Hausarbeit

Markus Dreyer

Architekturen für ein Collaborative Workspace

Fakultät Technik und Informatik
Department Informatik

Markus Dreyer

Thema der Hausarbeit

Architekturen für ein Collaborative Workspace

Stichworte

Kollaborativ, Architekturen, Technologien, AJAX

Kurzzusammenfassung

Für kollaboratives Zusammenarbeiten an einem Ort bedarf es einiger Voraussetzungen in der Systemarchitektur. Diese Arbeit stellt Konzepte und Architekture vor, die Teamarbeit unterstützen. Es werden einige allgemeine Probleme aufgezeigt und Technologien vorgestellt.

Markus Dreyer

Title of the paper

Architectures for Collaborativ Workspaces

Keywords

Collaborativ Workspace, Architectures, Technologies, AJAX

Abstract

Working in collaborative Workspaces has the need of well developed Applications. This article describes some architectures which support the interactive working in teams. It showes some common problems and technologies helping to setup a good infrastructure for workspace needs.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

In der heutigen digitalen Welt verschmelzen Computertechnologie und Arbeit immer mehr zu einer Einheit. Dies führt dazu, dass in allen Bereichen auf neue Technologien zurückgegriffen werden soll. Damit dieses Ziel erfolgreich erfüllt werden kann, ist es wichtig, dass der technologische Fortschritt sich an die Bedürfnisse der Benutzer anpasst, besser noch, die Benutzung der Technologien gar nicht auffällt. In diesem Zusammenhang spricht man von Ubiquitous Computing.

In dieser Arbeit soll ein Schritt in diese Richtung unternommen werden. Zu diesem Zweck werden die aktuellen Softwarearchitekturen vorgestellt und einige Technologien aufgezeigt, die in solchen Gebieten eingesetzt werden können. Sie stellen die Grundlage für weitere Entwicklungen und sollen, wenn möglich, im Masterlabor der Hochschule für Angewandte Wissenschaften Hamburg in reale Anwendungen einfließen. Das Labor dient unter anderem der prototypischen Entwicklung von Collaborative Workspaces, die in dieser Arbeit die Grundlage bieten sollen.

1.1 Ziele

Die erörterten Architekturen sollen die Grundlage schaffen, um eine Anwendung zu entwerfen, die sich ohne Installation und sofort einsetzbar in ein solches Umfeld eingliedern lässt. Diese Funktionalität bietet eine große Flexibilität für einen Workspace, da nicht auf die Geräte, die dort genutzt werden eingegangen werden braucht, sondern die Anwendungen systemunabhängig laufen sollen.

1.2 Gliederung

In Kapitel 2 wird der Begriff des Collaborative Workspace kurz erläutert. Weiterhin werden einige wünschenswerte Eigenschaften von Workspaces angeführt.

Kapitel 3 stellt die gängigsten Architekturen von Systemen vor. Dabei wird zwischen den Softwarearchitektur (3.1) und Systemarchitektur (3.2) unterschieden. Beide bieten unterschiedliche Sichtweisen auf eine Software.

Unter Anwendungsbeispielen (4) werden Konzepte für den Aufbau von Collaborative Workspaces vorgestellt. Diese stehen für verschiedene Ansätze bei der Kommunikation und ihrem Verhalten in dem System und werden im Abschnitt 4.5 mit Beispielen von existierenden Anwendungen belegt.

Das Kapitel 5 gibt einen kurzen Überblick der einsetzbaren Technologien.

Am Ende erfolgt eine Zusammenfassung und ein Ausblick auf den zu entwickelnden Prototyp.

2 Collaborative Workspaces

Ein Collaborative Workspace ist ein Raum oder ein Arbeitsplatz, an dem mehrere Personen gemeinsam an einem Projekt oder einer Idee arbeiten. Eine solche Arbeitsweise an einem Ort ermöglicht es den Teilnehmern, produktiver an der Lösung eines Probelms zu arbeiten. Sie können Hilfsmittel wie Mimik und Gestik für die Kommunikation benutzen und auf unterstützende Geräte, wie Blackboard oder einfach nur Papier zurückgreifen, um Ihre Ideen aussagekräftiger zu veranschaulichen. Ein solcher Raum (z.B. Johanson u. a. (2002) http://iroom.stanford.edu/) sollte flexibel einsetzbar sein und die Teilnehmer ohne weitere Installation oder Konfiguration ihrer mitgebrachten Geräte (Notebooks, PDA etc.) in den Workspace integrieren. Wie man aus dieser kurzen Einführung entnehmen kann, sind an einem solchen Workspace mehrere Computersysteme beteiligt.

