern hindeutet. Ebenfalls lässt sich hier zeigen, dass, auch wenn die Werte im Vergleich höher sind, ein kontinuierlicher Anstieg vorliegt. Auch hier liegt ein besonders hoher Anstieg ab einer bestimmten Zahl Klienten vor. Dies ist jedoch schon beim Übergang von acht auf zehn Klienten zu beobachten. Bis zehn Klienten weist der Algorithmus 2 im Vergleich geringere Werte auf und ab zwölf Klienten hat Algorithmus 1 deutlich höhere Werte.

Eine genauere Betrachtung der durchschnittlichen 90. Perzentilen (siehe Tabelle 5.3) bestätigt die zu den Durchschnittswerten gemachte Beobachtung. Es gibt einen bestimmten Punkt ab dem die Zunahme deutlich größer ist als bei den vorigen Zunahmen von zwei Klienten. Hier liegt der

[1] Aufgrund von Bauarbeiten an der HAW Hamburg waren zum Zeitpunkt der Auswertung der Versuche keine weiteren Versuchsdurchläufe möglich.

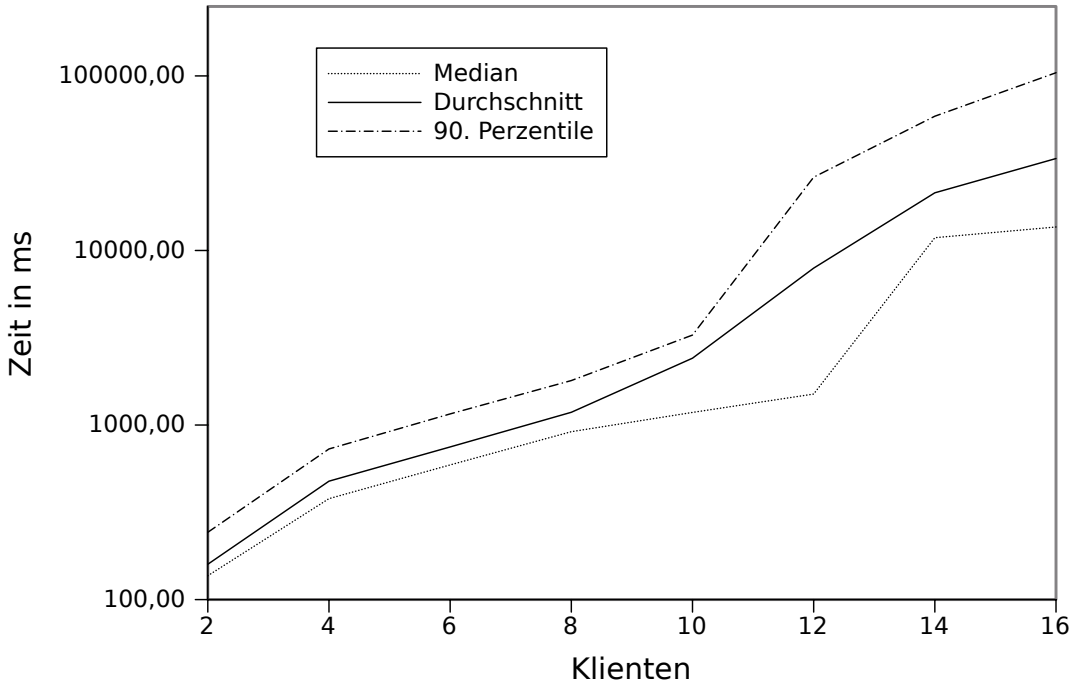

Abbildung 5.5: Algorithmus 2: Gemittelte Werte über drei Läufe mit einer Dauer von je 5 Minuten

Punkt zwischen zehn und zwölf Klienten. Die 90. Perzentile der paarweisen Latenz steigt hier auf mehr als das vierfache an.

