Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Frederik Klauß

Verteilte Transaktionen auf Basis von asynchronen Nachrichten in einem Eventual Consistency Persistenzsystem

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Frederik Klauß

Verteilte Transaktionen auf Basis von asynchronen Nachrichten in einem Eventual Consistency Persistenzsystem

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. Martin Hübner
Zweitgutachter: M.Sc. Lutz Behnke

Eingereicht am: 21. April 2015

Frederik Klauß

Thema der Arbeit
Verteilte Transaktionen auf Basis von asynchronen Nachrichten in einem Eventual Consistency Persistenzsystem

Stichworte
Transaktionen, Konsistenz, MMO, Datenbanken, CAP, ACID

Kurzzusammenfassung
In dieser Arbeit wird ein Transaktionssystem für komplexe, verteilte Transaktionen für das Timadorus Projekt der HAW Hamburg entwickelt. Dabei werden zunächst die nötigen Techniken erklärt und ein allgemeiner Entwurf erstellt. Zum Schluss wird eine Analyse der Verwendbarkeit für das Timadorus Projekt erstellt, eine Erklärung zu Transaktionen in verteilten System sowie eine Aussicht auf Alternativlösungen für zukünftige Implementationen für das Handling von Transaktionen im Timadorus Projekt gegeben.

Frederik Klauß

Title of the paper
Distributed transactions based on asynchrone messages in an eventual consistency persistence system

Keywords
Transactions, consistency, massive multiplayer, database, CAP, ACID

Abstract
This thesis discribes the development of a transaction system for complex, distributed transactions designed for the Timadorus project. The principles that have to be known for this will be explained as well as an general aproach to the problem. At the end the result will be rated in usability for the Project and some prospects given regarding implementations for the Timadorus project.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Kontext der Arbeit . . . 1
- 1.3 Zielsetzung . . . 3
- 1.4 Aufbau der Arbeit . . . 4
- 1.5 Problembeschreibung . . . 5
  - 1.5.1 Fachliche Problembeschreibung . . . 5
  - 1.5.2 Technische Problembeschreibung . . . 6
  - 1.5.3 Bisherige Umsetzung in Spielen . . . 7

2 Stand der Technik **8**
- 2.1 BASE - Eventual Consistency . . . 8
- 2.2 Transaktionen . . . 8
  - 2.2.1 Herkömmliche Transaktionen . . . 9
  - 2.2.2 Einfache verteilte Transaktionen . . . 9
    - 2.2.2.1 Ein Client mit vielen Servern . . . 9
    - 2.2.2.2 Ein Server mit vielen Clients . . . 10
    - 2.2.2.3 Koordinator in verteilten Transaktionen . . . 10

3 Anforderungen an die Lösung **12**
- 3.1 Fachliche Anforderungen . . . 12
  - 3.1.1 Anforderungen der Spieler . . . 12
  - 3.1.2 Anforderungen der Administratoren . . . 12
  - 3.1.3 Anforderungen der Programmierer . . . 13
- 3.2 Technische Anforderungen . . . 13

4 Konzept **15**
- 4.1 Verfahrensauswahl . . . 15
  - 4.1.1 Transaktionsverfahren . . . 15
- 4.2 Entwurf . . . 16
  - 4.2.1 Beschreibung Komplexe Transaktionen . . . 16
  - 4.2.2 Komponentenentwurf . . . 18
  - 4.2.3 Nachrichtenprotokoll . . . 19
    - 4.2.3.1 Beschreibung der Nachrichten . . . 19
    - 4.2.3.2 Kommunikationsdiagramm . . . 21
  - 4.2.4 Implementierungsaspekte . . . 24

5 Analyse der Verwendbarkeit **26**
5.1 Skalierbarkeit . . . 27
5.2 Technische Korrektheit . . . 27
5.3 Fachliche Korrektheit . . . 28
5.4 Erklärung durch CAP-Theorem . . . 30
5.5 Resultat Analyse . . . 30

6 Zusammenfassung **32**

7 Ausblick **33**
7.1 Zurückhalten der Sperren . . . 33
7.2 Auflösbare Sperren . . . 34
7.3 Durchlässige Sperren . . . 35
7.4 Fachliche Prüfungsmechanismen . . . 35
7.5 Nicht transaktionale Lösungen . . . 36
7.6 Weitere Ausblicke . . . 37

1 Einleitung

1.1 Motivation

Motivation zu diesem Thema ist der Traum von einem Spiel, das anders als bisherige Spiele, keinerlei Einschränkungen im Bereich der Spieleranzahl und Größe der Spielwelt unterliegt und dabei ebenso wenig technisch begründete Störungen, wie offensichtliche Menüs und Verzögerungen im Spielgeschehen durch Nachrichtenaustausch vom Client zum Server hat, die den Spielfluss negativ beeinflussen.

In der aktuellen Zeit werden viele Online Spiele von *Cheatern* (Personen die Schwachstellen in Programmen ausnutzen um sich einen illegalen/unfairen Vorteil zu verschaffen) untergraben, die den meisten normalen Spielern, gerade in kompetitiven Spielen, das Spielerlebnis zerstören. Dies entspricht nicht dem durch die Spieler und Entwickler des Systems angestrebten Weltbild von Fairness und verhindert die erfolgreiche Umsetzung eines Spiels, da viele legitime Spieler von solchen Personen nicht zusammen spielen wollen und eine Vergrößerung sowie Weiterentwicklungen des Spiels aus mangelndem Interesse nicht mehr realisiert werden. Technische Verbesserungen wie etwa Transaktionen können dafür sorgen, dass Cheater es schwieriger haben Schwachstellen zu finden oder sich anderweitig einen Vorteil zu verschaffen.

Transaktionen werden außerdem angewendet um Fehlern im System vorzubeugen, die dem Spielfluss und damit dem Spielerlebnis schaden.

Diese Arbeit ist ein kleiner Beitrag dazu dieses System und damit die später beherbergte Welt fehlerresistenter und vom Spielerlebnis her befriedigend zu machen.

1.2 Kontext der Arbeit

Diese Arbeit steht im Kontext des Universitätsprojektes *Der Weg nach Timadorus* unter der Leitung von Lutz Behnke an der HAW-Hamburg.

Bei diesem Projekt handelt es sich um ein Massive Multiplayer Online *Rollen Spiel* (engl. MMORPG), welches sich das Ziel der möglichst realistischen Darstellung der echten Welt gesetzt hat. Dabei soll neben seiner Persistenz auch eine unendlich hohe Skalierbarkeit der Anzahl der möglichen Server als auch der Clients gewährleistet werden. Im Fall der Maximalverteilung kann es dabei zu einer 1:1-Beziehungen zwischen der Anzahl der Server und der Clients kommen. Behnke.

Das System des Spieles besteht aus drei Hauptkomponenten:

Spiel-Client *Theoderich*: Der Client ist der Teil des Systems, welches der Spieler direkt steuert. Er verarbeitet Eingaben des Spielers und zeigt die Spielwelt an. Derzeit gibt es eine Java-Applikation, eine Umsetzung für Mobilgeräte unter Android-Betriebssystemen ist jedoch zusätzlich geplant.

Spiel-Server *Timadorus Game Server*: Der Server (kurz: TGS) ist das Bindeglied zwischen dem QuP-Server und dem Spiele-Client. Er verarbeitet in erster Instanz Anfragen durch den Spieler und beinhaltet die Spiellogik sowie einige Kopien der Objekte im Spiel zum schnelleren Zugriff auf diese. Zur Kommunikation verwendet dieser zwei Schnittstellen-Komponenten: zum einen zu den angeschlossenen Spiele-Clients und zum anderen zu den *QuP-Servern*. Clients können sich bei diesen Servern auf dort liegende Objekte als Observer registrieren lassen und bekommen so Änderungen an diesen mitgeteilt. Ebenso werden alle Anfragen der Clients für neue Objekte an die TGS gestellt, welche wiederum bei den QuP-Servern die Informationen abrufen. Behnke (2014)

QuP-Server: Bei dieser Backend Server Node handelt es sich um die verteilte Datenbank des Timadorus Systems. Hier werden alle Objekte in einem speziell dafür ausgelegten, verteilten Octree gespeichert und an die TGS auf Anfrage ausgeliefert. TGS, die Objekte erfragen, registrieren sich, wie die Clients bei den TGS, als Observer auf den QuP-Serven um dadurch über jede Änderung informiert zu werden.

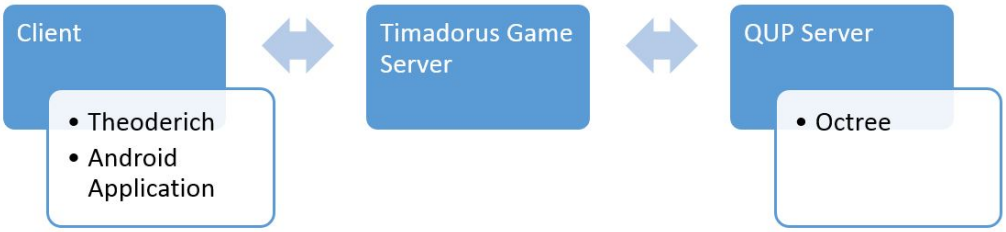

Abbildung 1.1: Aufbau des Timadorus Systems

Des Weiteren gibt es in diesem System noch weitere außenstehende Kontrollelemente wie den Loadbalancer oder den Anmeldeserver für Clients. Diese sind jedoch im weiteren Verlauf dieser Arbeit nicht wichtig.

1.3 Zielsetzung

Im Kern dieser Arbeit steht die Konzeptionierung und der Entwurf eines Transaktions Systems für die Umsetzung im *Timadorus Projekts* der HAW-Hamburg.
Anschließend wird analysiert, unter welchen Voraussetzungen man dies in dieses Projekt einbauen kann.

Ziel ist die Entwicklung eines parallelen Funktionszweiges für das *Timadorus* Projekt. Ein „Paralleler Funktionszweig“meint hier, dass es zu den alten Entwicklungen zusätzliche Programmteile geben soll, welche die neuen Möglichkeiten der Transaktion umsetzen.

Zukünftige Entwickler an diesem Projekt sollen die Möglichkeit haben mit möglichst wenig Aufwand voneinander abhängige Aktionen zu erstellen und sie korrekt, nach den Regeln der Transaktion, auszuführen.
Bereits bestehende Implementierungen ohne Transaktionen sollen dabei nicht verändert werden. Ebenso soll es weiterhin möglich sein Aktionen auf regulärem Weg, d.h. ohne Transaktionen, durchzuführen.

Der Spielfluss, welcher ausschlaggebend für das Spieler-Erlebnis ist, soll dabei nicht gestört werden.

