Konzeption und Entwicklung eines Konferenzsystems auf Basis einer webbasierten Push-Architektur

Filipe Campos Santos
Große Studienarbeit
Wirtschaftsinformatik
fc004@hdm-stuttgart.de

Wolf-Fritz Riekert

Wirtschaftsinformatik
riekert@hdm-stuttgart.de

Abstract

Gegenstand der hier vorgestellten Arbeit ist eine Methode zur Realisierung einer asynchronen Informationsarchitektur im derzeitigen World Wide Web. Es wird am Beispiel eines webbasierten Konferenzsystems gezeigt, wie sich zeitlich ändernde Daten in Echtzeit und ohne periodisch wiederholte Anfragen (Requests) von Seiten des Clients visualisiert werden können.

Durch einen einmaligen Browser-Aufruf einer PHP-Seite wird eine permanente Verbindung zwischen dem PHP-Skript der aufgerufenen Seite und einem Push-Server aufgebaut. Über diese Verbindung können anschließend im Push-Server eingehende Daten sofort an das PHP-Skript zurückgesendet werden und im Browser ohne erneute Anfrage visualisiert werden. Als Übertragungsprotokoll wird dabei lediglich das im WWW gängige Hypertext Transfer Protocol (HTTP) verwendet, so dass das Verfahren auch hinter Firewalls genutzt werden kann. Der verwendete Browser benötigt für das Verfahren kein besonderes Plugin, er muss lediglich JavaScript unterstützen. Dadurch ist das Verfahren in praktisch allen gängigen Umgebungen einsetzbar.

Wie Messergebnisse zeigen, können mit dieser Push-basierten Methode nicht nur leistungsfähige Echtzeit-Webapplikationen entwickelt, sondern gegenüber den bisher üblichen Verfahren auch maßgeblich Ressourcen gespart werden.

Keywords: Push-Technik, Push-Server, Echtzeit, Web2.0, asynchrones Internet

1. Einleitung

Charakteristisch für das heutige Web sind die erleichterte Kommunikation, die Informationsverteilung und die Kollaboration. Die Inhalte werden nicht mehr nur zentralisiert von großen Medienunternehmen erstellt und über das Internet verbreitet, sondern auch von einer Vielzahl von Nutzern, die sich mit Hilfe sozialer Software untereinander verbinden. Im Zuge des Web2.0 sind so viele Anwendungen entstanden, welche die Interaktion mit digitalen Medien und mit anderen Benutzern erleichtert haben.

Das klassische Webparadigma bringt jedoch einen Nachteil mit sich. Der Client, in diesem Fall der Browser, erhält Daten vom Server nur synchron zu einem Request. Das bedeutet, dass jedes Mal wenn der Client ein Update der Daten benötigt, dieser den Server explizit anfragen muss. Diese Technik wird auch als „Pull-Technik" bezeichnet. Heutige Web-Server sind nicht in der Lage, Informationen unaufgefordert an den Client zu senden. Die Daten, die vom Server kommen und im Browser visualisiert werden, sind demnach statisch und ändern sich erst, wenn die Seite neu angefordert und geladen wird. (Alione, 2005)

Es entstehen jedoch immer mehr Webapplikationen, welche die Aktualisierung der Daten in Echtzeit benötigen. Aktienkurse in Online-Handelsbörsen, Live-Ticker auf Sport- oder Nachrichtenseiten oder Echtzeit-Updates von Auktionspreisen sind nur einige Anwendungsbeispiele, die, um ein Maximum an Benutzerfreundlichkeit und Qualität zu bieten, kontinuierliche Updates der visualisierten Daten im Browser benötigen. (Alione, 2005)

Um diese Probleme zu beheben, muss das heute vorliegende Paradigma von der synchronen zur asynchronen Informationsbereitstellung verändert werden. Um sowohl geringe Latenzzeiten als auch eine wirtschaftlichere Server- und Clientauslastung zu erzielen, ist der Übergang von der Pull-Technik zur so genannten Push-Technik nötig. So wird es möglich, dass Browser von äußeren Ereignissen ausgelöste Updates in einer asynchronen Art und Weise erhalten, ohne vorherige Anfragen an den Server starten zu müssen.

Die Push-Technik ist kein wirklich neues Prinzip für Netzwerkdienste; Blackberry-Mobilgeräte nutzen sie beispielweise schon seit mehr als 10 Jahren für die Zustellung von E-Mails. Obwohl die Push-Technik grundsätzlich auch im Web anwendbar ist, gibt es in der Praxis hierfür bislang kaum Beispiele. Dennoch ist die Push-Technik, vor allem im Web, ein hochaktuelles Thema.

So griff im Jahr 2010 das Mitgliedermagazin der Gesellschaft für Informatik diese Thematik auf und veröffentlichte einen Artikel, in dem eine Push-Architektur für Webapplikationen vorgestellt wird (Faßhauer, Hielscher & Wagenknecht, 2010). Trotz der durch diese Architektur erzielten Fortschritte bleiben jedoch wesentliche Einschränkungen durch Abhängigkeiten und spezielle Transportprotokolle bestehen, die im Folgenden noch aufgegriffen werden.

Dieses Papier zeigt, wie die Push-Technik mit dem Web-Standard-Protokoll HTTP auf einer besseren Art und Weise realisierbar werden kann und über die von den meisten Browsern unterstützte Web-Programmiersprache JavaScript hinaus keine weitere Ausstattung des Web-Browsers erfordert. Um dies zu demonstrieren, wurde ein webbasiertes asynchrones Konferenzsystem entwickelt, das die Praxistauglichkeit des Ansatzes demonstriert und deutlich messbare Effizienzsteigerungen gegenüber klassischen Lösungen aufweist.

2. Zielsetzung

Hauptziel dieser Arbeit ist es zu zeigen, wie eine asynchrone Informationsübertragung durch eine Push-Architektur im Internet ermöglicht werden kann. Es wird gezeigt, wie über das standardmäßig verwendete HTTP-Protokoll persistente Verbindungen aufgebaut werden können, sodass der Server Daten ohne vorherige Anfragen des Browsers an diesen senden kann.

Zur Veranschaulichung der Push-Architektur wird ein webbasiertes Konferenzsystem implementiert. Das Konferenzsystem soll dafür ausgelegt werden, dass mehrere Teilnehmer, die sich meist an unterschiedlichen Orten befinden, in einem virtuellen Raum über ein gemeinsames Problem diskutieren können. Die Kommunikation erfolgt durch die Übertragung von Textnachrichten und den Austausch von Dokumenten.

Da das Konferenzsystem auf der hier vorgestellten Push-Architektur basiert und somit die Interaktion asynchron verläuft, können Teilnehmer in Echtzeit Informationen austauschen. Neben der Möglichkeit, Textnachrichten an alle sich im selben virtuellen Raum befindlichen User zu versenden und zu empfangen, können auch PDF-Dokumente oder Bilder hochgeladen und den Konferenzteilnehmern in Echtzeit bereitgestellt werden. Abbildung 1 geht auf die sonstigen Funktions- und Interaktionsmöglichkeiten mit dem System ein.

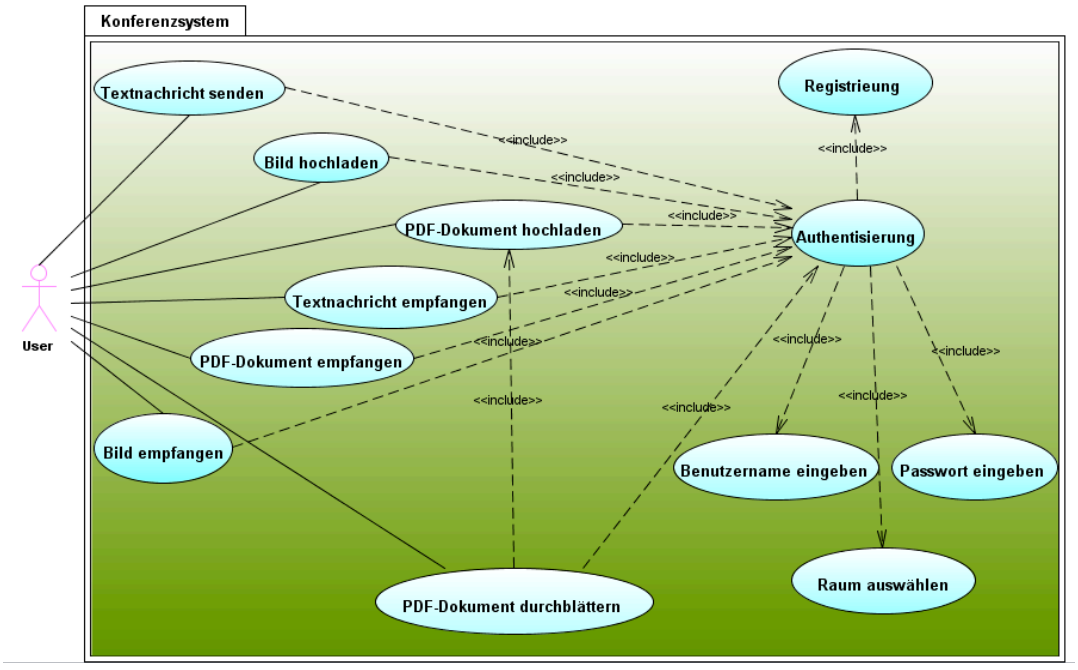

Abbildung 1: Use-Case

3. State-of-the-Art-Strategien

Um das Echtzeitkonferenzsystem zu programmieren, sind wesentliche Veränderungen im derzeit vorliegenden synchronen Informationsfluss des heutigen Webs nötig. Die nachfolgenden Unterkapitel erläutern zunächst, worin die Gründe wie auch die Probleme des synchronen Informationsaustausches liegen, wie konventionelle

Lösungen für die Entwicklung von asynchronen Echtzeitwebapplikationen aussehen und welche neuen Ansätze hierfür existieren.

3.1 Ausgangspunkt

Der klassische Stil des heutigen World Wide Webs basiert auf dem Hypertext Transfer Protocol, kurz HTTP. Das HTTP-Protokoll setzt auf TCP[1] auf und regelt die Kommunikation von Web-Servern und Browsern. (Kannengiesser, 2005, S. 41)

Obwohl HTTP auf dem verbindungsorientierten Protokoll TCP basiert, ist HTTP ein verbindungsloses Protokoll. (Kannengiesser, 2005, S. 41)

Diese Eigenschaft des HTTP-Protokolls ist für die Beschränkung verantwortlich, dass Informationen nicht asynchron bzw. serverseitig an die Browser kommuniziert werden können.