Auch hier liegt bei der vergleichenden Betrachtung der Algorithmen der Wechsel zwischen den Algorithmen, d.h. wessen Wert höher ist, zwischen zehn und zwölf Klienten. Auffällig ist, dass die 90. Perzentile extrem hohe Werte annimmt. Dies lässt darauf schließen, dass die Anteil der Ausreißer nach oben zugenommen hat. Vor allem bei der Klientenanzahl von 16 wird deutlich, dass bei vielen Nachrichtenpaaren wohl eine falsche Zuordnung stattgefunden hat. Denn eine paarweise Latenz von 1 min und 44 s erscheint eher unrealistisch.

Weiter sind noch die gemittelten Maxima und Minima zu betrachten. Bei Algorithmus 1 steigen die gemittelten Minima mit der Klientenzahl an – mit Ausnahme von zwei Ausreißern (von acht zu zehn und von zehn zu zwölf Klienten) – wohingegen bei Algorithmus 2 keine solche Tendenz zu erkennen ist. Beide Algorithmen weisen bei den Minima paarweise Latenzen im einstelligen und niedrigen zweistelligen Bereich auf.

Bei den Maxima hat Algorithmus 1 Abweichungen vom kontinuierlichen Anstieg der Werte. Insgesamt besteht allerdings schon ein Unterschied zwischen dem niedrigen vierstelligen Wert bei

Klienten	Algorithmus 1	Algorithmus 2
2	135,33	137,00
4	377,67	378,67
6	594,33	591,00
8	928,67	916,00
10	1204,67	1181,00
12	1796,00	1505,00
14	11185,00	11835,33
16	14004,33	13635,33

Tabelle 5.1: Vergleich der Algorithmen: Gemittelter Median der paarweisen Latenz in ms

Klienten	Algorithmus 1	Algorithmus 2
2	192,50	159,38
4	533,58	476,85
6	833,13	749,03
8	1371,98	1183,15
10	2463,95	2415,39
12	6427,96	7922,79
14	16263,49	21395,79
16	22807,67	33629,97

Tabelle 5.2: Vergleich der Algorithmen: Gemittelter Durchschnitt der paarweisen Latenz in ms

zwei Klienten sowie den sechsstelligen Werten ab 14 Klienten. Eine Zunahme ist also erkennbar. Anzumerken ist zudem, dass diese hohen Werte der Maxima ein Drittel bis die Hälfte der gesamten Messdauer ausmachen.

5.6 Bewertung der Ergebnisse

Zusammenfassend lässt sich sagen, dass umso mehr Klienten die DVE nutzen, umso größer werden Median, Durchschnitt und 90. Perzentile der paarweisen Latenzen. Ebenso konnte mit diesen drei statistischen jeweils ein Punkt identifiziert werden, ab dem es zu einem starken Leistungsabfall kommt – die DVE also an ihre Grenze gelangt. Das Modell und die für dieses entworfenen Algorithmen erfüllen also die im Rahmen der Aufgabenstellung gestellte Anforderung, die stärkere Belastung des Server nachzuzeichnen.

Wenn man sich die gemessen Daten genauer anguckt, wird deutlich, dass, wie schon bei der Konstruktion der Algorithmen erwähnt, die Zuordnung nicht unbedingt präzise erfolgt. Die Maxima nehmen mit der Anzahl der Klienten so zu, dass die gemessenen Werte aufgrund ihrer

Klienten	Algorithmus 1	Algorithmus 2
2	248,33	243,00
4	732,67	729,33
6	1161,67	1158,00
8	1810,67	1798,00
10	4146,00	3276,33
12	18382,33	26295,00
14	40612,67	58775,33
16	62254,67	104299,33

Tabelle 5.3: Vergleich der Algorithmen: Gemittelte 90. Perzentile der paarweisen Latenz in ms

Klienten	Minimum		Maximum	
	Algorithmus 1	Algorithmus 2	Algorithmus 1	Algorithmus 2
2	3,67	26,00	5617,67	522,33
4	9,33	5,33	15857,67	1155,33
6	14,33	10,67	20595,67	2552,67
8	39,67	24,00	97122,00	11740,00
10	29,33	29,33	49553,00	48606,67
12	26,67	17,67	62236,00	81999,67
14	52,00	52,00	129893,33	111689,67
16	66,67	18,67	109702,33	154657,00

Tabelle 5.4: Vergleich der Algorithmen: Gemittelte Minima und Maxima der paarweisen Latenz in ms