1.4 Aufbau der Arbeit

Zunächst werden die Probleme angesprochen, welche diese Arbeit zu lösen versucht, diese sind unterteilt in die technischen und die fachlichen Probleme.
Im Anschluss wird ein Überblick über die bisher angewandten Verfahren für Transaktionen gegeben. Hier werden auch kurz Verfahren angesprochen, die in dieser Arbeit keine Anwendung finden werden, jedoch zum Verständnis beitragen und später für einen Vergleich der Methoden angeführt werden.
Nach dieser Zusammenfassung kommt der Hauptteil der Arbeit, welcher sich zunächst mit den Anforderungen an die Lösung befasst. Dieser ist ebenfalls nach fachlichen und technischen Anforderungen unterteilt, wobei die Anforderung aus fachlicher Sicht beschreibt, welche für den Spieler wahrnehmbare Ergebnisse, wie zum Beispiel geringere Ladezeiten oder kein Kontakt zu Fehlern, zu erzielen sind. Die technische Seite, die bisher im Projekt verwendeten Techniken und die zu erstellende Art des Codes und deren weitere Verwendung innerhalb des *Timadorus Projekts* beschreibt.

Zu Beginn des Entwurfes der Arbeit steht eine Abwägung der Vor- und Nachteile, welche bereits vorhandenen Techniken verwendet werden sollen, beziehungsweise welche Techniken neu entwickelt werden müssen. Im Anschluss wird das Programm mit den geänderten Nachrichten, Protokollen und benötigten Komponenten entworfen und einer Prüfung unterzogen ob dieses den gestellten Anforderungen entspricht oder ob es Teilaspekte nicht erfüllt, die jedoch essentiell wichtig sind.

Da diese Arbeit auf das *Timadorus* Projekt ausgelegt ist und im Laufe des Entwurfs klar wurde, dass es nicht möglich ist mit der entworfenen Technik ein sicheres und zu dem Projekt passendes Transaktionssystem zu entwerfen, wird im Kapitel nach dem Entwurf die genaue Problematik erläutert und erklärt warum dieser Entwurf nicht verwendet werden kann.

Den Schluss der Arbeit bildet eine Zusammenfassung der Ergebnisse dieser Arbeit und ein Ausblick auf die Möglichkeiten, die sich durch diese Arbeit eröffnen sowie die Beschreibung

einiger alternativen Lösungen mit denen man Transaktionen weniger sichtbar machen beziehungsweise umgehen kann.

1.5 Problembeschreibung

1.5.1 Fachliche Problembeschreibung

In einer Spielwelt wie der von *Timadorus*, gleichwohl auch ähnlichen virtuellen Welten sowie der realen Welt, gibt es Aktionen, die im Zusammenhang mit anderen Aktionen stehen. Das heißt eine Aktion A kann nicht geschehen, ohne dass eine Aktion B gleichzeitig ausgeführt wird.

Ein einfaches Beispiel hierfür ist der Handel: Zwei Spieler wollen einen Tausch eingehen. Beide Personen legen das zu tauschende Objekt auf einen Tisch und greifen zum jeweils erhandelten Gegenstand. Beide Personen haben jetzt ihren eigenen Gegenstand verloren und dafür einen neuen Gegenstand erhalten.

Diese Abhängigkeiten sollen in das Spiel *Timadorus* übersetzt werden, sodass die Spieler ganzheitliche angenehme Spielerlebnisse haben die, unter Vereinbarung mit den Gesetzen der Spielwelt, realistisch sind und allgemein den Erwartungen der Spieler an eine realistische Welt entsprechen.

Des weiteren soll durch die Entwicklung der Transaktionen verhindert werden, dass es zu Betrügereien aufgrund von Ausnutzungen der Nichtzusammengehörigkeit von Aktionen kommt. Ein Beispiel hierfür in der Vergangenheit wäre ein Fehler im Gameboy-Spiel "Pokémon Rot/Blau". Hier ist es möglich durch das Fehlen von Transaktionen Objekte zu verdoppeln.

> „Die Möglichkeit, Pokémon zu klonen, findet vor allem bei legendären oder hoch trainierten Pokémon besonderen Reiz unter den Spielern. Dabei verbindet man zwei Game Boys mit je einem Pokémon-Spiel mit einem Linkkabel. [...] Kurz vor Abschluss des Tauschvorgangs schaltet derjenige, der das zu klonende Pokémon besaß, den Gameboy ab, während der andere wartete, bis auf seinem Bildschirm der Tausch als vollzogen bekanntgegeben wurde. Dadurch wird das zu klonende Pokémon verdoppelt, während das andere Pokémon verschwindet. Dieser Bug war in allen Editionen der 1. Generation vorhanden.“PokeBugsErsteGeneration2014

Ebenso störend sind Objekte, die ohne den Abschluss der kompletten Aktionskette verschwinden. Beispiel hierfür aus dem echten Leben: Ein Ticketautomat, welcher zwar das Geld für das Ticket annimmt, dieses jedoch nie ausdruckt, weil ein Unterprozess abgestürzt ist.

Bei einem MMORPG kommt es stark darauf an, dass die Spielwelt nachvollziehbar bleibt und keiner der Teilnehmer durch Systemfehler benachteiligt wird. Dies wird im Allgemeinen unter dem Begriff *Immersion* zusammengefasst und beschreibt damit, wie sehr sich ein Spieler in ein Spiel verlieren kann ohne, dass er aus dieser virtuellen Welt gerissen wird. Grau (2003)

Da diese Immersion durch das offensichtliche Auftreten von technischen Eingriffen, wie langen Ladezeiten oder Auftreten von Menüs und Tabellen, getrübt wird, sind alle Schutzmaßnahmen, wie auch Transaktionen, so zu verbergen, dass der Spieler sie nicht wahrnimmt.

Zusammenfassend sollen alle diese Probleme mit der Einführung eines Transaktionsmodells gelöst werden und damit der Supportaufwand verringert werden, sowie eine aus Inkonsistenz resultierende negative Wahrnehmung des Spieles verhindert werden.

1.5.2 Technische Problembeschreibung

Für die Umsetzung der komplexen Transaktionen in das *Timadorus*-System muss zunächst ein neues, übergreifendes Nachrichtenprotokoll definiert werden, auf dem später sämtliche transaktionsabhängige Programmierung aufsetzt.

Dadurch, dass es im bisherigen System (Stand 09.01.2015) keinerlei Ressourcenmanagement gibt, d.h. Sperrmechanismen für gleichzeitiges Zugreifen auf eine Ressource, muss dieses, zumindest in primitiver Form, entworfen und implementiert werden.
Der Ablauf der eigentlichen Transaktion muss ebenso entworfen und implementiert werden.

Bei allen Entwürfen und Umsetzungen ist darauf zu achten, dass es sich bei dem *Timadorus* System um ein unendlich verteiltes System handeln kann. Ebenso basiert es auf vollständig asynchronem Nachrichtenaustausch zwischen den Applikationen des Systems.

Bereits bestehende Programmteile dürfen durch die zusätzliche Implementierung der Transaktionen in keiner Weise behindert werden. Das heißt, bereits bestehende Funktionen dürfen

nicht verändert werden. Der Inhalt dieser Arbeit ist daher als eine Erweiterung und nicht als komplette Veränderung des bisherigen Codes zu verstehen.

1.5.3 Bisherige Umsetzung in Spielen

Wenn man sich mit der Welt der Computerspiele befasst und andere MMOs wie *„World of Warcraft“* oder *„Eve Online“*, um ein paar Beispiele zu nennen, betrachtet, stellt man fest, dass es bereits viele Systeme gibt, welche Transaktionsmechanismen benötigen und auch implementieren. Deren Umsetzungen bewegen sich jedoch meistens in stark gekapselten Bereichen des Spiels, wie dem Handeln, und fühlen sich aufgrund der gesonderten GUI eher wie technische Mechanismen an, was die Immersion zerstört und dem Spielfluss abträglich ist. Es ist daher mit Schwierigkeiten bei der Umsetzung zu rechnen. Einer generellen Umsetzung dieses Programmcodes sollte nichts im Weg stehen, eventuell aber einer spezialisierten Umsetzung für das *Timadorus* Projekt.

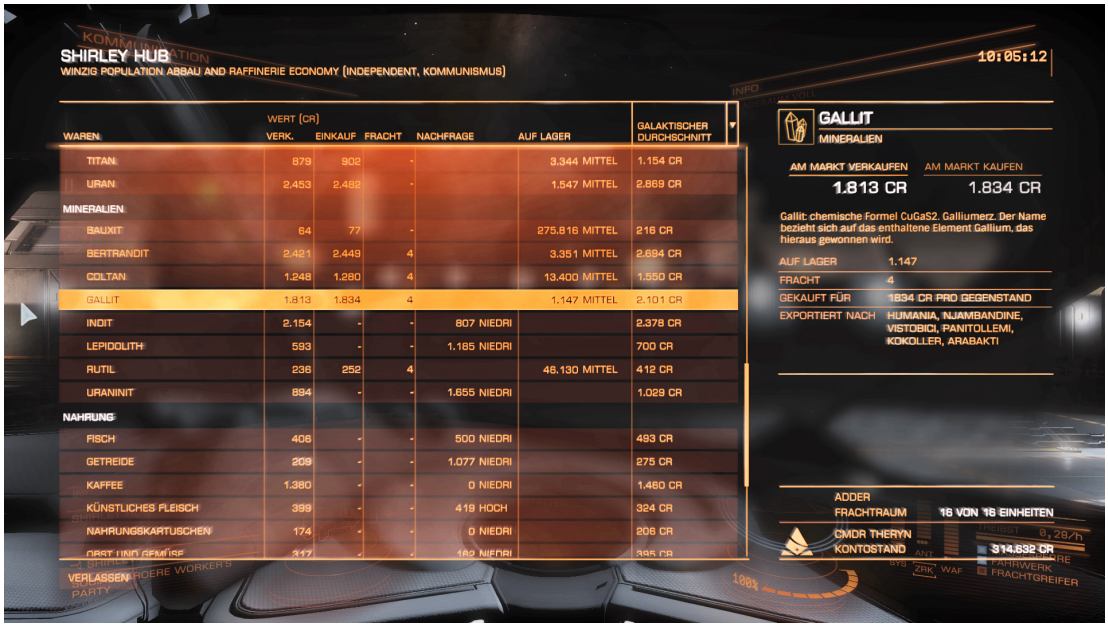

Abbildung 1.2: Handeln über Tabellen im Computerspiel "Elite-Dangerous“ ElitePcGames (2015)

2 Stand der Technik

2.1 BASE - Eventual Consistency

BASE *(Basically Available, Soft state, Eventual consistency)* basiert auf dem CAP-System von Eric Brewer Brewer (2000), das besagt, dass es von den drei Eigenschaften Konsistenz (Consistency), Verfügbarkeit (Availability) und Partitionierbarkeit (Tolerance to Network Partitions), nur zwei zur gleichen Zeit erfüllt sein können.

Allgemein kann man sagen, dass das BASE-Modell im Kontrast zum ACID (Atomicity, Consistency, Isolation und Durability) Model steht. ACID ist darauf ausgelegt, dass Konsistenz im Vordergrund steht und stellt daher sicher, dass Updates sofort auf dem Gesamtsystem ausgeführt werden.
BASE dagegen verfolgt eine eher freizügige Update-Politik im System, was zu möglichen Unterschieden an Dateninformationen auf den unterschiedlichen Servern führen kann. *Eventual Consistency* beruht auf der Annahme, dass in einem Datenbanksystem, welches BASE benutzt, alle Daten schlussendlich, zu einem Zeitpunkt X, konsistent sind.