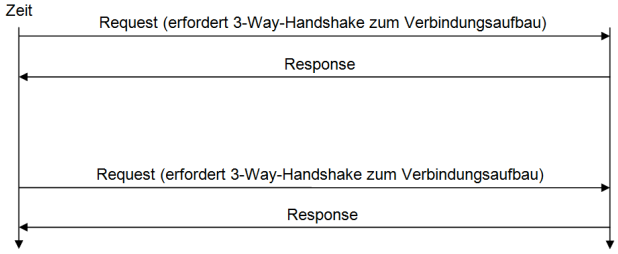

Abbildung 2: Funktionsweise des Hypertext Transfer Protocol

Für die Kommunikation setzen die Browser zunächst eine Anfrage (engl. Request) an den Web-Server ab. Hierfür wird durch einen 3-Way-Handshake, ein dreistufiges Verfahren zum Aufbau verlustfreier Datenübertragungen zwischen zwei Instanzen, eine Verbindung aufgebaut. Entsprechend diesem Verfahren sendet der Client ein Datenpaket mit einem Synchronisations-Flag (SYN) an den Server. Dieser bestätigt die SYN-Anfrage des Clients mit einem SYN/ACK. Daraufhin bestätigt der Client die SYN/ACK-Bestätigung mit einem Bestätigungs-Flag (ACK). Wenn der 3-Way-Handshake erfolgreich war, ist eine Verbindung zum Web-Server aufgebaut worden. Anschließend wird der Web-Server nach neuen Informationen gefragt und nach deren Übertragung die Verbindung zum Client serverseitig beendet. Aufgrund dieser abgebrochen Verbindung ist der Web-Server jedoch nicht mehr in der Lage, die Browser selbstständig über neue Daten zu informieren. Wenn der Internutzer ein Update der Daten benötigt, muss dieser den Web-Server ein weiteres Mal explizit anfragen (vgl. Abbildung 2).

[1] TCP (Transmission Control Protocol) ist ein verbindungsorientiertes Transportprotokoll. Vor der eigentlichen Übertragung wird eine Verbindung aufgebaut, indem als „Handshake" bezeichnete Kontrollinformationen zwischen beiden Endpunkten übertragen werden. (Hunt, 2003, S. 21)

Aufgrund dieser verbindungslosen Eigenschaft des HTTP-Protokolls entstehen Probleme bei der Realisierung des asynchronen Echtzeitkonferenzsystems. So müsste etwa der Browser ununterbrochen beim Server anfragen, ob neue Textnachrichten oder Dokumente von anderen Konferenzteilnehmern vorhanden sind.

3.2 Konventionelle Lösungen

Um trotz der verbindungslosen Eigenschaft des HTTP-Protokolls Webapplikationen zu entwickeln, die dem User einen asynchronen Informationsfluss suggerieren, wurde bisher entweder versucht, die Informationen in regelmäßigen Abständen automatisch zu holen oder durch Zusatzsoftware das HTTP-Protokoll zu umgehen.

3.2.1 Polling

Die Polling-Technik ist eine Möglichkeit dem Internet-User die Asynchronität des Informationsflusses zu suggerieren. Polling ist eine Technik, womit beim Web-Server in regelmäßigen kurzen Abständen, wie in Abbildung 1, automatisch angefragt wird, ob neue Daten vorliegen (Jäger, 2007, S. 330).

Mit Hilfe von Ajax[2] kann die Suggerierung der Asynchronität noch verstärkt werden, da für das Anzeigen der Informationen der Browser die Seite nicht vollständig neu laden muss. Es werden lediglich einzelne Bereiche der Seite neu gefüllt.

Für die Entwicklung des Konferenzsystems könnte man demnach eine Webapplikation schreiben, die in regelmäßig kurzen Abständen automatisch über Ajax den Web-Server nach neuen Daten fragt. Für den Browser würde diese Methode keine größeren Probleme bereiten. Jedoch wäre diese Methode auf Serverseite mit großen Problemen verbunden. Wenn mehrere Clients mehrmals pro Sekunde einen 3-Way-Handshake durchführen würden, um eine Verbindung aufzubauen, die nach der Datenübertragung wieder beendet wird, würde dies zu einer sehr großen Serverlast führen.

Die folgende Auflistung geht auf sonstige Probleme ein, die mit der Polling-Technik assoziiert werden.

- Es herrscht ein Konflikt zwischen der Latenzzeit und der Auslastung des Servers. D.h., je geringer die Latenzzeit der empfangenen Informationen sein soll, desto kleiner müssen die Abstände zwischen den regelmäßigen Anfragen sein. Jedoch ist die Auslastung

[2] Ajax steht für Asynchronous JavaScript and XML. Konkret bedeutet das, dass die Daten bei Ajax im Hintergrund – für den Benutzer nicht merkbar – übertragen werden. Damit können auch einzelne Bereiche einer Webseite durch von der Serverseite abgerufene Informationen aktualisiert werden, ohne dass dafür die Seite ganz neu aufgebaut werden muss. (Maurice, 2007)

des Servers umso größer, je kleiner die Abstände zwischen den Anfragen sind.
- Aufgrund des weiterhin existierenden synchronen Paradigmas ist es nur bedingt möglich, Daten in Echtzeit zu empfangen. Selbst bei extrem klein gewählten Abständen zwischen den Anfragen kommt die Latenzzeit der ankommenden Informationen dem Idealwert nie so nahe wie in einer „echten" asynchronen Architektur.
- Heute vorhandene Web-Server bieten keine Funktionalität dafür, automatisch neue Informationen zu erkennen und diese einer Webapplikation zu signalisieren. Informationen müssten deshalb bis zur Anfrage des Browsers, zum Beispiel in einer Textdatei oder einer Datenbank, zwischengespeichert werden. Periodische Anfragen würden demnach nicht nur für eine große Auslastung des Web-Servers, sondern auch des Dateisystems oder der Datenbank sorgen.

Durch die nötigen Anfragen, um zu erfahren, ob neue anzuzeigende Informationen vorhanden sind, entsteht eine hohe Serverauslastung, selbst wenn sich das System im Leerlauf befindet.

3.2.2 Plug-ins

Ein Plug-in ist ein Zusatzprogramm, welches sich in ein anderes Softwareprodukt, in diesem Fall den Browser, „einklinkt" und dadurch dessen Funktionalität erweitert (vgl. Levine, 2007, S.119).

Da heutige Web-Browser ausschließlich mit dem verbindungslosen HTTP-Protokoll arbeiten, versucht man dieser Einschränkung durch entsprechende Plug-ins wie dem Flash-Plug-in[3] oder dem Java-Plug-in, das den Betrieb sogenannter Java-Applets[4] erlaubt, entgegenzuwirken. Diese Plug-ins erweitern unter anderem die Funktionalität des Browsers hinsichtlich der Möglichkeit, andere Transportprotokolle zu nutzen und die Begrenzungen des HTTP-Protokolls zu umgehen.

Plug-ins bieten somit die Möglichkeit, den in Kapitel 3.2.1 erläuterten Ansatz zur Suggerierung eines asynchronen Datenflusses zu umgehen.

Da Plug-ins weiterhin, im Gegensatz zu Web-Browsern, die Eigenschaft besitzen, sich mit verschiedensten Servertypen zu verbinden, sind diese nicht von den Restriktionen heute vorhandener Web-Server abhängig.

[3] Flash: Flash ist ein proprietäres Format zur Darstellung multimedialer Inhalte. Die Flash-eigene Programmiersprache „ActionScript" erlaubt die Erstellung komplexer Websites und die Programmierung browserbasierter Anwendungen. (Hammer & Bensmann, 2009, S. 101)

[4] Applets: Ein Applet ist ein in Java geschriebenes Programm, das in HTML-Dokumente eingefügt werden kann. (Sun, o.J.)

In dem in der Einleitung erwähnten Artikel aus dem Jahr 2010 stellen die Autoren Faßhauer, Hielscher und Wagenknecht eine Push-Architektur vor, die auf Plug-ins basiert. Nachteilig an der dort vorgestellten Architektur ist jedoch, dass zunächst nur der von Mozilla hergestellte Firefox-Browser unterstützt wird. Darüber hinaus muss der Endbenutzer ein Plug-in für den Firefox installieren, bevor Informationen servergesteuert und ohne vorherige Anfragen übertragen werden können (Faßhauer et al., 2010). Plug-ins sind nämlich nicht nativ installiert. Sie müssen erst geladen, installiert und ausgeführt werden. All dies kostet Leistung und ist aufgrund der Tatsache, dass dem User eine zusätzliche Software aufgezwungen wird, einschränkend.

Ein weiterer Nachteil, der sich aus dem Einsatz von Plug-ins ergibt, besteht darin, dass sich auch eine Gefahr in der Ausführung solcher Zusatzprogramme ergibt. Firewalls könnten aufgrund der Tatsache, dass für die Benachrichtigung der Web-Clients (Browser) nicht das HTTP-, sondern das UDP-Protokoll verwendet wird, jederzeit den Transport der Daten blockieren. „Grenzen des beschriebenen Verfahrens werden deutlich, sobald ein Administrator […] das UDP-Protokoll deaktiviert hat. Auch das Sperren des Signalserver-Ports verhindert den Signalaustausch. Bei unidirektionalen Firewalls ist eine sorgfältige Konfiguration erforderlich, da sie im Grundzustand die Umkehrung der Senderichtung eines UDP-Kanals blockieren. Auch Proxies und VPN-Tunnels […] beeinträchtigen unsere Implementierung." (Faßhauer et al., 2010, S. 619)

Für die Entwicklung des Konferenzsystems könnte ebenfalls ein entsprechendes Plug-in programmiert werden, um eine direkte Verbindung zwischen dem Web-Server und dem Web-Client (Browser) aufzubauen. Alle in Kapitel 3.2.1. erwähnten Probleme wären zwar gelöst, die erläuterten Nachteile, die durch Plug-ins vorhanden sind, wären jedoch weiterhin vorhanden.

3.3 Neue Ansätze

Um die Probleme, die in Kapitel 3.2.1 erwähnt wurden, und die Einschränkungen von Plug-ins und anderen Transportprotokollen zu überwinden, muss demnach ein Weg gefunden werden, das HTTP-Protokoll hinsichtlich seiner Funktionsweise so zu verändern, dass aus der verbindungslosen Funktionsweise eine verbindungsorientierte Lösung entsteht. Da HTTP außerdem, wie in Kapitel 3.1 bereits erwähnt, auf dem verbindungsorientierten TCP-Protokoll basiert, dürfte es möglich sein, die durch das HTTP-Protokoll vorgeschriebenen Verbindungstrennungen seitens des Web-Servers nach der Informationsübermittlung zu verhindern.

Nichtsdestotrotz sollten, um Ressourcen zu sparen und um Risiken zu minimieren, zusätzliche Softwarekomponenten wie Plug-ins vermieden werden.

Die nachfolgenden Unterkapitel gehen eine Reihe von Möglichkeiten durch, um eine derartige verbindungsorientierte Lösung zu schaffen.

3.3.1 Long-Polling

Die Long-Polling-Technik ist der Polling-Technik ähnlich. Mit Long-Polling wird jedoch versucht, das Problem der entstehenden Auslastung auch bei nicht vorhandenen neuen Informationen zu umgehen.