Höhe als unrealistisch einzustufen sind. Dies deutet auf gelegentliche falsche Zuordnungen hin. Beide Algorithmen haben Probleme, wenn ihre Annahmen („Es gibt für jede Wirkung eine Ursache“ oder „Jede Ursache hat eine Wirkung“) nicht mehr mit der Realität übereinstimmen. Der Median der paarweisen Latenzen scheint von diesem Effekt wenig beeinflusst. Jedoch taucht dieses Problem der unrealistischen Werte auch bei der 90. Perzentile auf. Dies deutet darauf hin, dass die fehlerhaften Zuordnungen bei einigen Messungen, besonders unter hoher Last, überhand nehmen. Die Algorithmen erweisen sich hier also als instabil.

Anhand des hohen Anteils an scheinbar fehlerhaften Zuordnungen stellt sich die Frage, inwieweit der so stabil wirkende Median eigentlich noch ein Zentralwert darstellt. Dieser Effekt ist ein deutliches Indiz auf die in Abschnitt 4.5.4 festgehaltenen Defizite des generellen Ansatzes. Hier müssten genaue Kriterien entwickelt werden, wann eine Zuordnung als falsch bzw. unrealistisch zu erachten ist. Bisher ist die Feststellung einer falschen Zuordnung eher eine Vermutung.

Unrealistisch hohe Werte sind ein Indiz für eine falsche Zuordnung, reichen jedoch nicht zur eindeutigen Identifikation.

Der Vergleich beider Algorithmen zeigt, dass die Abweichungen der Mediane überraschend gering ist, was auf eine prinzipiell ähnliche Funktionsweise hinweist. Beide Algorithmen haben jedoch Defizite in der Zuordnung. Dabei scheint der Algorithmus 1 auch Schwächen bei geringen Klientenzahlen zu haben, denn die Maximalwerte liegen sehr hoch. Bei dem Algorithmus 2 sind bei großen Klientenzahlen die Maximalwerte nach oben hin deutlich größer als bei Algorithmus 1 – die falschen Zuordnungen führen zu deutlich größeren Werten. Die Schwächen beider Algorithmen sind also in unterschiedlichen Bereichen der Skalierung der DVE stärker bzw. schwächer ausgeprägt.

5.7 Überlegungen zur Lösung der auftretenden Probleme

Das Hauptproblem des verfolgten Ansatzes ist, dass die Zuordnung nicht exakt erfolgt. Eines der Symptome ist, dass die gemessenen Werte für die paarweise Latenz unrealistisch groß werden. Im Folgenden sollen Möglichkeiten diskutiert werden, um die in der Auswertung der Ergebnisse aufgefallenen Defizite des Vorgehens anzugehen.

5.7.1 Verwendung von Schwellwerten

Eine Möglichkeit, den erhobenen statistischen Werte zu mehr Aussagekraft zu verhelfen, wäre, ab bzw. bis zu bestimmten Schwellwerten Zuordnungen zu verwerfen. So könnte die Verwendung von als unrealistisch erachteten Werten verhindert werden. Nachteil dieses Vorgehens wäre es, dass bei zu großzügigen Schwellwerten weiterhin falsche Zuordnungen die statistischen Werte verfälschen. Bei zu enger Festlegung der Schwellwerte könnten richtige Zuordnungen ausgeschlossen werden. Dies wäre z.B. vorstellbar, wenn die DVE unter zu hoher Last sehr langsam reagieren würde.

5.7.2 Verwendung eines Konsistenzmodells

Eine Lösung für das grundlegende Problem der nicht exakten Zuordnung könnte sein, dass der Zuordnung ein Konsistenzmodell zugrundelegt wird. RDS folgt dem Modell der sequentiellen Konsistenz. D.h. dass die Operationen von allen Prozessen in der gleichen Reihenfolge gesehen werden. RDS erreicht dies, indem ungültige Zustände beim Klienten überschrieben werden. Der hier vorgestellte Ansatz versucht jedoch, anhand der Kommunikation zweier Klienten mit dem Server die Übertragung der Zustandsänderung von einem Klienten an den anderen zu rekonstruieren. Dass bestimmte Zustandsänderungen verworfen werden, wird in der Kommunikation mit

dem Server nicht durch eine bestimmte Nachricht kenntlich gemacht. Dieses Problem lässt sich durch das Konsistenzmodell von RDS nicht umgehen. Es ist datenzentriert und damit an dem Datenstand des Servers orientiert. Darüber, ob bestimmte Nachrichten verworfen werden, sagt es nichts aus.