Anders formuliert: *Eventual Consistency* braucht anstelle von sofortigen systemweiten Updates eine gesetzte Zeitspanne für eine vollständige Auflösung im System. Ein Programm muss entsprechend in der Lage sein auf Inkonsistenzen zu reagieren und muss Strategien entwickeln um diese fachlich richtig aufzulösen. Vogels (2009)

2.2 Transaktionen

Transaktionen sind eine Entwicklung mit der man sicherstellen kann, dass Änderungen an Objekten, die von einander abhängig sind, wie zum Beispiel bei Überweisungen, gekoppelt werden können. Vgl.: Weikum und Vossen (2001) oder Tanenbaum und Steen (2008)

2.2.1 Herkömmliche Transaktionen

Die herkömmliche Transaktion basiert auf der Annahme, dass es nur einen Server und nur einen Client gibt, die gleichzeitig mit den Daten der Transaktion arbeiten (1:1-Verhältnis). In diesem Verfahren vergibt der Server allen Aktionen eine gemeinsame Transaktionskennung, welche eindeutig über das gesamte System ist. Sollte es in einer Abfrage innerhalb dieser Transaktion zu einem Fehler kommen, werden alle Aktionen auf der Datenbank rückgängig gemacht, bzw. die Änderungen an den Objekten in der Datenbank nicht festgeschrieben.

2.2.2 Einfache verteilte Transaktionen

Bei den einfachen verteilten Transaktionen gibt es zwei unterschiedliche Szenarien. Funktional leisten sie das Selbe wie die herkömmliche Transaktion. Die unterschiedlichen Varianten sind zum einen eine Transaktion von vielen Clients auf einen Server (n:1-Verhältnis) und zum anderen eine Transaktion von einem Client auf Ressourcen, die auf mehreren Servern verteilt liegen.

2.2.2.1 Ein Client mit vielen Servern

Die am häufigsten anzutreffende Variante der verteilten Transaktion ist die, in der es einen Client gibt der seine Operationen auf mehreren verschiedenen Servern aufruft. Hierbei gilt es zu unterscheiden ob es sich bei der Transaktion um eine flache Transaktion handelt oder um eine verschachtelte Transaktion.

Bei einer flachen, verteilten Transaktion wird jede ihrer Anforderungen an den Server zunächst abgeschlossen, bevor die nächste Anforderung abgesetzt wird. Dies bedeutet, dass die flachen, verteilten Transaktionen maximal auf ein Objekt warten, sofern eine Sperrung dieses Objekts vorliegt.

Verschachtelte, verteilte Transaktionen hingegen können beliebig tief kaskadiert werden, denn jede der Transaktionen kann seine Abfragen in einzelne Unter-Transaktionen aufteilen. Diese zählen für sich wie vollwertige Transaktionen und können dementsprechend abermals ihre Anfragen weiter in Unter-Transaktionen aufteilen. Der Vorteil dieser Verschachtlung ist die Nebenläufigkeit der einzelnen untergeordneten Transaktionen.

An einem Beispiel: Wir haben eine Transaktion T die zwei Unter-Transaktionen T1 und T2 erstellt hat. T1 hat die Unter-Transaktionen T1.1 T1.2 und T2 hat die Unter-Transaktionen T2.1 und T2.2. T1 und T2 sind jetzt ebenso nebenläufig ausführbar wie T1.1, T1.2, T2.1 und T2.2. Auf diese Weise können, in diesem Beispiel, vier Abfragen gleichzeitig ausgeführt werden anstelle von nur einer.

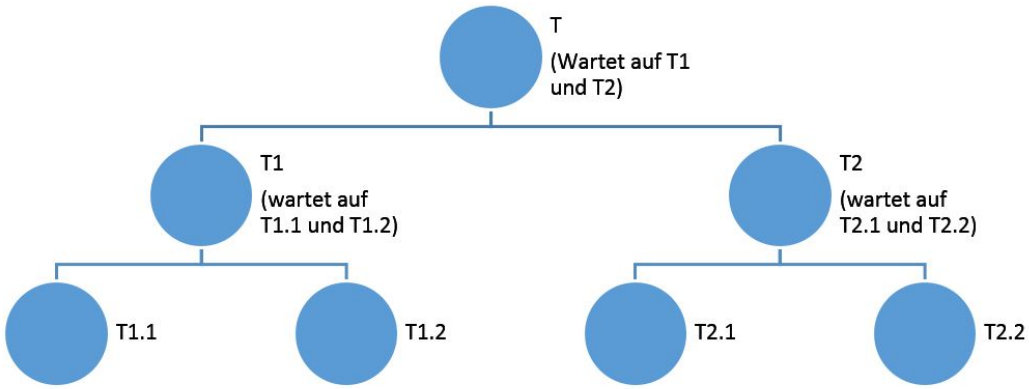

Abbildung 2.1: Verschachtelte Transaktionen

2.2.2.2 Ein Server mit vielen Clients

Die andere Variante der einfachen verteilten Transaktionen ist die, bei der es einen zentralen Server gibt und mehrere Clients die Objekte auf diesem verändern wollen, die voneinander abhängig sind. Wie bereits bei der 1 zu n Variante der einfachen verteilten Transaktionen, gibt es auch hier Unterscheidungen zwischen flachen und verschachtelten Transaktionen. Diese unterscheiden sich jedoch nicht von denen der 1 zu n Variante.

Der Unterschied ist, dass mehr als ein Client dem Server Anfragen schickt. Der Server an sich behandelt sie normal, demnach als eine sequentielle Abarbeitung der Befehle bei flachen Transaktionen oder er unterteilt sie in verschachtelte Transaktionen.

2.2.2.3 Koordinator in verteilten Transaktionen

In beiden Versionen der einfachen verteilten Transaktionen gibt es einen sogenannten Koordinator der Transaktion. Dieser kann sowohl ein externes Steuerungsprogramm sein als auch

direkt in den Server integriert sein. Seine Aufgabe bleibt dabei immer die selbe.

Der Koordinator ist dafür zuständig, dass alle Knoten, die in der Transaktion beteiligt sind, seien es nun Clients in der n zu 1 Variante oder Server in der 1:n-Variante, zu verwalten und sicherzustellen, dass alle Anfragen der Clients und Änderungen der Objekte vollständig sind. Außerdem ist er für die Eindeutigkeit der TransaktionsID innerhalb des Gesamtsystems zuständig. Er ist es auch der dafür sorgt, dass Transaktionen festgeschrieben, oder im Fehlerfall abgebrochen, werden.

Zum Bewältigen seiner koordinierenden Tätigkeiten führt der Koordinator Listen der Teilnehmerknoten und besitzt die entsprechende Logik zum Auswerten der Aktionen der Teilnehmer. Im Folgenden wird kurz vorgestellt, wie der Koordinator bei der 1:n- beziehungsweise bei der n:1-Variante der einfachen Transaktionen arbeitet.

In der **1:n-Variante** hält der Koordinator eine Liste aller beteiligten Server auf denen Objekte die zur Transaktion gehören liegen. Jeder Server auf dem der Client eine Aktion auslöst die der Transaktion zugehörig ist meldet sich jetzt mit einer Beitrittsanfrage bei dem Koordinator. Dieser trägt den Server in der Liste ein.
Wenn die Transaktion jetzt abgeschlossen werden soll, egal ob erfolgreich oder mit einem Abbruch, schickt der Koordinator jedem der Teilnehmer in der Liste eine Benachrichtigung und wartet dessen Rückmeldung ab. Im Falle, dass einer der Teilnehmer einen Fehler hat, meldet er dies dem Koordinator, worauf dieser entscheiden kann was mit der Transaktion geschieht.

Ähnliches gilt für die **n:1-Variante**. Hier sind allerdings die Clients und nicht die Server in einer Teilnehmerliste. Will ein Client innerhalb der Transaktion ein Objekt auf dem Server verändern, muss er sich zunächst beim Koordinator gemeldet haben. Dies geschieht ebenfalls über eine Beitrittsanfrage beim Server.
Ist ein Client mit dem Senden seiner Befehle fertig und hat für diese eine Bestätigung meldet er dem Server, dass er fertig ist. Sind alle Clients in der Liste mit ihren Änderungen auf den Objekten fertig, kann der Koordinator dem Server die Mitteilung geben, dass dieser die Transaktion festschreiben kann.

3 Anforderungen an die Lösung

3.1 Fachliche Anforderungen

3.1.1 Anforderungen der Spieler

Als Spieler wird im Folgenden die Person bezeichnet, die keinerlei Einblicke in den Code oder sonstige technische Elemente des Gesamtsystems hat. Für den Spieler ist das System eine Blackbox, die er ausschließlich durch den Spiel-Client bedienen kann.

Für den Spieler ist ein wartezeitfreies und konsistentes Spielerlebnis ausschlaggebend. Zudem sollte er nicht von den auf dem Server ablaufenden Berechnungen in der Art eingeschränkt werden, dass sich sein Spielerlebnis verschlechtert.

Dies bedeutet in Hinsicht auf die Rechenzeit auf dem Server, dass es zu keinerlei größeren Verzögerungen durch die im Hintergrund ablaufenden Transaktionen, kommen darf. Fallen diese Verzögerungen dennoch an, gilt es diese bestmöglich für den Spieler zu kaschieren. Der Spieler soll durch erhöhte Wartezeiten nicht das Gefühl bekommen, dass sein Spiel *hakt*. Ab wann eine Anwendung *hakt* ist beschreibbar durch die *sub second response* Nieson (1993).

Für ein stimmiges Erlebnis ist nicht nur die Reaktionszeit des Programmes ein wichtiger Punkt, sondern auch die Konsistenz des Systems. Aus diesem Grund muss gewährleistet werden, dass die Objekte im Spiel den Gesetzen der Spielwelt weiterhin unterliegen. Negativbeispiele hierzu wären verschwindende oder doppelt auftretende Objekte nach Ausführen einer Transaktion sowie einseitig auftretende, regelwidrige, Objektveränderungen bei mehreren involvierten Spielern.

3.1.2 Anforderungen der Administratoren

Als Administrator wird im Folgenden die Person bezeichnet, die für die Wartung des Gesamtsystems zuständig ist, allerdings keinerlei aktive Einflüsse auf den Quellcode des Servers oder

der Spiele Clients nehmen kann. Ihm stehen Programme zur Wartung der Datenbank zur Verfügung sowie Wartungswerkzeuge des Spiels.

Für den Administrator ist ausschlaggebend, dass er das System bestmöglich warten und für gegebenenfalls anstehende Support-Tickets schnellstmöglich eine Lösung finden kann. Das heißt, dass er in seinen Programmen einen direkten Blick auf aufgetretene Fehler bekommt und somit schnellstmöglich die Ursache dieser Fehler findet.
In Fehlerfällen sollte er außerdem in der Lage sein, diese Fehler nachzuvollziehen und Wiederherstellungsmaßnahmen einleiten können.

3.1.3 Anforderungen der Programmierer

Als Programmierer wird im Folgenden der Personenkreis bezeichnet, der dafür zuständig ist, das Spiel mit neuen Funktionen zu erweitern und Fehler im existierenden Programmcode zu beheben.