Mit Long-Polling wird eine persistente Verbindung geöffnet und so lange offen gehalten bis neue Information vorhanden sind (vgl. Abbildung 3). Nach der Übertragung der neuen Informationen wird serverseitig die Verbindung beendet. Der Browser eröffnet im Anschluss zum wiederholten Mal eine persistente Verbindung und wartet auf neue Informationen (vgl. Arcand, 2007).

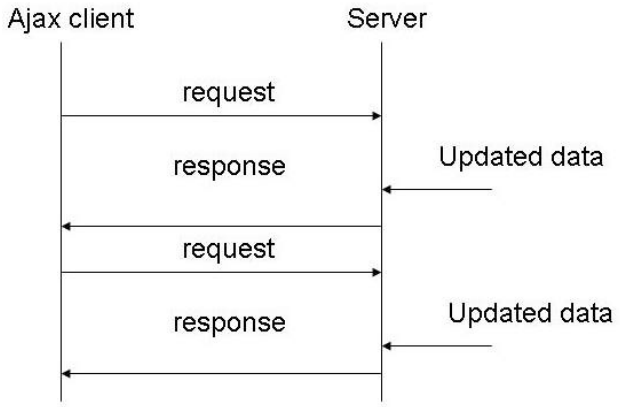

Abbildung 3: Funktionsweise der Long-Polling-Technik (Quelle: Arcand, 2007)

3.3.2 HTTP-Stream

Mit der HTTP-Stream-Technik wird der Long-Polling-Ansatz mit seinen persistenten Verbindungen erweitert. Hier wird jedoch, im Gegensatz zu den bereits vorgestellten Techniken, nur noch eine einzige persistente Verbindung über die gesamte Sitzung hinweg verwendet (vgl. Arcand, 2007).

Zunächst sendet der Browser eine Anfrage an den Web-Server. Der Web-Server verarbeitet die Anfrage und schickt dem Browser die von ihm geforderten Daten. Nach der vollständigen Übertragung der Daten wird die Verbindung serverseitig jedoch nicht beendet. Sie wird für zukünftige Informationsanfragen seitens des Browsers offen gehalten. Hierdurch kann die Auslastung des Web-Servers und des Browsers stark minimiert werden, da für die Übertragung neuer Daten nicht jeweils immer neue Verbindungen auf- und wieder abgebaut werden müssen (vgl. Abbildung 4).

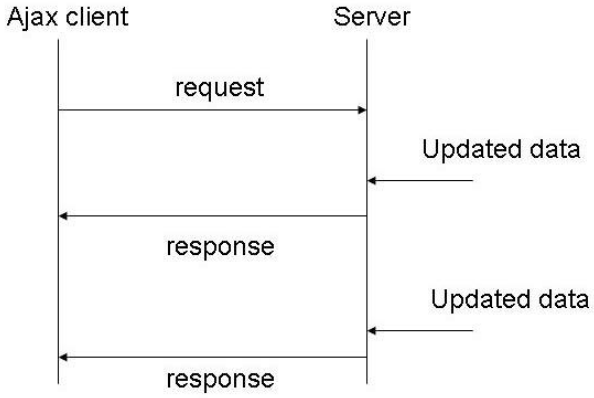

Abbildung 4: Funktionsweise der HTTP-Stream-Technik (Quelle: Arcand, 2007)

Betrachtet man die Probleme, die es mit der einfachen Polling-Technik gibt, so lösen sich mit diesem Ansatz einige der in Kapitel 3.2.1 erläuterten Probleme. In der Tat kommt diese Lösung dem optimalen Zustand schon sehr nahe.

Die Probleme mit den heute vorhandenen HTTP-Web-Servern, die nicht in der Lage sind, neue Informationen zu erkennen und diese an alle aktiven Instanzen einer Webapplikation zu schicken, verbleiben jedoch weiterhin. Daher werden die periodischen Datenabfragen auch bei Verwendung der HTTP-Stream-Technik nicht überflüssig.

3.3.3 Die Push-Technik

Die Push-Technik ist de facto nicht neu. Als Desktop-Programme sind solche Push-Techniken schon seit Anbeginn des Internets vorhanden. Jedoch erfordern diese die Installation eines Clients, welcher mit dem Server des Anbieters kommuniziert.

Die Push-Technik beschreibt eine Methode, meist realisiert über eine verbindungsorientierte TCP-Verbindung, bei der Informationen von einem zentralen Server an die Clients ausgeliefert werden. Da jedoch das Web als am häufigsten verwendeter Internet-Dienst, wie in Kapitel 3.1 erläutert, auf dem verbindungslosen HTTP-Protokoll basiert, muss versucht werden, die Push-Technologie an dieses Protokoll anzupassen.

Betrachtet man die standardmäßige Funktionsweise des HTTP-Protokolls, dargestellt in Abbildung 1, müssen folgende Eigenschaften geändert werden:

1. Der Verbindungsabbau nach dem Übermitteln der Informationen soll verhindert werden.
2. Neue Informationen sollen automatisch erkannt und ohne vorherigen Request über die persistente Verbindung übermittelt werden.

Die nachfolgende Skizze verdeutlicht den Gedanken hinter der Push-Technik.

Die Verhinderung des Verbindungsabbaus und die Erzeugung einer permanenten Verbindung können mit der bereits in Kapitel 3.3.2 eingeführten HTTP-Stream-Technik umgesetzt werden. Dieser Ansatz wird jedoch durch einen Push-Server erweitert. Das ist ein Server, der eigenständig neue Informationen erkennt und diese an alle aktiven Instanzen der Webapplikation über den Web-Server weitermeldet. Der Push-Server ist zwingend erforderlich, da heutige Web-Server nicht in der Lage sind, neue Informationen zu erkennen.

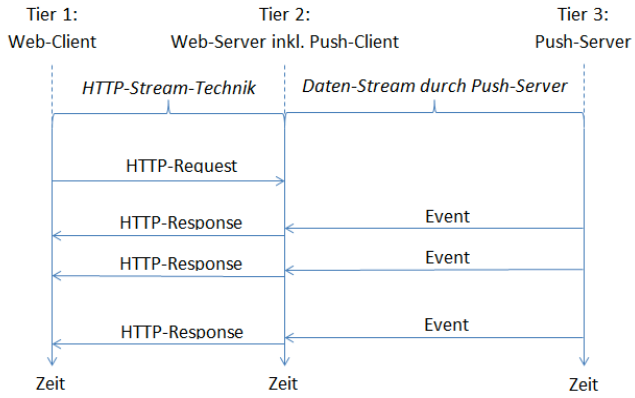

Abbildung 5: Der Push-Ansatz

Somit ergibt sich die in

Abbildung 5 dargestellte 3-Tier-Architektur (dt. 3-Schichten-Architektur):

- Der erste Tier besteht aus dem Browser. Er ist zugleich der Client für die Kommunikation mit dem zweiten (mittleren) Tier über das HTTP-Protokoll.
- Der zweite (mittlere) Tier besteht aus dem Web-Server. Dieser Tier ist sowohl der Server für die HTTP-Kommunikation mit dem ersten Tier als auch der Client für die Kommunikation über eine TCP-Verbindung mit dem dritten Tier.
- Der dritte Tier besteht aus einem Push Server. Dieser erkennt, wenn ein neues Ereignis stattfindet (im Konferenzsystem beispielsweise wenn eine neue Nachricht oder ein neues Dokument zur Weitergabe ansteht) und benachrichtigt alle auf solche Ereignisse wartenden und mit dem Web-Server verbundenen Browser.

Die Push-Technik bringt zahlreiche Vorteile mit sich. Neben der Lösung der Request/Response-Situation sind auch keine Risiken und Beschränkungen, wie sie es mit dem Einsatz von Plug-ins gibt, vorhanden. Da diese Lösung weiterhin lediglich auf Serverseite agiert und der Web-Server weiterhin HTTP-konform Informationen an den Browser des Endnutzers sendet, bereiten eventuell dazwischengeschaltete Firewalls oder Anti-Viren-Programme keine Probleme.

Die Push-Technologie bietet das Potential, das aktuell vorherrschende Paradigma des synchronen Informationsflusses zu ändern, ohne jegliche Änderungen auf Seiten des Browsers vornehmen zu müssen. Wie die HTTP-Stream-Technik und der Push-Server umgesetzt werden können und wie die Funktionsweise der gesamten Architektur im Genaueren aussieht, soll im Folgenden erklärt werden.

4. Das Konzept

Das Konzept umfasst in erster Linie die Sicherstellung der in Kapitel 3.3.3 erwähnten zwei Punkte zur Anpassung des Response/Request-Verhaltens des HTTP-Protokolls.

Dies wird mit Hilfe einer HTTP-Stream-Verbindung zwischen Tier 2 und Tier 1 realisiert. Neben der Realisierung der HTTP-Stream-Technik müssen die Daten, die durch die permanente Verbindung in den Browser gelangen, mit JavaScript clientseitig sortiert und jeweils an die richtige Stelle und ohne vorherige Anfragen auf der Seite positioniert werden.

Die Konzeption von Tier 3 umfasst die Realisierung eines Push-Servers, der neue Informationen automatisch erkennt und dem Web-Server in Tier 2 automatisch - zur Weitergabe an die verbundenen Browsern - sendet.

4.1.1 Konzeption der HTTP-Stream-Technik zum Aufbau einer permanenten Verbindung zwischen Tier 2 und Tier 1

Um eine permanente Verbindung zwischen Tier 2 und Tier 1 zu realisieren wird mit jeder durch den Browser gestarteten Anfrage eine Socket[5]-Verbindung zwischen dem Web-Server (Tier 2) und dem Push-Server (Tier 3) aufgebaut. Über diese Verbindung erhält der Web-Server neue Informationen, die er über die zuvor aufgebaute HTTP-Stream-Verbindung an den Browser weiterleitet.

Zur Programmierung der Socket-Verbindung zwischen dem Web-Server und dem Push-Server wurde die Sprache PHP gewählt. PHP steht für Hypertext Preprocessor und ist eine Interpretersprache, die hauptsächlich zur Erstellung dynamischer Webseiten oder Webanwendungen verwendet wird. Der Grund warum die Wahl auf diese Sprache fiel ist, dass sie weit verbreitet ist und sie sich deshalb gut dafür eignet, die Anwendbarkeit der Kombination von HTTP-Stream- und Push-Technik zu demonstrieren.

[5] Sockets sind der De-Facto Standard einer Programmierschnittstelle für die Kommunikation über z.B. TCP. (Müller, 2002, S.132)

Die Aufrechterhaltung einer permanenten Verbindung zur Realisierung der HTTP-Stream-Technik ist ein zentraler Punkt in dieser Arbeit und wird durch die Ausführung einer While-Schleife erreicht.

Das Besondere an dieser While-Schleife ist, dass sie nur durchläuft, wenn Daten durch die Socket-Verbindung ankommen. So lange die durch das PHP-Skript aufgebaute Socket-Verbindung besteht, verweilt die While-Schleife im Wartemodus. Erst bei eingehenden Informationen wird diese aktiv und läuft einmal durch, um anschließend wieder darauf zu warten, dass neue Informationen über die aufgebaute Socket-Verbindung eingehen. Abbildung 6 verdeutlicht die Funktionsweise.