5.7.3 Messen innerhalb der Software

Im Abschnitt 4.2 wurde diskutiert, die Klienten-Software der DVE so zu manipulieren, dass die Zustandsänderungen in der Software selbst und nicht an der Netzwerkschnittstelle abgefangen werden. Damit ist aber das Grundproblem der nicht präzisen Zuordnung von Zustandsänderungen nicht behoben. Denn auch hier stehen nur die Informationen zur Verfügung, die über die Nachrichten übermittelt wurden.

5.7.4 Manipulation der Software und des Protokolls

Ein weiterer white box Ansatz scheint vielversprechender. Das Protokoll der Nachrichten müsste so verändert werden, dass mit jeder MOVE-Nachricht ein aktueller Zeitstempel des Klienten geliefert wird. Dieser Zeitstempel müsste dann in dem data store für die Bewegungsinformationen mitgespeichert werden und den MOVEOBJECT-Nachrichten hinzugefügt werden.

Der Vorteil ist offensichtlich: Die Zuordnung aus Zeitstempel und Klienten-ID ist eindeutig. Die Zuordnungen der Nachrichten könnten fortan exakt erfolgen. Ein Nachteil wäre der enorme Entwicklungsaufwand. Zum einen müsste das Protokoll angepasst werden. Dies bedeutet eine Anpassung der Kommunikationseinheiten sowohl auf der Seite der Klienten als auch der des Servers. Die Server-Software müsste zudem umfangreicher angepasst werden, denn das Durchreichen des Nachrichtenstempels hat von der Kommunikationseinheit hin zum data store und wieder zurück durch mehrere Schichten der RDS Architektur zu erfolgen (zu den Details der Architektur siehe Abbildung 4.1). Sollten andere Zustandsänderungen neben der Bewegung erfasst werden, müssten diese Anpassungen auch für diese Nachrichtentypen und die damit zusammenhängenden gespeicherten Daten erfolgen. Außerdem ist zu berücksichtigen, dass durch die zusätzlich gesendeten Daten in de Nachrichten die Größe der Nachrichten zunimmt, was sich ebenfalls auf die Leistung auswirkt.

Trotz des hohen Aufwands wäre es jedoch ein Verfahren, dass bei der gegebenen Architektur der DVE eine exakte Messung ermöglichen würde. Der Schwachpunkt der Messung, die nicht präzise Zuordnung von Nachrichtenpaaren mittels eines Algorithmus, würde damit beseitigt werden können. Der Ansatz bliebe zwar im weitesten Sinne übertragbar, ein Großteil der verwendeten Software jedoch nicht.

6 Fazit

In dieser Arbeit wurde ein besonderer Aspekt von verteilten virtuellen Umgebungen untersucht: deren Leistung. Dabei stand nicht die Leistung selbst im Mittelpunkt, sondern das Messen eben dieser. Aufbauend auf einer Erhebung darüber, welche Kriterien für die Leistung von DVEs in der Literatur verwendet werden, wurde beispielhaft anhand eines Leistungskriteriums und einer DVE ein Verfahren zum Messen der Leistung einer DVE entwickelt. Dabei war Ziel dieses Vorgehens nicht, ein möglichst korrektes Verfahren zu entwickeln, sondern die Probleme des Messens zu verdeutlichen.

Die Untersuchung der Leistungskriterien hat veranschaulicht, dass viele unterschiedliche Kriterien existieren. Es bleibt jedoch häufig unklar, wie diese genau definiert sind und erhoben werden. Dies ist vor allem dem Umstand geschuldet, dass häufig Verfahren zur Verbesserung der Skalierbarkeit von DVEs im Mittelpunkt stehen und das Messen der Leistung nur ein Mittel zum Veranschaulichen der Erfolge des eigenen Verfahrens darstellt.