Aus Sicht der Programmierer muss es für weitere Programmierungen geeignete Schnittstellen geben, auf die sie aufsetzen können. Diese sollten so allgemein wie möglich und gleichzeitig so präzise wie nötig gehalten sein.

Falls es spätere Programmierungen am Transaktionscode geben sollte, ist der bisherige Programmcode ausreichend zu dokumentieren und übersichtlich zu halten.

3.2 Technische Anforderungen

Dadurch, dass die verteilten Transaktionen in ein bereits bestehendes Programm gebaut werden müssen, haben sie sich den bereits vorhandenen Konventionen anzuschließen. Am wichtigsten wäre hier, dass das Persistenz System in *Timadorus* auf den Prinzipien der *Eventual Consistency* basiert. Vgl. Kapitel 2 Stand der Technik.

In Online Diensten im Allgemeinen und Online Rollenspielen im Speziellen kommt es eher auf eine hohe Verfügbarkeit der Dienste an, als auf ein homogenes Datenabbild.
Entsprechend ist zu bedenken, dass es zu Unterschieden der Versionen der selben Objekte auf unterschiedlichen Servern kommen kann sowie zu Problemen aufgrund der Race Condition in

verteilten Systemen.

Als Endprodukt soll ein Programm in der Sprache Erlang geschrieben werden, welches seinen Einsatz im *Timadorus* Projekt finden soll. Die spezielle Entscheidung für diese Programmiersprache ist damit begründet, dass es eine Sprache ist, welche speziell auf Netzwerk Kommunikation ausgelegt ist. Außerdem sind bereits bestehende Teile des *Timadorus* Projekt in dieser Sprache geschrieben.
Optimal ist eine Erweiterung des existierenden Quellcodes der QuP-Server, um die Logik für die komplexen verteilten Transaktionen unterzubringen.

4 Konzept

4.1 Verfahrensauswahl

Im Folgenden wird entschieden welche der im Kapitel 2 Stand der Technik beschriebenen Verfahren Anwendung finden soll und aus welchen Gründen die Entscheidung für diese gefallen ist, sowie eventuelle Modifikationen die vorgenommen werden müssen um den Ansprüchen zu genügen.

4.1.1 Transaktionsverfahren

Zur Auswahl des Transaktionsverfahrens ist zu beachten, dass es sich bei den hier umzusetzenden Transaktionen um Transaktionen mit n beteiligten Servern und m beteiligten Clients handeln kann.
Entsprechend sind die herkömmlichen Transaktionen nicht verwendbar, da diese von einer 1:1 Verbindung von einem Client und einem Server ausgehen.

Die einfachen verteilten Transaktionen bieten da schon einen besseren Ansatz. Diese gehen von einem Client zu Server Verhältnis von n:1 oder 1:m aus. Trotzdem ist eine genaue Umsetzung dieser Methode nicht zielführend, wenn es zu n:m Beziehungen kommt.

Als Lösung wird eine erweiterte und komplexere Variante der einfachen verteilten Transaktionen verwendet im Sinne einer Kombination aus den beiden möglichen Szenarien. Dabei sieht es für die jeweiligen Teilnehmer so aus wie bei den einfachen komplexen Transaktionen, das Gesamtsystem ist jedoch komplexer.
Eine genaue Beschreibung dieser komplexen Transaktionen findet sich im Kapitel 4.2.1 Beschreibung Komplexe Transaktionen.

4.2 Entwurf

4.2.1 Beschreibung Komplexe Transaktionen

Wie bereits im Kapitel 2 Stand der Technik beschrieben, soll eine Kombination der 1:n und m:1 Varianten der einfachen verteilten Transaktion verwendet werden.
Diese Kombination benutzt aus beiden Varianten die jeweilige Verwaltung für mehr als einen Teilnehmer. Das bedeutet es werden die entsprechenden Teilnehmerlisten übernommen.

Als Ausgang für eine Transaktion haben wir, wie auch in den einfachen verteilten Transaktionen, einen Transaktionsmanager, welcher die Verwaltung der Transaktion und der internen Listen zur Verwaltung der beteiligten Clients und Server übernimmt.
Als Transaktionsmanager wird hier nicht ein eigenständiges Programm genutzt sondern der Server selbst wird zum Transaktionsmanager ernannt wenn er den Befehl bekommt eine Transaktion zu starten (vgl. 4.2.3 Nachrichtenprotokoll).

Alle Clients, die an der Transaktion teilhaben wollen, müssen sich jetzt bei diesem Server, welcher zum Transaktionsmanager ernannt wurde, mit der entsprechenden TransaktionsID melden und werden dort in der Client-Liste festgehalten.
Informationen über eine bestehende Transaktion an der sie teilhaben wollen müssen zuvor an diesen Client gesendet worden sein. Ein kleines Beispiel: Spieler X und Spieler Y wollen handeln. Spieler X startet die Transaktion und sendet eine Handelseinladung an Spieler Y. Nimmt dieser an, sendet sein Spiele-Client eine Beitrittsnachricht an den Transaktionsmanager. Dies geschieht auf Client zu Client Ebene und wird hier nicht genauer erläutert, da dies von Anwendungsfall zu Anwendungsfall unterschiedlich sein kann.

Ähnlich verhält es sich wenn auf ein Objekt zugegriffen wird, welches sich auf einem Server befindet, der noch nicht in die Transaktion mit einbezogen wurde. Dieser sendet ebenfalls eine Nachricht zum Registrieren an den Server. Dort wird er in die Server-Liste eingetragen und in der Transaktion mit beachtet.

Alle Objekte können weiterhin innerhalb von Transaktionen verändert werden, jedoch wird bei dem Änderungsbefehl die TransaktionsID und die Netzwerkadresse des Transaktionsmanagers mitgegeben. Alle Änderungen an den Objekten werden zunächst nur vorläufig auf dem Server vorgenommen bis der Transaktionsmanager die Anweisung gibt um die Transaktion festzuschreiben. Entsprechende Sperren sorgen dafür, dass ein Objekt nicht verändert werden

kann während eine Transaktion läuft. Lesevorgänge sind auch mit Sperren weiterhin möglich. Ist der entsprechende Server, der das Objekt hält, noch nicht in der Transaktion registriert, wird sie eine Beitrittsnachricht an den Transaktionsmanager versenden um sich in der Transaktion zu registrieren.

Um eine Transaktion abzuschließen, müssen zunächst die Clients dem Transaktionsmanager melden, dass sie alle ihre Änderungen gesendet haben und ein Commit einleiten wollen. Diese Anfrage wird gültig sobald der Transaktionsmanager von allen registrierten Clients diese Meldung bekommen hat.
Ist dies erfolgt muss zunächst festgestellt werden ob die Server (noch) in der Lage sind die Änderungen an den Objekten festzuschreiben. Der Transaktionsmanager erfragt daher bei allen in der Server-Liste eingetragenen Servern ob sie die Transaktion festschreiben können (vgl. 4.2.3 Nachrichtenprotokoll).
Die befragten Server bestätigen dies oder senden eine Fehlermeldung an den Transaktionsmanager der wiederum darüber entscheiden muss ob die Transaktion den Erwartungen der Spielwelt entsprechend richtig ist oder sie abgebrochen werden muss. In den meisten Fällen, in denen eine Transaktion scheitert, wird ein Abbruch mit anschließendem Wiederherstellungsversuch der Transaktion ausgeführt.

Haben alle Server der Transaktion erfolgreich zurückgemeldet, dass sie Ressourcen festschreiben können, sendet der Transaktionsmanager den Befehl zum endgültigen Festschreiben der Objekte auf den Servern. Der Transaktionsmanager wartet jetzt noch auf eine positive Rückmeldung von allen Servern und sendet darauf den Clients der Transaktion eine Erfolgsnachricht. Der Transaktionsmanager muss die Transaktion noch eine Weile im Cache vorrätig halten, da die Server von sich aus noch einmal erfragen, ob die Transaktion denn auch wirklich geklappt hat. Erst wenn sie dies positiv beantworten können, werden die Objekte auf den Servern freigegeben und können von anderen Prozessen verwendet werden.

Im Falle einer abgebrochenen Nachricht müssen alle Server über diesen Abbruch informiert werden. Dies geschieht über die Abbruchnachrichten die der Transaktionsmanager an die Server sendet. Die Server werden in diesem Fall ihre temporären Versionen der Objekte nicht festschreiben.
Im Falle des Abbruches nachdem alle Server ihre Daten festgeschrieben haben, jedoch noch nicht ihre geänderten Objekte wieder freigegeben haben muss ein Objekt Backup von vor der Transaktion geladen werden und erst danach die Objekte freigegeben werden.

Dies dient zum Vorbeugen eventueller Serverausfälle nachdem die Server bereits zurückgemeldet haben, dass sie Objekte festschreiben können.

Objekte werden gesperrt um keinerlei Veränderung am Objekt zuzulassen, damit es im Falle von Schreibaktionen von außen nicht zu inkonsistenten Objekten kommen kann.

4.2.2 Komponentenentwurf

Nachfolgend eine kurze Übersicht in welchem Teilbereich des Timadorus Projektes Elemente eingearbeitet werden und eine Beschreibung der Komponenten und deren Bestandteile wie sie nach der Beschreibung benötigt werden.

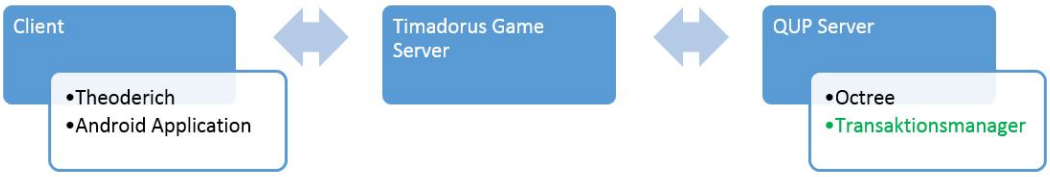

Abbildung 4.1: Erweiterung im Timadorus System

Der Hauptbestandteil des Transaktionsmanagers sind die zwei Listen, welche die an der Transaktion beteiligten Server und Clients festhalten. Diese werden in der Liste als eine Kombination aus TransaktionsID und Node-Adresse festgehalten. Die Node-Adresse muss einzigartig im gesamten Netzwerk sein und muss ebenso die Information enthalten wie man sie über das Netzwerk erreichen kann. Die TransaktionsID dient dabei der Zuordnung zu einer Transaktion, sodass der Transaktionsmanager mehr als nur eine Transaktion verwalten kann.

Zusätzlich zu den Listen benötigt der Transaktionsmanager noch zwei Schnittstellen für die Kommunikation zu den Qup-Servern und den Clients. Die unterstützten Nachrichten bzw. Funktionen zu diesen finden sich im Kapitel 4.2.3 Nachrichtenprotokoll.

Die Logik zum Verwalten der Transaktion und Senden der Nachrichten an die Server bzw. Clients wird im Kernprogramm des Transaktionsmanagers implementiert.