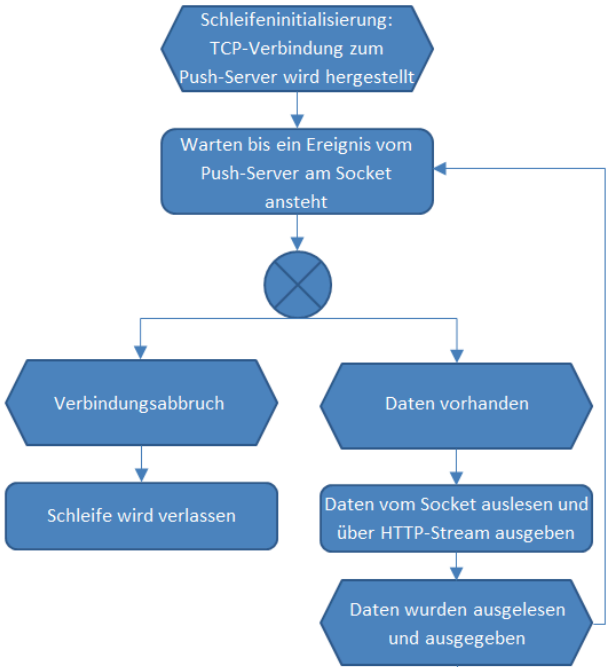

Abbildung 6: Darstellung der Funktionsweise der Socket-Schleife zum Aufbau einer permanenten Verbindung

Da, wenn einmal die Socket-Verbindung zum Push-Server aufgebaut wurde, die While-Schleife folglich nie verlassen wird, solange der Endbenutzer den Web-Browser nicht beendet, verbleibt die Seite mit diesem PHP-Skript im PHP-Interpreter. Dies führt dazu, dass der Web-Server (Tier 2) die zuvor aufgebaute Verbindung zum Web-Client (Tier 1) zur Beantwortung der durch den Endbenutzer initiierten HTTP-Anfrage nicht schließt, sondern kontinuierlich die durch die Socket-Verbindung eingehenden Informationen ausgibt. Es entsteht eine permanent ladende Seite und dadurch die gewünschte permanente Verbindung zwischen dem Client (Browser) und dem Web-Server. Erst wenn der Endbenutzer die Webseite der Konferenz verlässt, kommt es zum Verbindungsabbruch der zuvor aufgebauten TCP-Verbindung. Daraufhin erkennt der Apache-Web-Server, dass er keine Verbindung mehr zum Browser hat, und beendet schließlich den PHP-Prozess. Nach der Freigabe hat der Web-Server die Möglichkeit, die Seite komplett zu übertragen und die Verbindung zu beenden.

4.1.2 Einsatz von Ajax

Der Begriff Ajax wurde von Jesse James Garrett erfunden und ist im Grunde genommen keine neue Technologie. Garrett definiert Ajax als das Zusammenwirken von mehreren Technologien zu einem sinnvollen Ganzen (Garrett, 2005).

Ajax bezeichnet ein Konzept der asynchronen Datenübertragung zwischen einem Server und dem Browser, welches ermöglicht, innerhalb einer HTML-Seite eine HTTP-Anfrage durchzuführen, ohne die Seite komplett neu laden zu müssen (Richter & Koch, 2007).

Die Ajax-Funktionalität wird durch die Ajax-Engine in Tier 1 bereitgestellt und ist durch HTML- und JavaScript-Code realisiert.

Für die hier vorgestellte Applikation wird Ajax verwendet, um vom Browser aus im Hintergrund die Verbindung zum in PHP geschriebenen Push-Client auf dem Web-Server aufzunehmen. Dieser in PHP geschriebene Push-Client baut dann schließlich über TCP eine Verbindung mit dem Push-Server auf. Sobald Daten über die aufgebaute Verbindung eingehen, werden diese mit der DOM[6]-Technik im Frontend an die richtige Stelle platziert. Dadurch werden keine neuen Anfragen zur Anzeige neuer Daten benötigt und der störende Hinweis, dass der Browser ständig lädt, vermieden. Für den User bleibt es jetzt verborgen, ob im Hintergrund eine permanente Verbindung aufgebaut wurde oder nicht. Abbildung 7 verdeutlicht diesen Vorgang.

Ein weiterer Vorteil ist die Tatsache, dass durch den Einsatz von Ajax sichergestellt werden kann, dass die persistente Verbindung durch ein neues Laden der Seite nicht abgebrochen wird. Würde man die komplette Seite neu laden müssen, um die aktualisierten Informationen zu visualisieren, könnte der Fall eintreten, dass genau während diesem Neuaufbau der Seite neue Daten auf dem Server eingegangen sind, die nicht gepusht werden konnten, da kurzzeitig keine Verbindung zwischen dem Web-Server und dem Client (Browser) bestand. Neben der Möglichkeit, neue Daten ohne den Neuaufbau von Seiten anzuzeigen, ist es über Ajax genauso möglich, Daten, die in ein Formular eingegeben wurden, im Hintergrund zu verarbeiten.

[6] Das Document Object Model ist eine plattform- und sprachneutrale Schnittstelle, die es Programmen und Skripten erlaubt dynamisch auf die Struktur, den Inhalt und dem Still von Dokumenten zuzugreifen und deren Inhalt zu aktualisieren. (W3C, 2005)

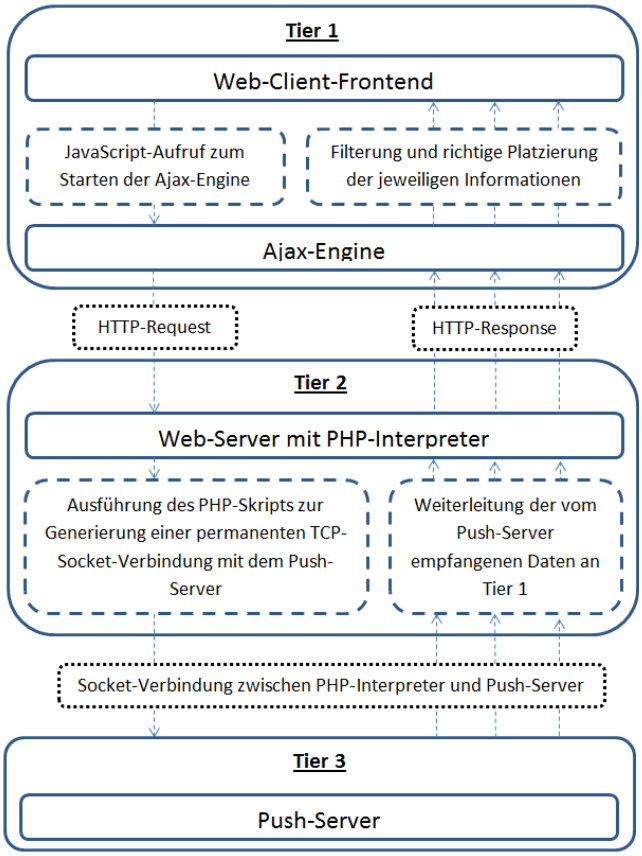

Abbildung 7: Zusammenspiel der einzelnen Tiers (dt. Schichten)

Dieser Ansatz wird hier z.B. verwendet, um Chatnachrichten an den Server zu senden, damit dieser die Nachrichten an alle mit dem Server verbundenen Konferenzteilnehmer weiterleiten kann.

4.2 Konzeption von Tier 3

Das Verhindern der nötigen Anfragen seitens des Browsers erreicht man mit der Programmierung eines in Java geschriebenen Push-Servers.

Die Idee ist, einen Server zu programmieren, welcher

- Socket-Verbindungsanfragen der PHP-Skripte akzeptiert,
- neue Clients jeweils als eigenständige Threads verwaltet und diese in einem Array aufnimmt,
- in der Lage ist, neu eintretende Ereignisse zu erkennen, um anschließend diese an alle im Array registrierten Clients zu melden,
- falls die Verbindung zu einem Client beendet wurde, den entsprechenden Client aus dem Array entfernt.

4.3 Das Architekturkonzept

Der Push-Server wurde in Java programmiert und hat die Aufgabe, Änderungen zu erkennen und diese über den PHP-Interpreter und dem Apache-Webserver dem Client mitzuteilen. Hierzu erzeugt der Apache-Webserver in Zusammenarbeit mit dem PHP-Interpreter eine Instanz des PHP-Skripts pro Client und baut für jeden Client eine eigenständige permanente Socket-Verbindung mit dem Push-Server auf. Über diese Verbindung ist es dann möglich, neue Ereignisse, die der Push-Server erkannt hat, an die zuständigen PHP-Skripts weiterzuleiten, damit diese die Daten http-konform über den Apache-Server an die jeweiligen Clients melden.

Abbildung 8 zeigt und erläutert die gesamte Architektur.

Auf Serverseite interagieren der Web-Server, das PHP-Skript und der Push-Server, realisiert in Java, miteinander. Auf der Clientseite wird das PHP-Skript zum Aufbau der Socket-Verbindung durch die Ajax-Engine, wie in Kapitel 4.1.2 erläutert, gestartet.

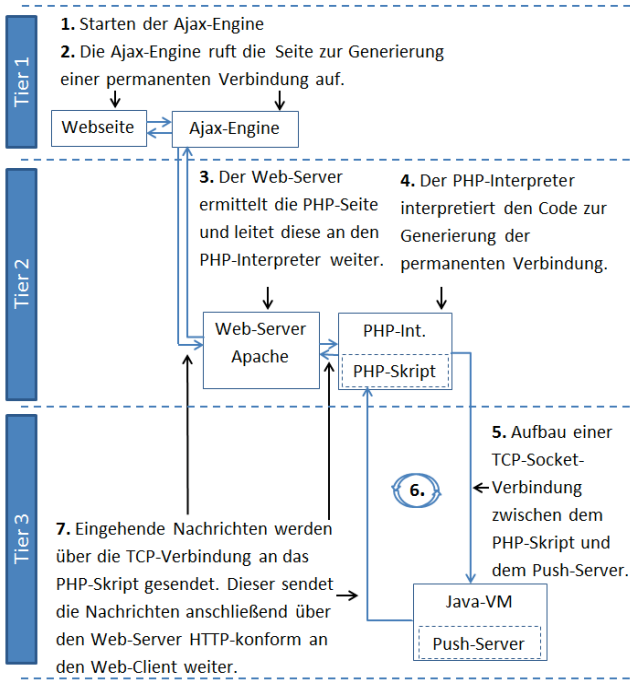

Abbildung 8: Darstellung der gesamten Architektur

Der schematische Ablauf sieht wie folgt aus:

1. Der Browser startet die Ajax-Engine.
2. Die Seite mit dem PHP-Befehl, eine Socket-Verbindung mit dem Push-Server aufzubauen, wird von der Ajax-Engine im Hintergrund aufgerufen.
3. Eine HTTP-Verbindung mit dem Web-Server wird aufgebaut und ein einmaliger Request erfolgt.
4. Der PHP-Interpreter interpretiert den PHP-Anteil der geforderten Seite.
5. Der PHP-Interpreter erzeugt eine Instanz für diese Verbindung und interpretiert den PHP-Code zur Erzeugung einer Socket-Verbindung mit dem Push-Server und zur Generierung der in Kapitel 4.1.1 erläuterten Programmschleife, um eine permanente Verbindung aufzubauen. Der Push-Server akzeptiert diese Verbindung, speichert den Client mit seinen

Parametern in einem Array und gibt über eintretende Ereignisse Bescheid.