Der Überblick über verschiedene Kriterien hat dabei geholfen, die paarweisen Latenz als geeignetes Kriterium zu identifizieren. Es betrachtet die Zeit, die eine DVE braucht, um die Zustandsänderung eines Klienten bei einem anderen bekannt zu machen. Dadurch steht das Aufrechterhalten der Illusion einer kohärenten Welt für die Klienten, ein Hauptziel einer jeden DVE, im Mittelpunkt des Kriteriums.

Die paarweise Latenz wurde anschließend verwendet, um ein Modell zum Messen der Leistung der DVE Timadorus zu entwickeln. Der gewählte Ansatz verfolgte die Ziele, die Leistung ausgehend von der Klienten-Seite zu messen und dabei möglichst nicht in die Software der DVE einzugreifen.

Eine Analyse des state meldings der DVE ergab, dass eine exakte Erhebung der paarweisen Latenz nicht möglich war. Deswegen wurden zwei Algorithmen entworfen, um eine Annäherung an die paarweise Latenz zu ermöglichen. Es zeigte sich somit, dass die Schwierigkeiten, das Kriterium der paarweisen Latenz zu erheben, durch das Protokoll und die Architektur der DVE bestimmt sind.

Die Evaluation des Modells im Versuch verdeutlichte dieses. Die bei der Diskussion des Ansatzes und der entworfenen Algorithmen angesprochenen Probleme wurden in der Praxis bestätigt.

Zwar konnte der Rückgang der Leistung mit der Skalierung der Anwendung nachgezeichnet werden. Eine exakte Messung ließ sich aufgrund des momentan Protokolls jedoch nicht verwirklichen. Trotzdem wurde deutlich, was an dem Protokoll und der Software geändert werden müsste, um solch einen Messung der paarweisen Latenz durchzuführen, die weiterhin die Klienten-Seite als Ausgangspunkt hat.

Die Uneinheitlichkeit hinsichtlich der Kriterien zur Bewertung der Leistung der DVEs und die Unklarheit über die verwendeten Verfahren war bei der Literaturrecherche aufgefallen. Die in der Arbeit festgestellte Abhändigkeit des Verfahrens zum Messen von der Architektur und dem Protokoll der DVE, deutet auf die Ursachen dieser Vielfalt an Kriterien hin. Je nach DVE scheint ein anderes Kriterium besser geeignet.

Es lässt sich fragen, ob die Entwickler von DVEs bei dem Entwurf ihrer Anwendungen die Möglichkeit zur Messung berücksichtigen sollten. Hierzu wären weitere Untersuchungen notwendig, um den Aufwand der Integration von Messansätzen in bestehenden DVEs zu erfassen.

Abschließend lässt sich festhalten, dass die Architektur- und Protokollabhängigkeit möglicher Verfahren zum Messen der Leistung von DVEs dazu führen, dass sich nur schwer verallgemeinerbare und übertragbare Verfahren entwickeln lassen.

Literaturverzeichnis

Ahmad, L., Zhang, M., and Boukerche, A. (2008). A scalable adaptive time synchronization protocol for Large Scale Distributed Collaborative Simulation Environment. In *Haptic Audio visual Environments and Games, 2008. HAVE 2008. IEEE International Workshop on*, pages 42–47.

Ahmed, D. T. and Shirmohammadi, S. (2011). Improving online gaming experience using location awareness and interaction details. *Multimedia Tools and Applications*, pages 1–18.

Barri, I., Giné, F., and Roig, C. (2010). A Scalable Hybrid P2P System for MMOFPS. In *Proceedings of the 2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, PDP '10, pages 341–347, Washington, DC, USA. IEEE Computer Society.

Bharambe, A., Douceur, J., Lorch, J., Moscibroda, T., Pang, J., Seshan, S., and Zhuang, X. (2008). Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. *ACM SIGCOMM Computer Communication Review*, 38(4):389–400.

Bouillot, N. and Gressier-Soudan, E. (2004). Consistency models for distributed interactive multimedia applications. *SIGOPS Oper. Syst. Rev.*, 38(4):20–32.