Der Transaktionsmanager agiert als vorgeschobener Nachrichtenempfänger zum QuP Server. Alle Nachrichten werden zunächst durch diesen geleitet und er entscheidet, ob es sich um Nachrichten für eine Transaktion handelt oder nicht. Herkömmliche Nachrichten, die Nachrichten ohne eine zugehörige Transaktion, werden einfach an den unterliegenden QuP Server weitergeleitet.

4.2.3 Nachrichtenprotokoll

4.2.3.1 Beschreibung der Nachrichten

Durch die Implementierung der Transaktionen in *Timadorus* kommt es naturgemäß auch zu einer Erweiterung im Nachrichtenprotokoll zwischen dem *Timadorus Game Server* und dem QuP Server. Diese Nachrichten werden im Folgendem mit Header und einer fachlichen Erläuterung beschrieben, wobei der Rückgabewert der Nachricht hinter dem Pfeil beschrieben wird:

OpenTransaction() -> TransaktionsID
Nachricht des *Timadorus Game Servers* an den QuP Server. Eröffnet eine Transaktion und erklärt den betroffenen QuP Server zum Koordinator der Transaktion. TransaktionsID muss eine eindeutige ID im kompletten System sein und ist daher eine Zusammensetzung aus Servername + Zeit + Laufnummer.

CommitTransaction(TransaktionsID) -> (Ja/Nein)
Der *Timadorus Game Server* gibt, nachdem alle fachlichen Bedingungen erfüllt sind, dem Koordinator den Befehl die Transaktion festzuschreiben. Der Koordinator verantwortet das Festschreiben und teilt dem Game Server mit ob die Transaktion erfolgreich geschrieben wurde oder nicht.

AbortTransaction(TransaktionsID)
Der *Timadorus Game Server* gibt dem Koordinator den Befehl die Transaktion abzubrechen. Dieser kümmert sich asynchron um das Rollback der Transaktion. Für den Game Server ist die Transaktion in dem Moment abgebrochen in der er die Nachricht versendet hat.

JoinTransaction(Netzwerkadresse, TransaktionsID) -> (Ja/Nein)
Der Timadorus Gameserver registriert sich, mit seiner Netzwerkadresse, in einer bereits bestehenden Transaktion als weiteres Mitglied in dieser. Der Koordinator trägt diesen in seiner

internen Liste ein und muss von nun an auch auf diesen TGS warten bevor er anfängt eine Transaktion abzuschließen.
Alle anderen Nachrichten des TGS an die bisherigen QuP Server werden ebenfalls unterstützt. Diese sind jedoch alle um einen Parameter, die TransaktionsID, erweitert um eine Zuordnung durchzuführen.
Außerdem gibt es neue Nachrichten der QuP Server untereinander. Diese sind dem „Zwei-Phasen-Commit-Protokoll“entnommen:

CanCommit(TransaktionsID) -> (Ja/Nein)
Aufruf des Transaktionsmanagers an andere Server auf denen die Datenmanipulation stattfinden soll, ob er die zur Transaktion gehörenden Änderungen festschreiben kann. Der gefragte Server antwortet positiv oder negativ auf diese Frage. Die Ergebnisse werden später benötigt um Entscheidungen über die Gültigkeit der Transaktion zu treffen.

DoCommit(TransaktionsID)
Asynchroner Befehl an einen an der Transaktion teilnehmenden Server alle Ressourcen festzuschreiben die zur Transaktion mit der angegebenen TransaktionsID gehören.

HaveCommitted(TransaktionsID, Netzwerkadresse)
Asynchrone Antwort des mit *DoCommit* angewiesenen Servers, an den Koordinator, welche bestätigt, dass alle Ressourcen der Transaktion festgeschrieben wurden. Die Netzwerkadresse dient zur Zuordnung des Servers im Koordinator.

DoAbort(TransaktionsID)
Asynchroner Befehl an einen an der Transaktion teilnehmenden Server alle der Transaktion zugehörigen Aktionen abzubrechen und zwischengespeicherte Werte zu löschen.

GetStatus(TransaktionsID) -> (Ja/Nein)
Aufruf eines Teilnehmers einer Transaktion der nicht der Koordinator ist an den Koordinator um nach dem Status einer Transaktion zu fragen. Gibt den Comitted oder Aborted Status der Transaktion als Booleschen Wert zurück.

JoinTransaction(TransaktionsID, Netzwerkadresse)
Nachricht die ein noch nicht in der Transaktion registrierter Server an den Transaktionsma-

nager schickt um sich bei diesem mit seiner eigenen Netzwerkadresse in der Transaktion zu registrieren.

4.2.3.2 Kommunikationsdiagramm

In den folgenden Diagrammen wird ein typischer Nachrichtenaustausch zwischen dem TGS und den QuP Servern dargestellt.