6. Solange die Socket-Verbindung besteht, wird die Socket-Schleife, erläutert in Kapitel 4.1.1, nicht verlassen. Die angeforderte Seite verweilt im PHP-Interpreter. Dies führt dazu, dass der Web-Server die Verbindung zum Client nicht beendet, sondern kontinuierlich versucht, die Seite vollständig zu übertragen. Eine permanente Verbindung zum Streamen neuer Daten ist jetzt aufgebaut.
7. Wenn neue Informationen eintreffen, erkennt der Push-Server diese und schickt die neuen Informationen mit Hilfe des PHP-Interpreters über den Apache-Web-Server HTTP-konform an die Ajax-Engine. Die Ajax-Engine filtert anschließend die eingehenden Informationen und platziert sie mit Hilfe von DOM auf der Webseite. Neue Ereignisse werden über die bereits aufgebaute Verbindung ohne weitere Anfragen an den Client weitergeschickt.

und benötigen keine detaillierte Darstellung in diesem Kapitel.

Für die Installation wird das Serverinstallationspaket XAMPP[7] verwendet. Das Paket besteht aus dem Apache-Web-Server, aus der Interpretersprache PHP und aus dem relationalen Datenbanksystem MySQL.

Zusätzlich zu den vom Serverinstallationspaket installierten Technologien ist das Kernstück der gesamten Architektur, der Push-Server, nötig. Auf dessen Programmierung soll in diesem Kapitel eingegangen werden.

Auf Tier 2 sind speziell für dieses Projekt zugeschnittene PHP-Skripte mit den Socket-Schleifen, erwähnt in Kapitel 4.1.1, zu realisieren und weiterhin zur besseren Gestaltung und Darstellung verschiedene JavaScripts bzw. Ajax-Implementationen notwendig.

Auf die JavaScript- und Ajax-Implementationen wird in diesem Kapitel nicht eingegangen, da diese nur eine

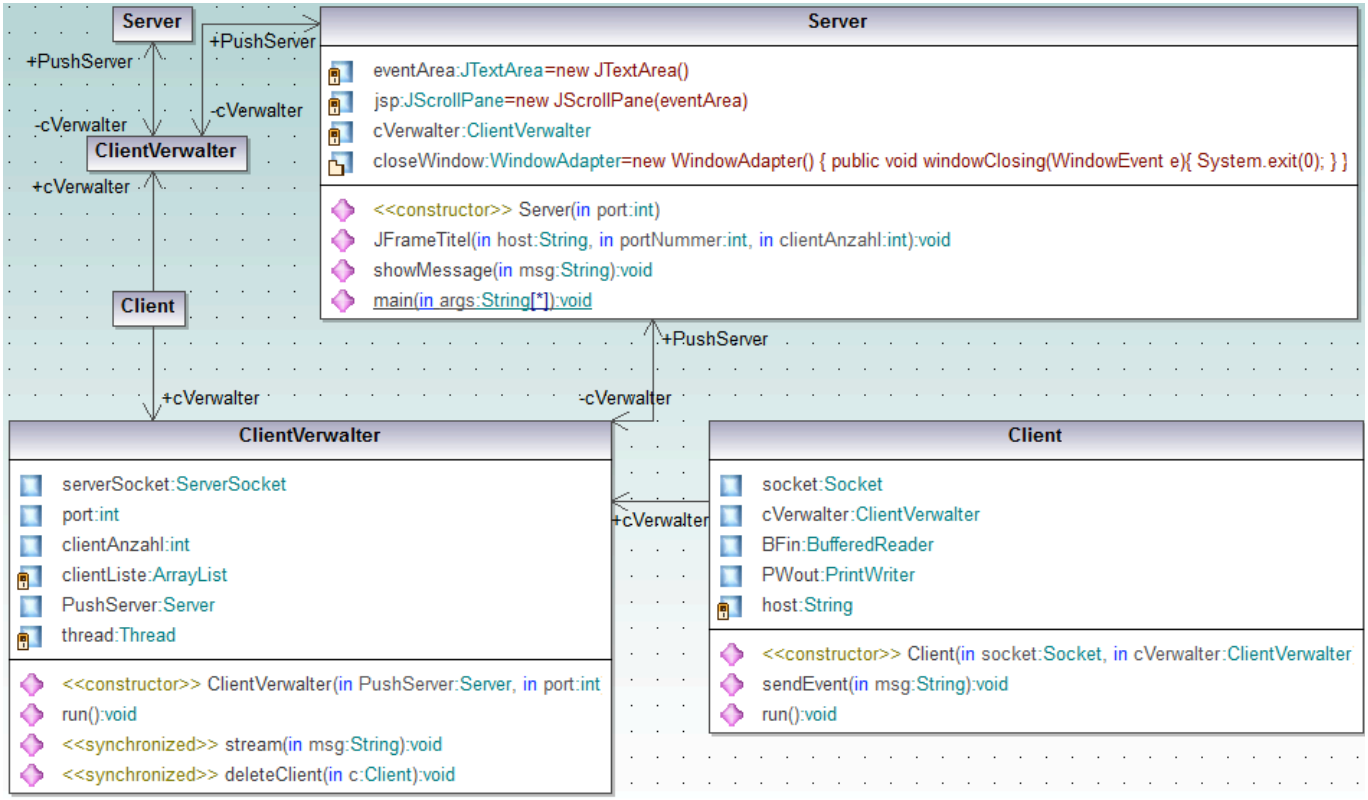

Abbildung 9: Package-Übersicht Push-Server

5. Realisierung des Konferenzsystems auf Basis der Push-Architektur

Da die Push-Architektur keine komplett neue Architektur ist, sondern wie in Kapitel 4 dargestellt, zum Großteil auf bereits bekannten Technologien basiert, werden diese mit Hilfe eines vorkonfigurierten Installationspakets installiert

unterstützende Funktion ausüben. Der clientseitige Teil dieses Kapitels konzentriert sich lediglich auf die zentralen PHP-Skripte zum Aufbau einer persistenten Verbindung zwischen dem Server und dem Browser.

[7] XAMPP ist eine Distribution zur Installation von Apache, MySQL und PHP. (Apache Friends, o.J.)

5.1 Einrichtung der Servers

Nach der Installation muss die PHP-Konfigurationsdatei angepasst werden. Da von Seiten des PHP-Interpreters für jeden Client eine permanente Socket-Verbindung mit dem Push-Server initialisiert wird, müssen die entsprechenden Einträge in der PHP-Konfigurationsdatei „php.ini" freigeschaltet werden.

Da es sich weiterhin um länger laufende PHP-Skripte handelt, muss der Wert der Variablen „max_execution_time" auf „-1" gesetzt werden, um für einen unbestimmten Zeitraum die nötigen Socket-Skripte laufen lassen zu können.

5.2 Der Push-Server

Der Server wurde in Java realisiert, besteht zur Gewährleistung einer besseren Übersichtlichkeit aus drei Klassen und setzt die in Kapitel 4.2 erläuterten Anforderungen um (vgl. Abbildung 9). Die Klassen werden im Folgenden beschrieben.

5.2.1 Die Klasse „ClientVerwalter"

Die Klasse „ClientVerwalter" verwaltet alle Socket-Verbindungen. Sie akzeptiert neue Clients, speichert diese als Client-Objekte in einem Array und löscht diese bei Bedarf wieder. Weiterhin ist diese Klasse für das Broadcasten von Nachrichten an alle Clients, welche sich im Array befinden, verantwortlich. Damit der Server neue, von den PHP-Skript-Instanzen eingehende Socket-Verbindungsanfragen akzeptiert, wird ein Server-Thread gestartet, welcher auf Verbindungsanfragen reagiert.

```
//Hält nach Clients Ausschau und fügt sie der
Arrayliste hinzu

public void run() {
 while(true) {
  try {
   Socket socket=serverSocket.accept();
   clientListe.add(new Client(socket, this));
  }
  catch(Exception e){
  }
 }
} …

…

//Nachricht an alle sich im Array befindlichen
Clients schicken

public synchronized void stream(String msg) {
 PushServer.showMessage(msg);
 for(int i=0; i < clientListe.size(); i++) {
  Client client=(Client)clientListe.get(i);
  client.sendEvent(msg);
 }
}

//Client aus der Array-Liste entfernen

public synchronized void deleteClient(Client c) {
 try{
     c.BFin.close();
     c.PWout.close();
     c.socket.close();
     for(int i=0;i<clientListe.size();i++) {
      if(((Client)clientListe.get(i)).equals(c)){
       clientListe.remove(i);
      }
     }
     clientListe.trimToSize();
     clientAnzahl--;
     PushSerer.JFrameTitel(serverSocket.getInet
     Address().getLocalHost().getHostName(),
     port, clientAnzahl);
    }
    catch(Exception e) {
    }
   }
  }
…
```

5.2.2 Die Klasse "Client"

Die Klasse „Client" ist eine Collections-Klasse[8], von der es für jede Verbindung eine Instanz als Client-Objekt gibt. Durch das Aufrufen des Konstruktors werden die nötigen Member-Variablen zur Verwaltung jedes einzelnen Clients mit Werten belegt.

Diese Klasse erzeugt für jeden Client eine eigene Instanz als Thread und bietet der Klasse „ClientVerwalter" die nötigen Methoden. Zu den wichtigsten Methoden gehören die „sendEvent()"-Methode, um ein Ereignis an das jeweilige Client-Objekt zu schicken, und die „run()"-Methode, um Ereignisse zu empfangen.

```
// Sendet Ereignis an ein Client-Objekt
public void sendEvent(String msg) {
 try {
  PWout.println(msg);
 }
 catch(Exception e) {
  cVerwalter.deleteClient(this);
 }
}
…

// Ereignis empfangen
public void run() {
 String inStream="";
 try {
  while((inStream = BFin.readLine()) != null) {
   cVerwalter.stream(inStream);
   inStream="";
  }
  PWout.close();
  BFin.close();
  socket.close();
 }
```

[8] Eine Collections-Klasse ist eine Klasse zur Sammlung von gleichartigen Objekten.

```
 catch(Exception e) {
 }
 cVerwalter.deleteClient(this);
} …
```

5.2.3 Die Klasse "Server"

Die Klasse „Server" enthält die Main-Methode. Sie startet die Applikation und erstellt die graphische Benutzeroberfläche.

```
…
//Main Methode starten
public static void main(String [] args){
 int port;
 try{
  port=Integer.parseInt(args[0]);
 }
 catch(Exception e){
  port=2000;
 }
 new Server(port);
} …
```

5.3 Der Client

Es sind, wie bereits im Konzeptionskapitel erwähnt, besondere PHP-Skripte von Nöten, um eine aufrechte Verbindung zu gewährleisten und um weiterhin eine Socket-Verbindung mit dem Push-Server aufzubauen. Trotz der Tatsache, dass PHP serverseitig ausgeführt wird, wird die Funktionsweise des Skripts in diesem Unterkapitel erläutert, da die PHP-Skripte zusammen mit clientseitigen Skripten wie JavaScript und HTML in einer Datei gespeichert werden.