Fritsch, T., Ritter, H., and Schiller, J. (2005). The effect of latency and network limitations on MMORPGs: a field study of everquest2. In *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, pages 1–9. ACM.

Gupta, N., Demers, A., Gehrke, J., Unterbrunner, P., and White, W. (2009). Scalability for Virtual Worlds. *2009 IEEE 25th International Conference on Data Engineering*, pages 1311–1314.

Jalote, P. (2005). *An integrated approach to software engineering (3. ed.)*. Texts in computer science. Springer.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565.

Liu, H., Bowman, M., and Chang, F. (2012). Survey of state melding in virtual worlds. *ACM Computing Surveys*, 44(4):1–25.

Miller, J. L. (2011). Distributed virtual environment scalability and security. Technical Report UCAM-CL-TR-809, University of Cambridge, Computer Laboratory.

Morillo, P., Ordufia, J. M., and Duato, J. (2006). A scalable synchronization technique for distributed virtual environments based on networked-server architectures. In *Parallel Processing Workshops, 2006. ICPP 2006 Workshops. 2006 International Conference on*, pages 8 pp.–81.

Najaran, M. T. and Krasic, C. (2010). Scaling online games with adaptive interest management in the cloud. In *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, pages 1–6.

Schöning, U. (2008). *Ideen der Informatik: Grundlegende Modelle und Konzepte der Theoretischen Informatik*. Oldenbourg.

Styn, H. V. (2011). tcpdump fu. *Linux J.*, 2011(210).

Tanenbaum, A. S. and Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Waldo, J. (2008). Scaling in games & virtual worlds. *Queue*, 6(7):10–16.

Willatowski, J. O. (2013). Maschinelles Lernen in kontinuierlichen Systemen am Beispiel von Nicht-Spieler-Charakteren in Multiplayer-Online-Games. Bachelorarbeit, HAW Hamburg.

Glossar

CSV	Comma-separated values, Seite 35
DVE	Verteilte virtuelle Umgebung. Englisch: Distributed virtual environment, Seite 1
JVM	Java Virtual Machine, Seite 35
MMOG	Massively multiplayer online game, Seite 4
MMORPG	Massively multiplayer online role-playing game, Seite 2
RDS	Red Dwarf Server, Seite 20
RPG	Role-playing game, Seite 4
RTS	Real-time strategy, Seite 4

Anhang

1 Tabellen

Klienten-Nachrichten	**Server-Nachrichten**
ADDOBJECT	ATTACK
ATTACKFAIL	READY
ATTACKOBJECT	LEAVING
CHANGEOBJECT	MOVE
CONSUMEFAIL	CONSUME
CONSUMEOBJECT	INSPECT
DROPOBJECT	PERCEIVE
ENTERING	GETINFO
ERROR	GETATTRIBUTE
GOODBYE	TEACH
OBJECTATTRIBUTE	CASTTARGET
OBJECTINFO	CASTAREA
OBJECTNOTREACHABLE	KICK
STARTGAME	BAN
MOVEOBJECT	SELECTCHARACTERCLASS
NOTVISIBLE	GETWORLDINFO
REMOVEOBJECT	
TEACHRESULT	
WORLDINFO	
GEODATA	
CHANGECHARACTERCLASS	
MOVETO	