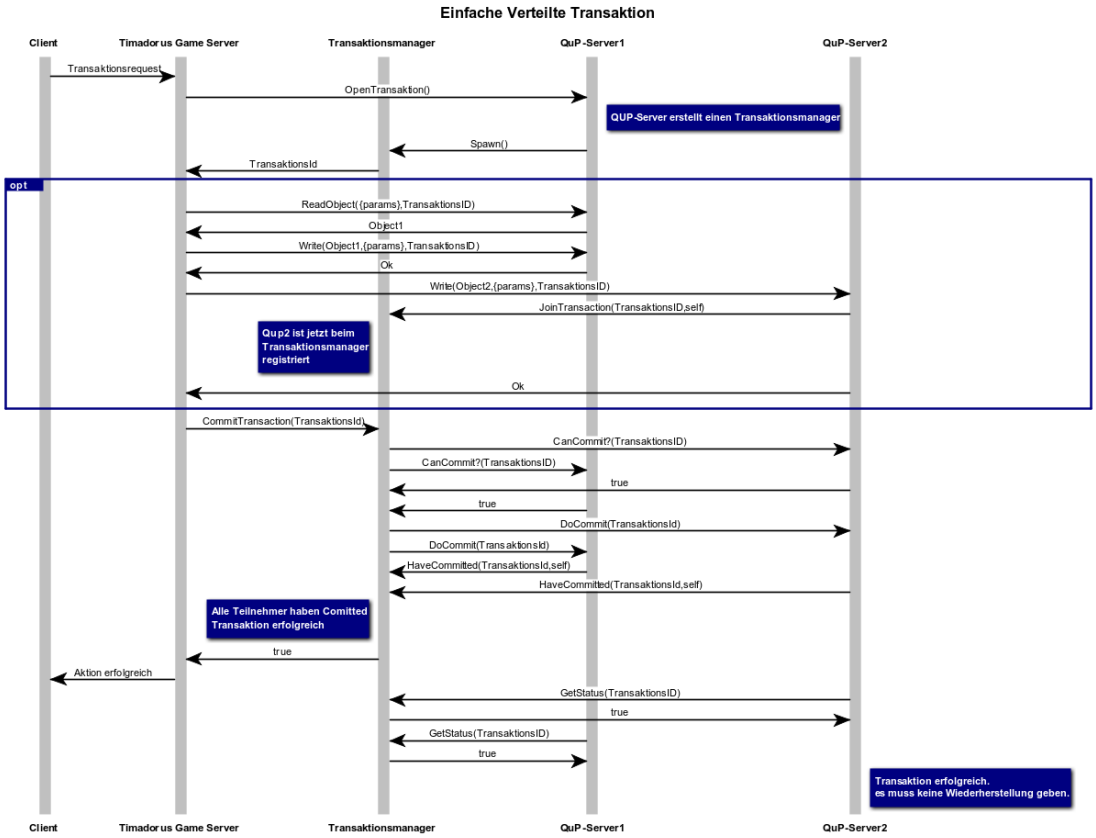

Abbildung 4.2: Nachrichtendiagramm einfache, verteile Transaktion

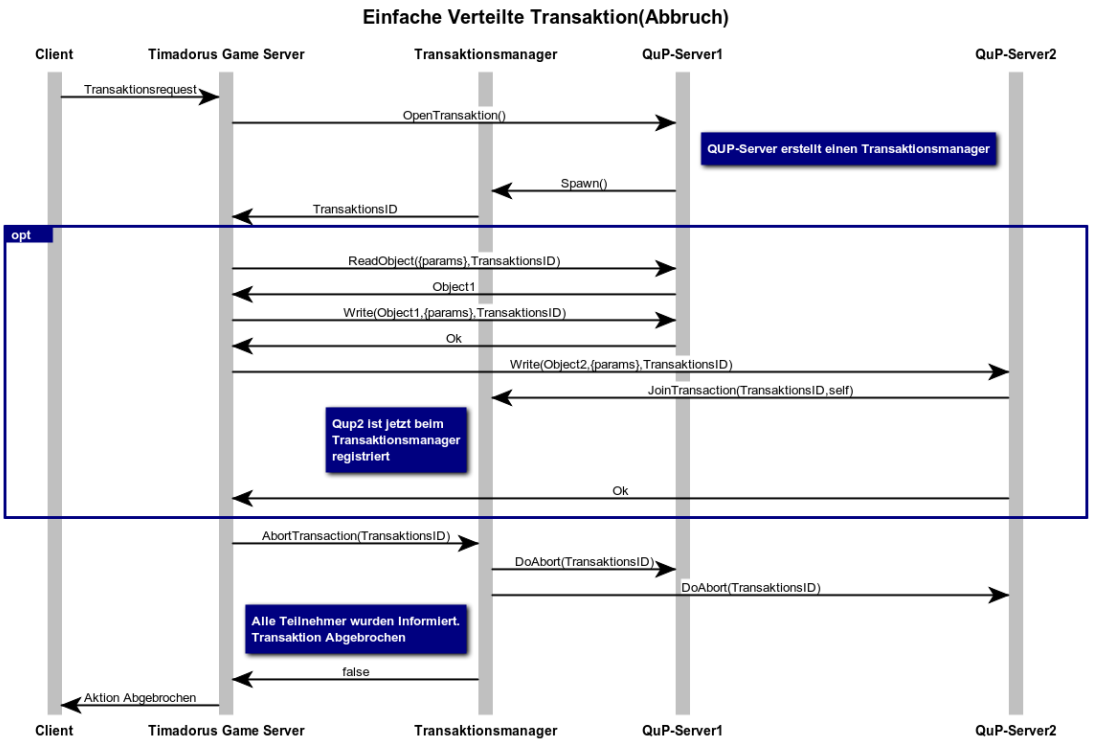

Abbildung 4.3: Nachrichtendiagramm einfache, verteile Transaktionen mit Abbruch

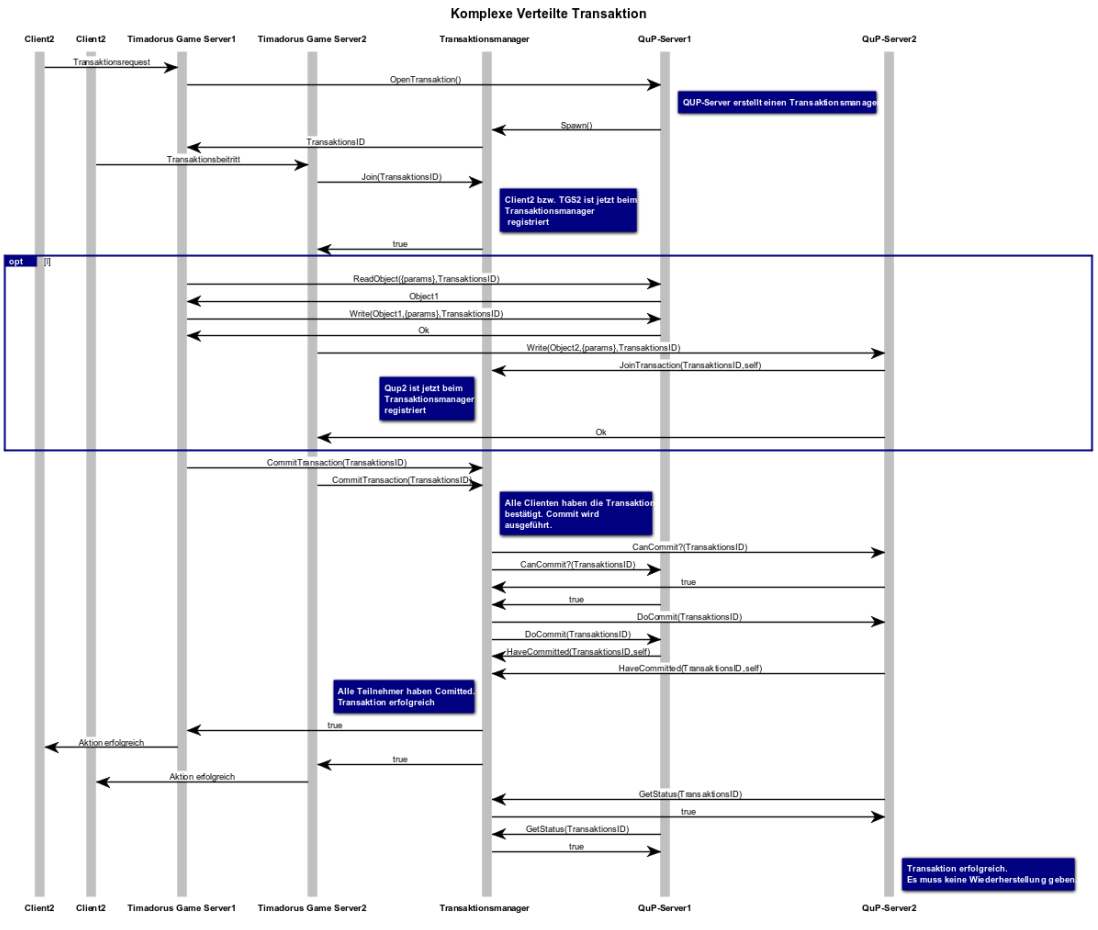

Abbildung 4.4: Nachrichtendiagramm komplexe, verteilte Transaktionen

4.2.4 Implementierungsaspekte

Im Falle, dass es zu einer Implementierung dieses Entwurfes kommt gilt es zu beachten, dass es derzeit im *Timadorus System* keinerlei Nebenläufigkeitskontrolle gibt. Diese müsste zunächst implementiert werden. Im Folgenden wird kurz die bevorzugte Art der Nebenläufigkeitskontrolle erläutert jedoch sind ebenfalls andere Arten denkbar.

Bei der Entscheidung für das Verfahren zur Nebenläufigkeitskontrolle, ist insbesondere die Architektur des *Timadorus* Systems zu beachten. Dieses ist auf eine nahezu unendliche Skalierbarkeit und Aufteilbarkeit der Spieleserver und der Datenbank ausgelegt, bei denen im Härtefall ein Server pro Benutzer oder Objekt eingesetzt wird.

Lokale Schleifenfindung, auch als *Edge Chasing* oder *Path Pushing* bezeichnet, ist ein auf verteilte Systeme optimiertes Verfahren welches sich nicht auf einen globalen Warte Graph beruft, sondern jede einzelne Server-Node kennt einige seiner Verbindungen zu anderen Server-Nodes (im Folgenden Kanten genannt). Durch das Senden von Analyse Nachrichten, genannt „Probes“(Sonden), sollen Deadlocks erkannt werden und darauf lokal entschieden werden, ob eine Transaktion abgebrochen werden soll oder nicht.
Eine Probe besteht aus den Transaktions-Wartebeziehungen einer Kante die einen Pfad in einem globalem Wartegraph entsprechen.
Nach Coulouris Coulouris u. a. (2002) gibt es einen Dreiphasenablauf im Edge-Chasing: Initiierung, Erkennung und Auflösung.
Initiierung: Ein Server erkennt, dass eine Transaktion T beginnt auf eine andere Transaktion U zu warten und, dass U auf die Freigabe einer Ressource auf einem anderen Server wartet. Darauf sendet er eine *Probe-Nachricht* mit der Kante <T -> U> an den Server auf den die Transaktion U wartet. Verwendet U eine gemeinsame Sperre werden die Probe Nachrichten an alle Teilnehmer der Sperre gesendet.
Erkennung: Bei der Erkennung werden diese Probe Nachrichten empfangen und ausgewertet ob ein Deadlock vorliegt und ob die Probe Nachrichten weitergegeben werden sollen oder nicht.
Empfängt so zum Beispiel ein Server eine Probe Nachricht <T -> U> (T wartet auf eine Transaktion U) prüft er für sich ob auch er auf U wartet. Sollte dies der Fall sein erweitert er die Probe Nachricht um seine eigene Transaktion, so dass sich die Nachricht <T -> U -> N> ergibt. Der Server prüft nun ob das Hinzufügen der eigenen Transaktion eine Schleife erstellt hat, wie zum Beispiel: <T -> U -> N -> T>. Sollte eine Schleife gefunden werden wird mit der Phase Auflösung fortgefahren, falls nicht werden die Nachrichten wie in der Initiierungsphase

verbreitet.
Auflösung: Wird eine Schleife von einem der Server erkannt muss eine der Transaktionen innerhalb dieser Schleife abgebrochen werden, um den entstandenen Deadlock zu lösen.

Die Entscheidung fällt in Kontext dieser Bachelorarbeit auf diese Methode, da diese in einem unendlich skalierbaren System funktioniert und keinerlei Einschränkungen besitzt. Einziges Manko hier ist die umfangreiche Implementierung dieser Methode.

5 Analyse der Verwendbarkeit

Das im Entwurf gezeigte Modell ist sehr ideell gehalten und geht von einer reinen technischen Seite an die Problemstellung heran.
Im Folgenden wird die Eignung dieses Ansatzes im Kontext des *Timadorus* Projekts analysiert und auf Konformität mit den fachlichen und technischen Anforderungen geprüft.

Um alle Beschreibungen an einem Beispiel festmachen zu können, führen wir hiermit das Beispiel des Handelns auf dem Tisch an. In diesem Beispiel sitzen sich 2 Spieler gegenüber und legen die zu handelnden Gegenstände vor sich auf einen Tisch. Ein Wachmann überwacht die Richtigkeit des Handels. Dies ist eine vereinfachte Form der komplexen Transaktion im echten Leben und deckt alle Sonderfälle ab.
In diesem Beispiel gibt es wie in der Transaktion 3 Phasen:
Angebot: Beim Angebot treffen sich die unterschiedlichen Handelspartner am Tisch und legen auf diesen die Gegenstände die gehandelt werden sollen. Dabei wird auch entschieden, welche Person welchen Gegenstand nach dem Handeln erhalten soll.
Einigung: Jede der Personen an diesem Tisch äußert, dass sie mit dem Handel zufrieden ist und signalisiert damit, dass sie fertig ist und nur noch auf die anderen Personen am Tisch wartet. Gegenstände dürfen ab dem Moment nicht mehr verändert werden ohne dass die zugehörige Person abermals zustimmt.
Auflösung: Haben alle Personen signalisiert, dass sie fertig sind und keines der Objekte auf dem Tisch mehr angefasst wird, schieben sich die Personen unter den Augen des Wachmannes die Gegenstände zu. Am Ende sagt der Wachmann noch ob der Tausch gültig war oder ob alle Gegenstände zurück an ihre ehemaligen Besitzer gegeben werden müssen.

5.1 Skalierbarkeit

Ziel des *Timadorus* Projektes ist es ein theoretisch unendlich skalierbares System zu bauen. Hierzu müssen auch bei den Transaktionen die nötigen Vorkehrungen getroffen werden um dies zu unterstützen.

Bei den Transaktionsmanagern wurde darauf geachtet, dass kein gesondertes Programm für die Transaktionskontrolle benötigt wird. Stattdessen ist es ein verankertes Stück im QuP Server selbst. Dies bedeutet, falls ein neuer Server hinzu geschaltet werden muss, hat man automatisch dafür gesorgt, dass ein potentieller Transaktionsmanager gestartet wurde. Entsprechend kann es nicht vorkommen, dass man vergisst einen Transaktionsserver oder eine externe Load-Balancing Prozedur zu starten.

Die internen Listen des Transaktionsmanagers sind einzig durch die verfügbare Speicherkapazität des Computers auf dem der Server läuft begrenzt. Demzufolge kann man unendlich viele Clients und Objekte an einer Transaktion teilhaben lassen. Große Listen und hohe Verteilung der Objekte auf viele Server führen jedoch zu Performance-Problemen die sich im Spielfluss als *Lag* bemerkbar machen können. Deshalb sollte man sich genau überlegen, welche Aktionen man transaktional ausführen möchte und welche lieber auf einem anderen Weg realisiert.
Am Beispiel: Solange man nur zwei Personen hat, die an einem Tisch sitzen und handeln bleibt alles übersichtlich und ist gut abwickelbar. Setzen sich allerdings 100 oder mehr Personen an einen Tisch wird schnell deutlich, wie umständlich das Auflösen in der dritten Phase des Handels wird.

5.2 Technische Korrektheit

Die Transaktionen müssen technisch korrekt ausgeführt werden. Dies bedeutet, dass es keine Duplizierungen oder Löschungen von Objekten geben darf, die nicht explizit gewünscht sind oder halb ausgeführte Transaktionen (einer der Handelspartner hat seine Objekte abgegeben, der andere bekommt diese gibt seine jedoch nicht ab).

Fehler dieser Art können auf zwei Weisen auftreten: Als Fehler im Ablauf, sei es Nachrichtenaustausch oder Ausfall der Server, und als aktives Einwirken von außen durch Cheats.

Zur Absicherung der Serverfehler, die auftreten können, wurde das *zwei-Phasen-Commit-Protokoll* verwendet. Dieses garantiert die für verteilte Transaktionen wünschenswerten ACID Eigenschaften. vgl. Haerder und Reuter (1983)
Zur Sicherstellung, dass Nachrichten die von Clients an die Server oder von Server an Server gesendet werden nicht verloren gehen, ist das Erlang zu Grunde liegende Nachrichtenprotokoll zuständig.

Zur Abwehr von Cheats werden alle Veränderungen an Objekten auf Serverseite ausgeführt und unterliegen dort einer fachlichen Prüfung auf die Korrektheit der Veränderungen und können bei Verdacht auf Manipulation Gegenmaßnahmen einleiten.
Gegen direkte Angriffe auf den Server ist die Transaktion jedoch nicht von sich aus geschützt und bedarf einer externen Absicherung durch Firewalls oder anderen Gegenmaßnahmen.