5.3.1 Das PHP-Skript

PHP liefert bereits die Möglichkeit, Socket-Verbindungen mit Servern aufzubauen. Die Funktionsweise wird im Folgenden unter der Zuhilfenahme des entsprechenden Quellcodes erläutert.

Zunächst müssen die nötigen Parameter zum Aufbau einer Socket-Verbindung festgelegt werden. Hierzu gehören die Host-Adresse und die Portnummer des Push-Servers.

```
<?php
$service_port = 2000;
$address = gethostbyname('127.0.0.1');
$socket = socket_create(AF_INET,
          SOCK_STREAM, SOL_TCP); …
```

Im Folgenden muss überprüft werden, ob ein TCP-Socket erzeugt werden kann. Falls dies scheitern sollte, soll eine Fehlermeldung ausgegeben werden.

```
if ($socket === false) {
 echo "socket_create() fehlgeschlagen: Grund: " .
 socket_strerror(socket_last_error()) . "\n";
} else {
 echo "OK.\n";
}
flush(); …
```

Im nächsten Schritt werden die Parameter überprüft und es wird versucht eine Verbindung aufzubauen.

```
$result = socket_connect($socket, $address, $ser-
vice_port);
if ($result === false) {
 echo "socket_connect() fehlgeschlagen.\nGrund:
 ($result) " . socket_strerror(socket_last_error(
              $socket)) . "\n";
 flush();
} else {
 echo "<b>Verbunden!</b><br/>";
 flush();
} …
```

Als nächstes wird die in Kapitel 4.1.1 erläuterte Socket-Schleife zur Realisierung der HTTP-Stream-Technik erzeugt, um eine permanente Verbindung aufzubauen. Hier hat man außerdem die Möglichkeit, die Ausgabe des Push-Servers mit Hilfe von regulären Ausdrücken zu filtern, um zu garantieren, dass nur die Daten aktualisiert werden, die auch tatsächlich vom Benutzer verlangt wurden. So ist es zum Beispiel möglich, ankommende Daten für den Chat des Konferenzsystems nach dem Raum des Chats zu sortieren oder durch Ajax an die richtige Stelle der Seite im Frontend zu platzieren. Aufgrund der besseren Übersichtlichkeit werden das Filtern der eingehenden Informationen mit Hilfe von regulären Ausdrücken und die Anwendung von Ajax nicht näher erläutert.

```
…
//Realisierung der HTTP-Stream-Technik
while ($out = socket_read($socket, 2048)){
 echo $out;
 //Ausgabe der vom Push-Server eingehenden
 //Informationen
 flush();
}

socket_close($socket);
//Schliesst Verbindung zum Server
flush();
?>
```

5.3.2 Realisierung sonstiger Skripte

Nach der Fertigstellung des Frameworks für die hier verwendete Push-Architektur wurde mit diesem das webbasierte Konferenzsystem entwickelt.

Da die Socket-Schleife, erwähnt in Kapitel 4.2 und realisiert in Kapitel 5.3.1, als auch das Abschicken der Nachrichten über Ajax im Hintergrund realisiert wurden, mussten zusätzlich zum entwickelten Framework auch die entsprechenden Skripte programmiert werden. Die

Applikation besteht aus fünf einzelnen DIVs[9] (vgl. Abbildung 10).

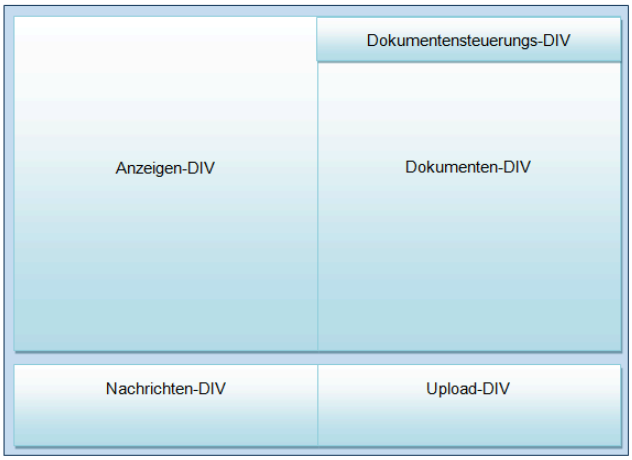

Abbildung 10: Grundaufbau der Webapplikation

Neben dem DIV für die Nachrichteneingaben und für den Upload der Dateien, gibt es noch zwei DIVs für das Anzeigen der Chatnachrichten und der hochgeladenen Dokumente. Ein weiteres DIV, eingebettet im Dokumenten-DIV, ermöglicht die Steuerung von hochgeladenen PDF-Dokumenten.

Der Nachrichten-DIV

Der Nachrichten-DIV befindet sich in der unteren linken Ecke der Seite und ist dafür verantwortlich, den Text, welcher im Formular eingegeben wurde durch Ajax im Hintergrund an den Push-Server über eine Socket-Verbindung zu melden.

Der eingegebene Text wird durch den „strip_tags()"-Befehl gefiltert, um nicht erlaubte bzw. gefährliche HTML- und PHP-Tags zu entfernen. Darüber hinaus wird der Text nach Smileys, die Gefühle, Stimmungen und Statements in Form von Tastatur-Kürzeln beschreiben, durchsucht und durch entsprechende graphische Symbole ersetzt. Weiterhin wird, bevor der Text an den Server geschickt wird, jedes Mal überprüft, ob die aktuell laufende Session gültig ist, damit garantiert werden kann, dass der abgeschickte Text auch tatsächlich vom eingeloggten Benutzer stammt. Abschließend wird der Text über eine aufgebaute Socket-Verbindung zusammen mit dem Usernamen und dem Raumnamen an den in Java geschriebenen Push-Server geschickt, damit dieser den Text an alle anderen verbundenen Clients (Browsern) senden kann.

[9] Ein Div-Tag definiert eine Einteilung oder einen Bereich in einem HTML-Dokument. (W3C, o.J.)

Der Upload-DIV

Der Upload-DIV ist dafür verantwortlich, Dateien von der Festplatte auf den Server hoch zu laden und den dazugehörigen Eintrag mit dem Pfad zur Datei in der Datenbank zu hinterlegen. Im Anschluss wird über die „Direktes Weiterleiten"-Methode ein spezifischer Befehl über den Push-Server an alle Clients geschickt. Dieser Befehl wird von den JavaScript-Engines der Clients erkannt und löst dort den Download des letzten in der Datenbank vermerkten Pfades aus. Nach dem Download werden die Dokumente im Dokumenten-DIV angezeigt.

Anzeige-DIVs für Nachrichten und Dokumente

Eingehende Nachrichten werden mit Hilfe von JavaScript gefiltert und je nachdem, ob es sich um eine Textnachricht oder um einen Pfad zu der anzuzeigenden Datei handelt, wird entweder die Textnachricht in den Nachrichten-DIV, oder das heruntergeladene Dokument in den Dokumenten-DIV geladen.

Dokumentensteuerungs-DIV

Im Dokumentensteuerungs-DIV befinden sich zwei Buttons mit der Bezeichnung „+" und „-", die für das Durchblättern der PDF-Dokumente zuständig sind. Dahinter verbirgt sich ein Skript, welches den Pfad der letzten hochgeladenen PDF-Datei in der Datenbank aufgreift und nach einer Reihe von Filterungen mit Hilfe regulärer Ausdrücke einen neuen Pfad-Eintrag in der Datenbank inklusive der neuen Seitenzahl hinterlegt.

6. Messergebnisse

Dieses Kapitel soll die Vorteile und die Wirtschaftlichkeit, die durch die Push-Architektur erzielt werden, belegen.

Als Testsystem wurde ein Server mit der Windows-Version 7 verwendet. Die Spezifikationen des Testsystems werden in Tabelle 1 vorgestellt.

Tabelle 1 Hardwarespezifikation des Testsystems

Hardware	Spezifikation
CPU	Intel CULV SU9400
RAM	4GB DDR3
HDD	320GB, 5400U/min
Netzwerk	1 GBit LAN
Internetanbindung	8 Mbit/s

Auf diesem Testsystem wurden unter verschiedenen Bedingungen sowohl die Client- als auch die Server-Auslastung unter Verwendung der Polling-Methode (Vgl.

Kapitel 3.2.1) bzw. der Push-Architektur (Vgl. Kapitel 4) getestet.

6.1 Client-Auslastung

Zur Messung der Client-Auslastung wurden zwei PHP-Seiten, die im Browser aufgerufen werden und ein Java-Programm, zur Generierung von Nachrichten, geschrieben.

6.1.1 Polling

Der Programmcode zur Messung der Client-Auslastung unter Verwendung der Polling-Methode sah wie folgt aus:

```
                    //CPoll.php

//1 Aktualisierung pro Sekunde
<META HTTP-EQUIV=Refresh CONTENT="1">

//Datenbankabfrage
<?php
mysql_connect("localhost", "root") or
die(mysql_error());

mysql_select_db("test") or die(mysql_error());

$result = mysql_query("SELECT * FROM tabelle")
or die(mysql_error());

$row = mysql_fetch_array( $result );
echo $row[0]."<br/>";
?>
```

Diese PHP-Seite wird im Browser aufgerufen und bewirkt, dass sich die Seite in festgelegten Zeitabschnitten selbst aktualisiert. Dadurch werden in regelmäßigen Abständen Anfragen an den Web-Server und an die Datenbank gesendet, um eventuell neu vorhandene Nachrichten abzufragen.

6.1.2 Push-Server

Zur Messung der Client-Auslastung unter Verwendung der Push-Architektur, wurde der folgende Programmcode verwendet:

```
                    //CPush.php

<?php
$service_port = 2000;
$address = gethostbyname('10.0.0.3');
$socket = socket_create(AF_INET, SOCK_STREAM,
SOL_TCP);
$result = socket_connect($socket, $address, $ser-
vice_port);
while ($out = socket_read($socket, 2048)) {
 echo str_pad($out."<br/>",8);
 flush();
}
?>
```

Dieses im Browser und innerhalb einer PHP-Seite aufgerufene PHP-Skript verbindet sich, analog zu dem in Kapitel 5.3.4 erläutertem Programmcode, mit dem in Kapitel 5.2 gezeigten Push-Server und verweilt nach einem erfolgreichen Verbindungsaufbau in der Socket-Schleife, um neue eingehende Nachrichten, analog zur Schleife aus Kapitel 4.1.1, sofort und ohne erneute Verbindungsaufbauten oder Datenbankabfragen anzuzeigen.