Tabelle 1: Server- und Klienten-Nachrichten in Timadorus

Nr.	Zeitstempel	Typ	Inhalt
0-2	-	-	Nachrichten zur Kontaktaufnahme (ausgelassen)
3	1374403281285	client message	ENTERING - {1602}, {0.000000,0.000000,0.000000}
4 – 28	-	-	Testnachrichten vor dem Start sowie Terraininformationen (ausgelassen)
29	1374403282702	server message	READY - {}
30	1374403282708	client message	STARTGAME - {}
31	1374403282739	server message	MOVE - {0.05,0.0,0.0, 1.0,0.0,0.0,0.0}
32	1374403283936	client message	ADDOBJECT - {[1602, 1, 782, 862, 863, 861, 823, 822, 821] objectIds}
33	1374403283975	client message	MOVEOBJECT - {862, true, 0.700000,0.000000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
34	1374403284015	client message	MOVEOBJECT - {863, true, 0.700000,0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
35	1374403284025	client message	MOVEOBJECT - {821, true, 0.000000,-0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
36	1374403284243	client message	ADDOBJECT - {[1602, 1, 782, 862, 863, 861, 823, 822, 821] objectIds}
37	1374403284288	client message	MOVEOBJECT - {1602, true, 0.386750,0.000000,0.000000, 0.05,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
38	1374403284293	client message	MOVEOBJECT - {821, true, 0.000000,-0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
39	1374403284295	server message	MOVE - {-0.05,0.0,0.0, 1.0,0.0,0.0,0.0}
40	1374403284373	client message	ADDOBJECT - {[1602, 1, 782, 862, 863, 861, 823, 822, 821] objectIds}
41	1374403284440	client message	MOVEOBJECT - {1602, true, 0.461400,0.000000,0.000000, -0.05,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
42	1374403284441	client message	MOVEOBJECT - {823, true, 0.000000,0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
43	1374403284442	server message	MOVE - {-0.049999999999999996,0.0,0.0, 1.0,0.0,0.0,0.0}
44	1374403284483	client message	MOVEOBJECT - {822, true, 0.000000,0.000000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
45	1374403284523	client message	MOVEOBJECT - {821, true, 0.000000,-0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
46	1374403284552	client message	ADDOBJECT - {[1602, 1, 782, 862, 863, 861, 823, 822, 821] objectIds}
47	1374403284605	client message	MOVEOBJECT - {1602, true, 0.458450,0.000000,0.000000, -0.049999999999999996,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
48	1374403284607	client message	MOVEOBJECT - {863, true, 0.700000,0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
49	1374403284654	client message	MOVEOBJECT - {861, true, 0.700000,-0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
50	1374403284656	client message	MOVEOBJECT - {821, true, 0.000000,-0.700000,0.000000, 0.0,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}
51	1374403284751	client message	ADDOBJECT - {[1602, 1, 782, 862, 863, 861, 823, 822, 821] objectIds}
52	1374403284788	client message	MOVEOBJECT - {1602, true, 0.449350,0.000000,0.000000, -0.049999999999999996,0.0,0.0, 1.0,0.0,0.0,0.0, 1.0,0.0,0.0,0.0}

Tabelle 2: Initiale Kommunikation zwischen einem Server und einem Klienten in Timadorus

2 Java-Code der Algorithmen

```
List<Pair<MoveServerMessage, MoveClientMessage>>
    calculateMessagePairs(
        List<IMoveMessage> messages) {
  List<Pair<MoveServerMessage, MoveClientMessage>> result = new
      ArrayList<Pair<MoveServerMessage, MoveClientMessage>>();
  int serverIndex = 0;
  Movement oldMovement = null;
  while (serverIndex < messages.size()) {
    IMoveMessage serverMessage = messages.get(serverIndex);
    if (serverMessage.isServerMessage()
        && !serverMessage.getMovement().equals(oldMovement)) {
      Movement newMovement = serverMessage.getMovement();
      int clientIndex = serverIndex + 1;
      while (clientIndex < messages.size()) {
        IMoveMessage clientMessage = messages.get(clientIndex);
        if (clientMessage.isClientMessage()
            && clientMessage.hasMovement(newMovement)) {
          result.add(Pair.create(
              (MoveServerMessage) serverMessage,
              (MoveClientMessage) clientMessage));
          oldMovement = newMovement;
          break;
        }
        clientIndex++;
      }
    }
    serverIndex++;
  }
  return result;
}
```