5.3 Fachliche Korrektheit

Zur fachlichen Korrektheit gehört, dass die Spielwelt im Sinne von *Timadorus* stimmig bleibt. Dies bedeutet eine hohe Verfügbarkeit der Objekte zu jedem beliebigem Zeitpunkt und keine, der realen Welt entsprechend, unlogischen Aktionen. Außerdem gilt es die messbare Reaktionszeit für Spieler zu unterschreiten, damit dieser nicht das Gefühl bekommt, dass er „nur“ ein Spiel spielt.

Durch die in der technischen Korrektheit genannten Punkte werden die vom Spieler erwarteten Resultate erzielt: Die zu manipulierenden Objekte sind entweder in ihrer Gesamtheit verändert oder gar nicht.

Bei der Analyse der entworfenen Methode fällt auf, dass sich aus der angewandten technischen Korrektheit eine fachliche Hürde aufbaut. Objekte sollen zu jedem Zeitpunkt verfügbar und interagierbar sein. Setzt man jedoch die beschriebenen Techniken ein, kommt es dazu, dass man Objekte hat, die sich in einem Lock befinden. Will eine dritte Partei auf diese Objekte zugreifen, wird sie durch die vorher angelegten Sperren am Zugriff gehindert.
An unserem Beispiel beschrieben wäre das, als wenn die beiden Handelnden jeweils einen Apfel auf den Tisch legten. Dieser Apfel ist jetzt vollkommen aus dem Spiel herausgelöst bzw. man sieht zwar, dass der Apfel existiert, kann ihn trotzdem nicht als Außenstehender anfassen oder manipulieren. Kommt also ein Dieb an den Tisch und versucht den Apfel zu nehmen wird

er merken, dass dieser nicht bewegbar ist.
Außerdem können diese Mechanismen ausgenutzt werden um unrealistische Zustände zu erreichen. In *Timadorus* gibt es Diebstähle von Spielern an Spielern. Würde das Handeln so implementiert werden wie in diesem Entwurf wäre es möglich, dass Spieler die sich von der Tastatur entfernen möchten sich mit einem anderen Spieler treffen und das Handelsfenster öffnen. Dort geben sie vor den gesamten Inhalt des Inventars handeln zu wollen, tun dies jedoch nicht. Ihre Gegenstände sind jetzt durch das Lock vor potentiellen Dieben geschützt und beide Spieler können sich von ihrer Tastatur entfernen ohne sich vor Diebstählen fürchten zu müssen.
Dieses Erstellen von Schlupflöchern im Spiel kann nicht Sinn der Transaktionen sein.

Vergleicht man andere bereits am Spielemarkt vorhandene MMOs und deren Behandlungen der Transaktionen fällt auf, dass diese meistens ein sehr gekapseltes Transaktionsverfahren haben. Das heißt, sie verschleiern die Transaktionen durch Dialogfenster und schränken den Zugriff von Außen auf ein Minimum ein. Meistens sind Zugriffe von Außen sogar komplett ausgeschlossen, womit die oben genannten Situationen gar nicht erst auftreten können.

Abbildung 5.1: Herkömmliches abgekapseltes Verfahren am Beispiel „Borderlands2“

5.4 Erklärung durch CAP-Theorem

Transaktionen im herkömmlichen Sinne stellen die ACID Eigenschaften sicher, welche benötigt werden um ein konsistentes System zu erhalten. Bei verteilten Transaktionen gibt es jedoch ein Problem, welches Eric Brewer in seinem CAP-Theorem beschreibt (vgl. Page und Nance (1994)).
Das Theorem erklärt warum es in verteilten Systemen nicht möglich ist mehr als zwei der drei Eigenschaften Konsistenz, Partitionierbarkeit und Verfügbarkeit zu erreichen.
Auf das BASE-System, welches eine weichere Handhabung des CAP gewährleistet, kann in dem Moment, in dem Objekte, durch die Transaktion, verändert werden sollen, leider nicht zugegriffen werden. Diese Mechanismen würden erst nach der Transaktion greifen um ein konsistentes Gesamtsystem sicher zu stellen.

Eine Zusammenführung zwischen CAP und den fachlichen Anforderungen der dauerhaften Verfügbarkeit der Objekte ist nicht möglich wenn man gleichzeitig Konsistenz und Partitionierbarkeit haben möchte. Wobei ein verteiltes System schon vorgibt, dass die Eigenschaft partitionierbarkeit gewährleistet sein muss und durch die Transaktion die Eigenschaft konsistent belegt wird.

5.5 Resultat Analyse

Resultierend aus der vorhergehenden Analyse kann man sagen, dass einer Implementierung der komplexen Transaktionen aus technischer Seite nichts im Wege steht.

Betrachtet man jedoch den Kontext der Arbeit und die fachliche Beschaffenheit des *Timadorus* Projekts muss man feststellen, dass es mit diesem Ansatz der Transaktionen nicht möglich ist die benötigte dauerhafte Verfügbarkeit der Objekte, die für eine reale Darstellung der Welt benötigt wird, zu garantieren. Belegt wird dies durch den Verstoß gegen die Prinzipien des CAP-Theorems indem man Verfügbarkeit, Partitionierbarkeit und Konsistenz zusammen fordert.

Entsprechend ist die vorgestellte Lösung ***nicht***, ohne starke Einschränkungen des Spielflusses, verwendbar.

Um alternative Lösungen für diesen Spezialfall zu liefern, müssen Annäherungen oder Alternativen entwickelt werden. Diese werden im Kapitel 7 Ausblick einzeln vorgestellt und deren Vor- und Nachteile herausgestellt.

6 Zusammenfassung

Zusammenfassend lässt sich sagen, dass die Transaktionen umsetzbar sind und damit auch große Vorteile bringen können, jedoch sollte man sich vor der Implementierung dieser bewusst machen, welche Einschränkungen man sich dadurch in das System holt.
Entsprechend ist bei einer Anwendung mit extrem hoher Verfügbarkeit, wie dem *Timadorus* Projekt und auch anderen MMOs, von einer strikten Implementierung abzuraten. Man würde zwar eine große Stabilität innerhalb des Systems schaffen und dem Ausnutzen von Schwachstellen (Exploits) zuvorkommen jedoch nur auf Kosten der Immersion, beziehungsweise dem Spielererlebnis, da man starke Abkapselungen der Objekte vornehmen muss.
Dies kann man jedoch wiederum zum Ansatzpunkt für andere Arten der Transaktion und des Spielerlebnisses nehmen und etwas entwickeln, was der Fachlichkeit entspricht und weniger umfangreiche technische Eingriffe erfordert.

Falls man sich dennoch zur Umsetzung der Transaktionen entscheidet wie sie hier beschrieben sind, kann man dieser Arbeit entnehmen auf welche Schwierigkeiten man im Spielfluss achten muss. Diese gilt es darauf möglichst unauffällig zu kaschieren, sodass so wenig Immersion wie möglich zerstört wird.

7 Ausblick

Im Kapitel Analyse haben wir festgestellt, dass die technisch richtige Lösung für das Problem der komplexen verteilten Transaktionen im Kontext des *Timadorus-Projektes* nicht problemlos verwendbar ist. Es werden daher Alternativen entwickelt, um die genannten Probleme zu eliminieren oder einzudämmen.
Im Folgenden werden diese alternativen Ansätze beschrieben und kurz auf ihre Verwendbarkeit geprüft.

Eine detailierte Beschreibung dieser, sowie eine etwaige Implementierung der Lösung ist für andere Arbeiten im *Timadorus System* vorgesehen und werden daher hier nicht ausgearbeitet.

7.1 Zurückhalten der Sperren

Die naheliegende Lösung des Problems wäre die Sperrungen so lange zu verzögern, bis sie wirklich nötig werden. Dies bedeutet, dass man Objekte nicht sofort sperrt wenn sie für die Transaktion markiert werden, in unserem Beispiel wenn sie auf den Tisch gelegt werden, sondern erst in dem Moment an dem der Commit für die Transaktion gesendet wird.
Dies hätte zur Folge, dass Objekte bis zu diesem Moment frei manipulierbar sind und entsprechend mehr der realen Welt entsprechen. Der Transaktionsmanager muss entsprechend wissen wenn eines der Objekte außerhalb der Transaktion manipuliert wurde. Dazu gäbe es im *Timadorus* Projekt zwei Möglichkeiten:
Timadorus besitzt eine interne Kontrolle die verhindert, dass Objekte geändert werden können wenn sie vorher von anderer Stelle manipuliert wurden. Dies geschieht über einen Abgleich der Ursprungsobjekte vor der Genehmigung der Veränderung durch den Server. Soll heißen: Geht der Client von einer anderen Ursprungsversion aus als der Server, kann keine Manipulation dieses Objektes stattfinden. Für die Transaktion bedeutet dies, dass sie automatisch fehlschlagen würde und zurück gerollt wird, da eine der Objekte in der Transaktion nicht verändert werden kann.
Nachteil dieser Methode wäre, dass die Transaktion erfolgreich aussieht, bis auf den Knopf zum Committen gedrückt wird. Des weiteren müsste man die Kosten auf Serverseite tragen,

die aufkommen wenn eine Transaktion abgebrochen wird und zurückgerollt werden muss. Effektiver wäre hier eine Implementierung bei der der Transaktionsmanager sich als Observer auf das Objekt registriert und sofort reagiert wenn sich dieses geändert hat, sei es mit Informieren der Clients und Erfragen wie die Transaktion weiter gehandhabt werden soll oder dem direkten Abbruch der Transaktion.

Die Vorteile dieser Variante der Transaktionen werden sofort deutlich: Die Zeiten in denen ein Objekt nicht im Zugriff durch andere Spieler ist wird minimiert und nähert sich somit der Vorstellung der realen Welt mehr an als die rein technische Variante. Des weiteren wird durch diese Umsetzung der Fall des vorsätzlichen Sperrens von Objekten zum Schutz vor anderen Spielern ausgeschlossen, da die Sperre erst eintritt wenn die Objekte verschoben werden. Nachteilig ist dennoch weiterhin, dass es Sperren gibt. Dies kann bei der falschen Konstellation dazu führen, dass Objekte trotzdem über einen längeren Zeitraum gesperrt sind, etwa wenn ein Client nach dem Auslösen der Transaktion sich beendet. Dies führt abermals zu Verzögerungen im Spielfluss und zerstört die Immersion des Spielers.

Abschließend kann man zu dieser Alternative sagen, dass sie sehr nah an das gewünschte Ergebnis herankommt aber dennoch am Ende an den Sperren scheitert. Falls man diese Umsetzungsweise wählen sollte, muss man sich dessen bewusst sein.

7.2 Auflösbare Sperren

Eine weitere Methode ist das Erfragen der Auflösung von Sperren. Will ein Benutzer ein Objekt benutzen, welches sich in einer Transaktion befindet, muss er zunächst bei dem entsprechenden Server-Knoten erfragen ob er dieses Objekt trotzdem verändern darf. Der Transaktionsmanager kann darauf entscheiden ob er das Objekt temporär freigibt und Änderungen an diesem zulässt oder nicht.