Zur Generierung neuer Nachrichten wurde ein Java-Programm geschrieben, welcher Nachrichten an den Push-Server aus Kapitel 5.2 sendet:

```
import java.io.*;
import java.net.*;

class BenchmarkPushServer
{

 public static void main(String argv[]) throws
  Exception{

  String nachricht = "Nachricht</br>";
  Socket clientSocket = new Socket("10.0.0.3",
                        2000);
  PrintWriter outToServer = new PrintWriter(
     clientSocket.getOutputStream(),true);

  //2000 Nachrichten an Push-Server senden.
  for(int i = 0; i<2000; i++){
         outToServer.println(nachricht);

   //1 Nachricht pro Sekunde
   Thread.currentThread().sleep(1000);

  }

    clientSocket.close();
  }
}
```

6.1.3 Ergebnis

Abbildung 11 zeigt, dass bedingt durch die neuen Verbindungsaufbauten die Client-Auslastung unter Verwendung der Polling-Methode rapide steigt und bereits bei etwas über 10 Nachrichten pro Sekunde die CPU voll ausgelastet wird.

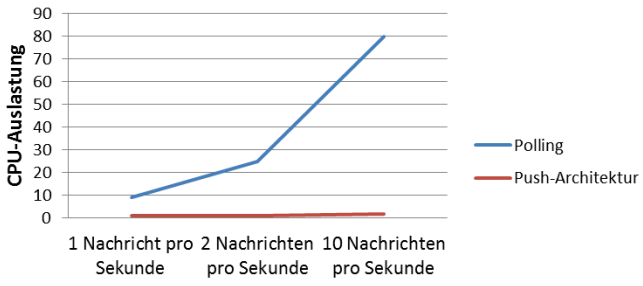

Abbildung 11: Client-Auslastung

Im Gegensatz dazu bleibt unter Verwendung der Push-Architektur die CPU-Auslastung des Clients nahezu konstant, da der Browser keine neuen Verbindungen aufbauen muss um neue Nachrichten zu erhalten.

6.2 Server-Auslastung

Die Server-Auslastung wurde gemessen, indem der maximal mögliche Durchsatz an Nachrichten ermittelt wurde.

Um den Server unter Verwendung der Polling-Methode auszulasten, wurde die Benchmark-Software „ApacheBench[10]“ verwendet, welche eine Vielzahl von HTTP-Anfragen simuliert.

Um den Durchsatz des Servers unter Verwendung der Push-Architektur zu messen, wurde das Java-Programm verwendet.

6.2.1 Polling

Die Leistungsmessung mit der kommandozeilenbasierten Software „ApacheBench“ (ab) ermöglicht es, das Verhalten von Web-Servern bzw. -Anwendungen unter Last zu überprüfen. Mit ApacheBench können eine Vielzahl von HTTP-Anfragen simuliert werden, um den Server auszulasten. Folgende PHP-Seite wurde durch ApacheBench aufgerufen:

```
                    //SPoll.php

<?php
mysql_connect("localhost", "root") or
die(mysql_error());

mysql_select_db("test") or die(mysql_error());

$result = mysql_query("SELECT * FROM tabelle")
or die(mysql_error());

$row = mysql_fetch_array( $result );
echo $row[0]."<br/>";

?>
```

Diese PHP-Seite verbindet sich mit der Datenbank und ruft die letzte, in der Tabelle vorhandene Nachricht ab. Es wurden zwischen 100 und 3000 Instanzen gestartet, die jeweils den Server nach neuen Nachrichten abfragen:

```
C:\> ab -n 100 SPoll.php

Time taken for tests:0.989 seconds
Total transferred:   1003100 bytes
HTML transferred:    949000 bytes
Requests per second: 52.94 [#/sec] (mean)
Time per request:    188.883 [ms] (mean)
Transfer rate:       518.62 [Kbytes/sec] received
```

6.2.2 Push-Server

Um auf dem Push-Server viele eingehende Nachrichten zu simulieren, wurde das aus Kapitel 6.1.2 bereits bekannte Java-Programm verwendet. Dieses Programm verbindet sich mit dem Push-Server und sendet in einer For-Schleife – je nach Testfall – zwischen 100 und 3000 Nachrichten.

```
import java.io.*;
import java.net.*;

class BenchmarkPushServer
{

 public static void main(String argv[]) throws
  Exception{

  String ToServer = "Nachricht</br>";
  Socket clientSocket = new Socket("10.0.0.3",
                        2000);
  PrintWriter outToServer = new PrintWriter(
     clientSocket.getOutputStream(),true);

  //100 Nachrichten senden
  for(int i = 0; i<100; i++){
         outToServer.println (ToServer) ;
  }

    clientSocket.close();
  }
}
```

Um nun den Server-Durchsatz zu messen, wurde der in Kapitel 6.1.2 verwendete PHP-Programmcode so erweitert, dass dieser die Zeit misst, bis alle vom Java-Programm abgesendeten Nachrichten eingegangen sind.

```
<?php
$service_port = 2000;
$address = gethostbyname('10.0.0.3');
$socket = socket_create(AF_INET, SOCK_STREAM,
SOL_TCP);
$result = socket_connect($socket, $address, $ser-
vice_port);

$datei = "data.txt";
$fp = fopen($datei,"a");
fputs($fp,"Zeit:\n");
fwrite($fp,"Mein Tutorial\n");
$start = microtime(true);
$nachrichten = 0;


while ($out = socket_read($socket, 2048)) {
 $nachrichten++;

//Nach 100 Nachrichten Socket-Schleife verlassen,
//vergangene Zeit messen und in data.txt schrei-
//ben.
 if($nachrichten == 100){
  break;
 }
$zeit = microtime(true)-$start.' Sekunden';
fputs($fp,$zeit);
}
?>
```

Sind alle Nachrichten angekommen, wird die benötigte Zeit in einer Datei gespeichert und ausgewertet.

[10] ab is a tool for benchmarking your [...] Hypertext Transfer Protocol (HTTP) server. (Apache, o.J.)

6.2.3 Ergebnis

Betrachtet man die Ergebnisse der Messung, so erkennt man deutlich, dass durch den Einsatz der Push-Technik vorhandene Ressourcen besser genutzt werden (vgl. Abbildung 12).

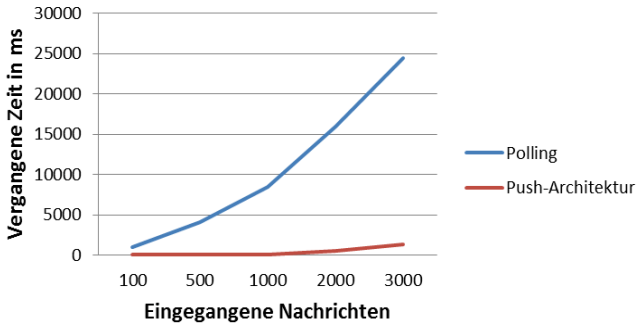

Abbildung 12: Server-Durchsatz

Analysiert man die Ergebnisse weiter, so erkennt man, dass durch die Anfragen, die für die Polling-Methode nötig sind, eine stark überproportionale Ressourcenauslastung verursacht wird und Nachrichten dadurch langsamer versendet werden. Der persistente Speicher, in diesem Fall die MySQL-Datenbank, der für die Polling-Methode nötig ist, verursacht eine zusätzliche Auslastung des Servers und verringert dadurch auch den Nachrichtendurchsatz.

Im Gegensatz zur Polling-Methode steht die Push-Architektur. Vor allem der Push-Server ist in der Lage, Ressourcen zu sparen. Seine Fähigkeit, eintreffende Ereignisse wie zum Beispiel Textnachrichten zu erkennen sowie diese über die persistente Verbindung ohne vorherige Anfragen und ohne ständige Verbindungsaufbauten zu senden, trägt maßgeblich zur Einsparung der Ressourcen bei.

6.2.4 Durchsatz in Abhängigkeit verbundener Clients

Eine wichtige Voraussetzung, um Echtzeitwebapplikationen zu entwickeln, ist eine Architektur, die möglichst viele Verbindungen verwalten und dennoch einen hohen Datendurchsatz erzielen kann. Im Folgenden wird gezeigt, wie hoch der Durchsatz in Abhängigkeit der verbundenen Clients mit der hier vorgestellten Push-Architektur ist.

Um viele Verbindungen zu simulieren, werden zwischen 10 und 10000 Instanzen des PHP-Skripts aus Kapitel 6.2.2 verwendet. Hierfür wird ApacheBench verwendet:

```
ab -n Anfragen -c Verbindungen server.php
```

Jede gestartete Instanz erhält nacheinander 1000 Nachrichten, die von dem in Kapitel 6.2.2 gezeigten Java-Programm an den Push-Server gesendet werden.

Sobald die jeweiligen Instanzen die Nachrichten erhalten haben, wird die Zeit die hierfür benötigt wurde in einer Datei gespeichert und über alle Instanzen aufsummiert. So ist es möglich, die Dauer, bis alle 1000 Nachrichten bei allen Instanzen eingegangen sind, zu messen. Die Auswertung ist in Abbildung 13 zu sehen.

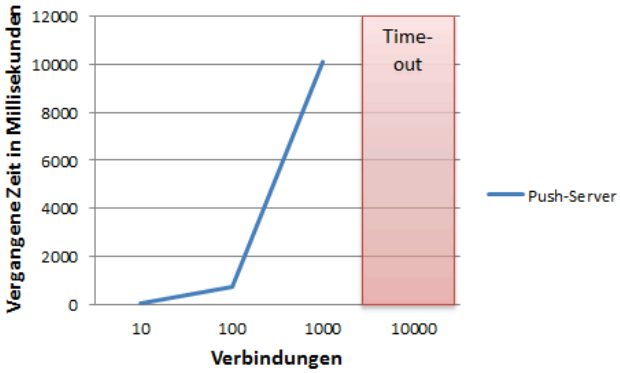

Abbildung 13: Durchsatz in Abhängigkeit verbundener Clients

7. Ergebnisse der Arbeit

Dieses Kapitel geht beispielhaft auf das Ergebnis der entwickelten Webapplikation ein. Die Applikation wird sowohl auf Client- als auch auf Serverseite durch Screenshots vorgestellt.

Wird die Webapplikation aufgerufen, so erscheint zunächst der Login-Bereich (vgl. Abbildung 14).

Abbildung 14: Login-Bereich

Hier können sich bereits vorhandene User einloggen oder neue User registrieren. Durch Betätigung des „Jetzt Registrieren"-Links wird die Seite für die Registrierung aufgerufen.

Für die Registrierung werden die entsprechenden Felder ausgefüllt. Durch die Betätigung des „Daten absenden"-Buttons, werden die eingetragenen Daten in einer Datenbank gespeichert und die Registrierung abgeschlossen (vgl. Abbildung 15).

Die eigentliche Webapplikation zeigt Abbildung 16. Hier finden sich auch die in Kapitel 5.3.2 erläuterten Bereiche wieder.