Abbildung 1: Java-Code für Algorithmus 1

```
List<Pair<MoveServerMessage, MoveClientMessage>>
    calculateMessagePairs(
        List<IMoveMessage> messages) {
  List<Pair<MoveServerMessage, MoveClientMessage>> result = new
      ArrayList<Pair<MoveServerMessage, MoveClientMessage>>();
  int clientIndex = 0;
  Movement oldMovement = null;
  while (clientIndex < messages.size()) {
    IMoveMessage clientMessage = messages.get(clientIndex);
    if (clientMessage.isClientMessage()
        && !clientMessage.getMovement().equals(oldMovement)) {
      Movement newMovement = clientMessage.getMovement();
      int serverIndex = clientIndex - 1;
      while (serverIndex >= 0) {
        IMoveMessage serverMessage = messages.get(serverIndex);
        if (serverMessage.isServerMessage()
            && serverMessage.hasMovement(newMovement)) {
          result.add(Pair.create(
              (MoveServerMessage) serverMessage,
              (MoveClientMessage) clientMessage));
          oldMovement = newMovement;
          break;
        }
        serverIndex--;
      }
    }
    clientIndex++;
  }
  return result;
}
```

Abbildung 2: Java-Code für Algorithmus 2

3 Unit-Tests für Algorithmus 1

Leere Eingabe

Eingabe:

leer

Ausgabe:

leer

Keine MOVEOBJECT-Nachricht in Eingabe

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVE	A

Ausgabe:

leer

Keine MOVE-Nachricht in Eingabe

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVEOBJECT	A
2	1	MOVEOBJECT	A

Ausgabe:

leer

Kein inhaltlich passendes Paar

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	B
3	2	MOVEOBJECT	C

Ausgabe:

leer

Potentielles Paar, aber in falscher Reihenfolge

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVEOBJECT	A
2	1	MOVE	A
3	2	MOVEOBJECT	C

Ausgabe:

leer

Einfache Zuordnung

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	B
3	2	MOVE	C
4	3	MOVEOBJECT	A

Ausgabe:

{ (1, 4) }

Eine Zuordnung, zweite wird aufgrund inhaltlicher Gleichheit ignoriert

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	A
3	2	MOVE	A
4	3	MOVEOBJECT	A

Ausgabe:

{ (1, 2) }

Zwei Zuordnungen

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	C
3	2	MOVEOBJECT	A
4	3	MOVE	B
5	4	MOVEOBJECT	B

Ausgabe:

{ (1, 3), (4, 5) }

Zwei Zuordnungen mit dem zwischenzeitlichen Ignorieren einer Nachricht aufgrund gleichen Inhalts

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	A
3	2	MOVE	A
4	3	MOVE	B
5	4	MOVEOBJECT	B
6	4	MOVEOBJECT	B

Ausgabe:

{ (1, 2), (4, 5) }

4 Unit-Tests für Algorithmus 2

Leere Eingabe

Eingabe:

leer

Ausgabe:

leer

Keine MOVEOBJECT-Nachricht in Eingabe

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVE	A

Ausgabe:

leer

Keine MOVE-Nachricht in Eingabe

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVEOBJECT	A
2	1	MOVEOBJECT	A

Ausgabe:

leer

Kein inhaltlich passendes Paar

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	B
3	2	MOVEOBJECT	C

Ausgabe:

leer

Potentielles Paar, aber in falscher Reihenfolge

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVEOBJECT	A
2	1	MOVE	A
3	2	MOVEOBJECT	C

Ausgabe:

leer

Einfache Zuordnung

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	B
3	2	MOVE	C
4	3	MOVEOBJECT	A

Ausgabe:

$\{ (1, 4) \}$

Eine Zuordnung, zweite wird aufgrund inhaltlicher Gleichheit ignoriert

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	A
3	2	MOVE	A
4	3	MOVEOBJECT	A

Ausgabe:

{ (1, 2) }

Zwei Zuordnungen

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	C
3	2	MOVEOBJECT	A
4	3	MOVE	B
5	4	MOVEOBJECT	B

Ausgabe:

{ (1, 3), (4, 5) }

Zwei Zuordnungen mit dem zwischenzeitlichen Ignorieren einer Nachricht aufgrund gleichen Inhalts

Eingabe:

Nr.	Zeitpunkt	Typ	Bewegung
1	0	MOVE	A
2	1	MOVEOBJECT	A
3	2	MOVE	A
4	3	MOVE	B
5	4	MOVEOBJECT	A
6	4	MOVEOBJECT	B

Ausgabe:

{ (1, 2), (4, 6) }

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 30. August 2013 Oliver Behncke