Sieht man sich dieses Verfahren an erkennt man, dass man im Prinzip auch das bereits in 7.1 Zurückhalten der Sperren behandelte Verfahren verwenden könnte, da dieses ähnliche Eigenschaften besitzt. Entsprechend ist auch dieses Verfahren nicht möglich da wie bereits beschrieben weiterhin Sperren eingesetzt werden müssen um zumindest die Endphase abzusichern.

7.3 Durchlässige Sperren

Durchlässige Sperren sind eine spezielle Implementierung der Sperren an einem Objekt, bei dem nur bestimmte Aktionen unterbunden werden sollen, welche die Transaktion zum scheitern bringen könnten. An unserem Beispiel mit dem Tisch wäre es demnach möglich einen Gegenstand auf dem Tisch zu verrücken, allerdings nicht ihm vom Tisch zu nehmen und damit den Bereich der Transaktion zu verlassen geschweige denn den Besitzer außerhalb der Transaktion zu verändern.

Bei einigen Aktionen ist der Spieler damit in der Lage auch von außen auf Objekte einzuwirken ohne, dass er dabei vom Spiel geblockt wird und aus der Spielwelt gerissen wird und trotzdem sind die Transaktionen in sich geschlossen konsistent. Nachteilig hierbei ist jedoch das Ausschließen der anderen Aktionen auf diesen Objekten. Würde man das Beispiel nehmen hätte ein Dieb weiterhin keine Chance sich der Objekte auf dem Tisch zu bemächtigen, eine Aktion die möglich sein sollte. Des weiteren ist es wie schon zuvor möglich Objekte in dieser Grauzone der Transaktion in Sicherheit vor anderen Spielern zu bringen.

Zusammenfassend ist zu sagen, dass diese Methode zwar einige Einschränkungen aufhebt diese jedoch nur kosmetischer Natur sind und man im Endeffekt die gleichen Probleme hat wie auch bei den komplexen verteilten Transaktionen. Siehe zur vollständigen Analyse: 5 Analyse der Verwendbarkeit.

7.4 Fachliche Prüfungsmechanismen

Die vorgestellten Transaktionen werden durch eine fachliche Prüfungsschicht gesperrt. Kommt es zu einer Änderung an einem Objekt muss es eine Prüfung geben ob die Transaktion noch der fachlichen Korrektheit entspricht oder nicht.
Dazu muss es eine Implementierung zur Handhabe der Änderungen innerhalb einer Transaktion geben. Optimal wäre eine Prüfung ob die Transaktion weiterhin ausgeführt werden kann oder ob es zu einem fachlichen Problem mit dem Objekt gekommen ist. Einfachste Implementierung hierfür wäre eine Abfrage beim Spieler.
Ebenfalls denkbar ist eine Automatisierung dieser Überprüfungen wobei es dadurch zu erheblichem Aufwand kommen kann, da jede Situation in der eine Transaktion auftreten kann abgedeckt sein muss.

Diese Lösung kommt wie die auflösbaren Sperren und dem Zurückhalten der Sperren sehr nah an eine Lösung heran. Diese Lösung würde es sogar noch ermöglichen, dass Objekte ein wenig später als bei diesen veränderbar ist. Dennoch bleibt auch hier eine Sperre.
Des Weiteren ist der programmiertechnische Aufwand zu beachten, welchen diese Methode mit sich zieht. Denkbar wäre hier die Festlegung einer eigenen Sprache oder einfacher eines Enums zur Bestimmung der Verwendbarkeit von Objekten.

7.5 Nicht transaktionale Lösungen

Die nicht transaktionalen Lösungen sind speziell auf das *Timadorus* Projekt angesetzt und sind damit keine direkten Verbesserungen der Transaktion aus dem Entwurf sondern eine alternative Herangehensweise an die fachlichen Problemstellungen.

Eine der einfachsten Lösungen wäre es, den Nachteil, dass keine Transaktionen in einem solchen System möglich sind, zu seinem Vorteil zu machen.
Dies könnte man zum Beispiel erreichen, indem man dies zu einem festen Bestandteil seines Spieles macht und keinerlei technischen Unterstützungen für solche Dinge zulässt, sondern dies vom Spieler ausspielen lässt. Will man demnach handeln, muss man sich tatsächlich im Spiel an einem Tisch treffen und die entsprechenden Gegenstände darauf legen und sie sich nacheinander zuschieben.
Gibt man diese Überlegung in unser Beispiel stellt man fest, dass man damit alle Eventualitäten, welche auftreten können, abgedeckt hat.

Zum Thema Fairness kann gesagt werden, dass dies alles Teil des Spieles ist. So sind Händler, die falsche oder keine Ware liefern nicht etwa administrativ vom Spiel zu bestrafen sondern die Gemeinschaft der Spieler kann sich eigene Mechanismen ausdenken um diese „Straftaten“zu ahnden.
Denken wir einmal an das europäische Gebilde des ehrbaren Kaufmannes Loetzsch u. a. (2013) welches voll und ganz auf dem Vertrauen von Menschen untereinander beruht. Spieler könnten diese Titel an Händler vergeben , die einen guten Ruf genießen.
Eine weitere Idee ist, dass man Wachpersonal für den Handel benötigt.

Für diese Art des transaktionslosen Handelns gibt es bereits Beispiele aus bisherigen Spielen wie zum Beispiel *Haven and Hearth* bei dem Gegenstände auch auf den Boden gelegt werden müssen (vgl. Tolf und Johannessen). Entsprechend hat sich hier auch eine gewisse Handelsmo-

ral entwickelt.

Nachteilig an dieser Lösung ist, dass im Falle eines Serverfehlers und damit verbunden einer Trennung der Verbindung zwischen Spieler und Spiel es zu unerfreulichen Effekten kommen kann. Legt ein Spieler demnach einen Gegenstand auf den Tisch und der Server stürzt ab, kommt es bei der Wiederherstellung des Servers und dessen Daten zu dem Effekt, dass der Apfel auf dem Tisch liegt, der Spieler jedoch nicht mehr da ist.

Unterstützend könnte man ein Moralsystem einbauen, welches dem Spieler einen Vertrauenswert zuordnet. Dieses könnte entweder durch die Gemeinschaft der Spieler selbst gepflegt werden oder es gibt eine technisch überprüfbare Vereinbarung, bei der man nach Abschluss Vertrauenswürdigkeit gewinnt.

7.6 Weitere Ausblicke

Mögliche Arbeiten an dem *Timadorus* Projekt zum Thema Transaktionen oder daran angelehnte Themen können aufbauend auf der Analyse und der gelieferten Alternativen überlegen, wie sie diese umsetzen und implementieren können.

Die entworfene Methode zum Verarbeiten von Transaktionen kann an anderen Stellen, die keine zwangsweise Verfügbarkeit der Objekte benötigen, angewendet werden. Die dortigen Anforderungen müssen jedoch neu geprüft werden. Für die Erleichterung dieser Prüfung können die in dieser Arbeit angeführten Probleme als Referenz genommen werden.

Im Anschluss an diese Arbeit könnte man untersuchen, welche technischen Eingriffe in ein Spiel den Spielfluss negativ beeinflussen. Dort könnte man die alternativ vorgestellten Konzepte auf ihren Einfluss und daraus resultierend die Umsetzbarkeit prüfen.

Abbildungsverzeichnis

1.1 Aufbau des Timadorus Systems 3
1.2 Handeln über Tabellen im Computerspiel " Elite-Dangerous“ ElitePcGames (2015) 7

2.1 Verschachtelte Transaktionen 10

4.1 Erweiterung im Timadorus System 18
4.2 Nachrichtendiagramm einfache, verteile Transaktion 21
4.3 Nachrichtendiagramm einfache, verteile Transaktionen mit Abbruch 22
4.4 Nachrichtendiagramm komplexe, verteilte Transaktionen 23

5.1 Herkömmliches abgekapseltes Verfahren am Beispiel „Borderlands2“ 29

Literaturverzeichnis

[PokeBugsErsteGeneration2014] : *Pokemon Bugs in der ersten Generation.* – URL http://www.pokewiki.de/Bugs_in_der_ersten_Generation

[ElitePcGames 2015] : *Handelsfenster Elite Dangerous.* Januar 2015. – URL http://www.pcgames.de/screenshots/970x546/2015/01/Handelsfenster-pc-games.jpg

[Behnke] Behnke, Lutz: *Timadorus Webseite.* – URL http://timadorus-web2.informatik.haw-hamburg.de/de/project/

[Behnke 2014] Behnke, Lutz: *Subscription and Event Notification Modell for QuP.* internal PDF. January 2014

[Brewer 2000] Brewer, Eric A.: Towards robust distributed systems. In: *PODC*, 2000

[Coulouris u. a. 2002] Coulouris, G.F. ; Dollimore, J. ; Kindberg, T.: *Verteilte Systeme: Konzepte und Design.* Pearson Education Deutschland, 2002 (Informatik - Pearson Studium). – URL http://books.google.de/books?id=FfsQAAAACAAJ. – ISBN 9783827371867

[Grau 2003] Grau, Oliver: *Virtual art : from illusion to immersion.* Cambridge, Mass. and MIT Press, 2003. – ISBN 0-262-07241-6

[Haerder und Reuter 1983] Haerder, Theo ; Reuter, Andreas: Principles of Transaction-Oriented Database Recovery. In: *Computing Surveys* 15 (1983), December, Nr. 4

[Loetzsch u. a. 2013] Loetzsch, Markus ; Renate, Boeblin ; Bantele, Michael: Der Ehrbare Kaufmann. In: *IHK* (2013). – URL http://www.ihk-nuernberg.de/de/media/PDF/Publikationen/Hauptgeschaeftsfuehrung/Ehrbarer-Kaufmann.pdf

[Nieson 1993] Nieson, Jakob: *Usability Engineering.* 1. Edition. Academic Press inc., September 1993. – ISBN 978-0125184069

[Page und Nance 1994] Page, Ernest H. ; Nance, Richard E.: Parallel Discrete Event Simulation: A Modeling Methodological Perspective. In: *SIGSIM Simul. Dig.* 24 (1994), Juli, Nr. 1, S. 88–93. – URL http://doi.acm.org/10.1145/195291.182536. – ISSN 0163-6103

[Tanenbaum und Steen 2008] Tanenbaum, Andrew S. ; Steen, Maarten v.: *Verteilte Systeme, Prinzipien und Paradigmen.* 2., aktualisierte Aufl. Pearson Studium, 2008. – Andrew S. Tanenbaum ; Maarten van Steen. – ISBN 3-8273-7293-3

[Tolf und Johannessen] Tolf, Fredrik ; Johannessen, Bjoern: *Haven and Hearth Webseite.* – URL http://www.havenandhearth.com/portal/

[Vogels 2009] Vogels, Werner: Eventually Consistent. In: *Commun. ACM* 52 (2009), Januar, Nr. 1, S. 40–44. – URL http://doi.acm.org/10.1145/1435417.1435432. – ISSN 0001-0782

[Weikum und Vossen 2001] Weikum, Gerhard ; Vossen, Gottfried: *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery.* Elsevier, 2001

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

Hamburg, 21. April 2015 Frederik Klauß