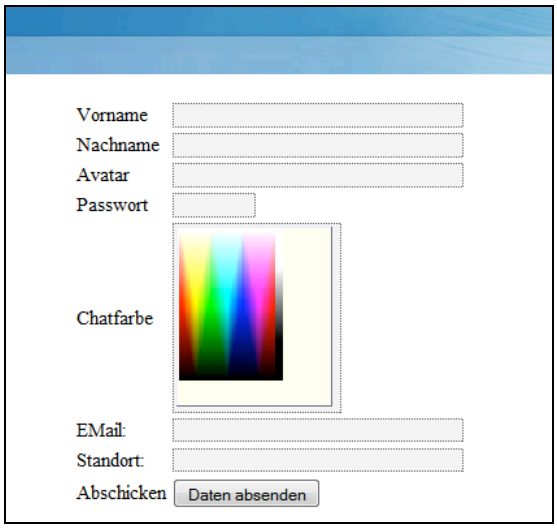

Abbildung 15: Registrierung

So können etwa erfolgreich eingeloggte User Nachrichten im Formular des Nachrichten-DIVs eingeben, die Unterhaltung im Anzeigen-DIV mitverfolgen, Dokumente mit Hilfe des Upload-Formulars im Upload-DIV von der lokalen Festplatte auswählen, anschließend für alle Konferenzteilnehmer hochladen und diese schließlich gemeinsam im Dokumenten-DIV in Echtzeit betrachten. Das Formular im Dokumentensteuerungs-DIV dient der Steuerung hochgeladener PDF-Dokumente.

8. Fazit und zukünftige Erweiterungsmöglichkeiten

Diese Arbeit veranschaulicht anhand des webbasierten Konferenzsystems, wie eine asynchrone Informationsverteilung im Internet trotz des grundsätzlich anfrageorientierten HTTP-Protokolls möglich ist. Mit der hier verwendeten Push-Technik können nicht nur Webapplikationen, die Echtzeitdaten benötigen, realisiert, sondern auch nachweislich Ressourcen gespart werden.

Auch wenn die Push-Technik eine Wendung im derzeitigen synchronen Paradigma verspricht, liefert diese Technik jedoch nicht die optimale Lösung.

So können mit der Push-Technik lediglich Simplex-Verbindungen, also solche, die Daten in jeweils nur eine Richtung gleichzeitig übertragen, aufgebaut werden. Eine einzige Verbindung aufzubauen, über die Daten gleichzeitig in beiden Richtungen gestreamt werden können, wäre die optimale Lösung. Umgesetzt werden

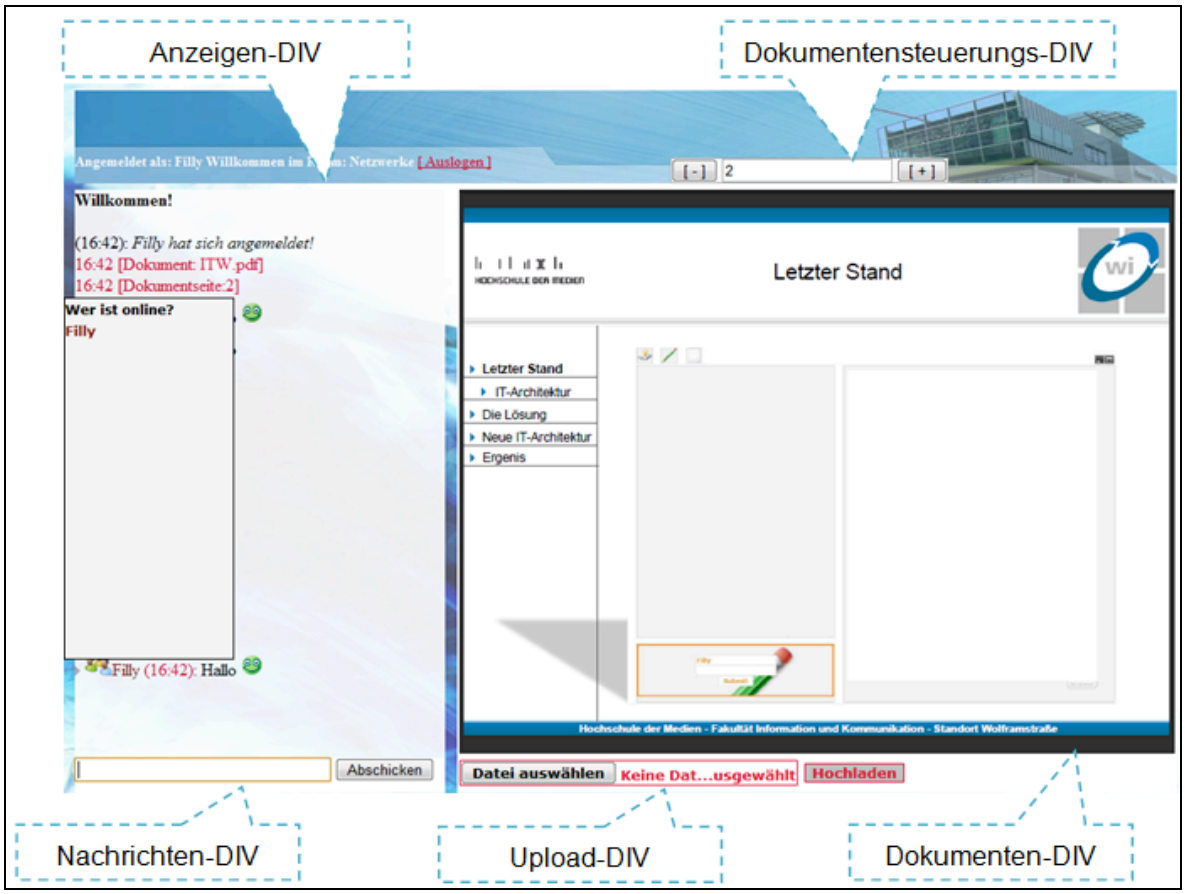

Abbildung 16: Screenshot Webapplikation

könnte dies mit so genannten Web-Sockets, die vom derzeit aufkommenden HTML5 unterstützt werden (Smith, 2008).

Um die Push-Technik außerdem produktiv einzusetzen und nicht nur wenige hunderte, sondern mehrere zehntausende persistente Verbindungen zu verwalten, sollten einige Änderungen auf der Server-Seite vorgenommen werden.

Zunächst sollten die Funktionalitäten von Apache-Server, PHP-Interpreter und Java-Push-Server zu einer Applikation verschmolzen werden, um die vorhandenen Systemressourcen nicht auf drei Prozesse aufzuteilen, sondern auf einen zu konzentrieren. Dabei muss man allerdings Einbußen in puncto Modularisierung und Wiederverwendbarkeit der Komponenten in Kauf nehmen.

Weiterhin sollten für die Verwaltung der Verbindungen nicht –wie in Kapitel 5.2 dargestellt- jeweils eigene Threads verwendet werden, sondern es sollte zum Beispiel mittels Thread-Multiplexing versucht werden, mehrere Verbindungen durch einen einzelnen Thread verwalten zu lassen. Auch die Verwendung von leichtgewichtigen Prozessen, wie sie aus der Programmiersprache Erlang bekannt sind, wäre denkbar (Erlang, o.J.).

Auf diese Weise könnten kostspielige Thread-Kontext-wechsel zwischen User- und Kernel-Space vermieden und sowohl der Durchsatz als auch der Speicherverbrauch enorm verbessert werden.

Um die Funktionsweise der Push-Architektur zu veranschaulichen, wurden diese komplexen Ansätze jedoch nicht verwendet. Stattdessen wurden die wichtigsten Aspekte am Beispiel einer möglichst einfachen Architektur erläutert.

Literaturverzeichnis

Alione, A. (2005). Changing the Web Paradigm. http://www.lightstreamer.com /Light streamer_Paradigm.pdf. Zugriff am 13.08.2009.

Apache (o.J.). ab - Apache HTTP server benchmarking tool. http://httpd.apache.org/docs/2.0/programs/ab.html. Zugriff am 18.08.09.

Apache Friends (o.J.).XAMPP. http://www.apachefriends.org/de/xampp-windows.html#628. Zugriff am 18.08.09.

Arcand, J. (2007). New Adventures in Comet: polling, long polling or Http streaming with AJAX. Which one to choose?. http://weblogs.java.net/blog/jfarcand /archive/2007/05/new_adventures.html. Zugriff am 13.08.2009.

Carl, D. (2006). Praxiswissen Ajax. 1. Auflage. Köln: O'Reilly Verlag.

Erlang (o.J.). Manual User's Guide. http://www.erlang.org/doc/reference_manual/processes.html. Zugriff am 18.08.09.

Faßhauer, C., Hielscher, M. & Wagenknecht, C. (2010). Signalserver. Informatik-Spektrum 06-2010 [33], S.612- S.620.

Garrett, J. (2005). Ajax: A New Approach to Web Applications. http://www.adaptivepath.com/publications/ essays/archives/000385.php. Zugriff am 13.08.2009.

Hunt, C. (2003). TCP/IP Netzwerk-Administration. Deutsche Übersetzung von K. Lichtenberg. 3. Auflage. Köln: O'Reilly Verlag.

Hammer, N. & Bensmann, K. (2006). Webdesign für Studium und Beruf: Webseiten planen, gestalten und umsetzen. 1. Auflage. Berlin: Springer Verlag.

Jäger, K. (2007). AJAX in der Praxis: Grundlagen, Konzepte, Lösungen. 1. Auflage. Berlin: Springer Verlag.

Kannengiesser, M. (2005). Webseiten mit PHP 5 und MySQL 4. 111 Lösungen für Ein- und Umsteiger. 1. Auflage. München: Markt & Technik Verlag.

Kollmann, T. & Häsel, M. (2007). Web 2.0 Trends und Technologien im Kontext der Net Economy. 1. Auflage. Wiesbaden: Gabler Verlag.

Levine, J. (2007). The Internet for Dummies. 1. Auflage. New Jersey: John Wiley & Sons Verlag.

Maurice, F. (2007). Web 2.0-Praxis. 1. Auflage. München: Markt & Technik Verlag.

Müller, G. (2002). Telematik- und Kommunikationssysteme in der vernetzten Wirtschaft. Berlin: Oldenbourg Verlag

Richter, A. & Koch, M. (2007). Social Software – Status quo und Zukunft. http://www.kooperationssyste me.de/wp-content/uploads/RichterKoch2007.pdf. Zugriff am 20.09.09.

Smith, R. (2008). The Future of the Web: HTML5 Web Sockets. http://ajax.sys-con.com/node/ 677813?page=0,1. Zugriff am 13.08.2009.

Sun (o.J.). Applets. http://java.sun.com/applets/. Zugriff am 13.08.2009

W3C (2005). Document Object Model (DOM). http://www.w3.org/DOM/. Zugriff am 18.08.09.

W3C (o.J.): HTML Tag. http://www.w3schools.com/tags/tag_DIV.asp. Zu- griff am 18.08.09.