2.1 Eigenschaften

Ein solcher Workspace setzt voraus, dass sich alle Teilnehmer auf ein Protokoll geeinigt haben. Dieses bezieht sich sowohl auf den Ablauf des Treffens als auch auf die Art und Weise, wie computerbasiert gearbeitet wird. Dazu gehört die Festlegung auf bestimmte Programme für die Bearbeitung von Daten, damit jedes Teammitglied sie später lokal weiterverarbeiten kann. Wünschenswert wäre daher ein Konverter, der zwischen unterschiedlichen Formaten transformieren kann. Weiterhin benötigt man eine geeignete Methode, den Verlauf des Meetings zu dokumentieren, um später die Entscheidungen nachvollziehen zu können. Diese Dokumentation, sowie alle anderen erzeugten Dokumente sollten sowohl den Teilnehmern lokal, als auch global zur Verfügung stehen, um sie, zum Beispiel, Mitarbeitern oder entfernt arbeitenden Personen automatisch bereit zu stellen.

3 Architekturen

Wie man aus der kurzen Einführung entnehmen kann sind an einem Collaborative Workspace mehrere Computersysteme beteiligt. In diesem Kapitel werden die gängigsten Architekturen für verteilte Systeme vorgestellt. Es ist aufgeteilt in die Bereiche Software- und Systemarchitekturen. Die Softwarearchitekturen stellen dabei die Sicht auf ein Programm und ihre Interaktion dar, während die Systemarchitektur die Implementierung einer Software auf Computern widerspiegelt.

3.1 Software Architekturen

Die Architektur einer Software beschreibt den logischen Aufbau und die Kommunikation innerhalb des Programms. Sie legt Regeln fest mit der Komponenten verknüpft werden können

und wie stark die Bindungen untereinander sind. Bei allen vorgestellten Architekturen handelt es sich um Konzepte, die ihre jeweiligen Vorteile haben. Genaueres wird in den folgenden Abschnitten thematisiert.

3.1.1 Schichten

Wie der Name schon sagt, ist in diesem Fall eine Einteilung der Software in unterschiedliche Bereiche vorzunehmen. Die Architektur gibt vor, dass Komponenten in ihren eigenen Schichten wirken. Hier haben sie lediglich die Möglichkeit mit Schichten, die an ihre eigene angrenzen, über Schnittstellen zusammen zu arbeiten. Dieses Verhalten ist aus dem Netzwerkbereich bekannt, dort ist das ISO OSI Schichtenmodel quasi zum Standard geworden. Die Kopplung der einzelnen Schichten ist relativ lose, sodass eine Schicht ausgetauscht werden kann, ohne dass eine andere davon betroffen wäre. Damit ist jede Schicht für sich frei definierbar, solange die Schnittstellen beachtet werden.

3.1.2 Objektbasiert

Dieser Ansatz benutzt für jede Softwarekomponente ein eigenes Objekt. Alle Objekte können miteinander interagieren. Das Modell besitzt eine losere Kopplung als die Schichten Architektur, dafür ist die Interaktion der Komponenten komplexer, weil es keine Kategorisierung gibt und somit jedes Objekt alle Objekte ansprechen können muss, dessen Dienste es benötigt.

3.1.3 Datenzentriert

In dieser Architektur findet die Kommunikation zwischen Komponenten über einen gemeinsamen Datenspeicher statt. Dies kann eine Datenbank oder auch das Dateisystem sein. Für diese Kommunikation müssen die einzelnen Komponenten lediglich das Datenformat lesen können. Dieser Ansatz ist zu vergleichen mit einer shared Memory Anwendung auf einem einzelnen Rechner.

3.1.4 Eventbasiert

Wird von einer Komponente ein Ereignis ausgelöst, werden alle an dieses Ereignis angemeldeten Komponenten darüber informiert. Dies ist ähnlich dem Observer Pattern aus Gamma u. a. (2005), mit dem Unterschied, dass eine Middleware die Verwaltung und Vermittlung der Ereignisse übernimmt. Dadurch entsteht eine lose Kopplung der einzelnen Softwareteile. Diese Architektur ermöglicht es, wie die datenzentrierte auch, dass Komponenten keine Kenntnis über andere - von ihnen abhängende - Software benötigen.

3.2 System Architekturen

Die hier vorgestellten Architekturen bieten einen Überblick über die möglichen Verteilungen von Software auf Systeme. Hierbei geht es um Interaktion der einzelnen Komponenten in verteilten Systemen. Vorgestellt werden zentrale, dezentrale sowie hybride Ansätze.

3.2.1 Zentrale Architektur

Diese oft als request-response Verhalten bezeichnete Architektur ist die verständlichste der hier vorgestellten. Die einzelnen Komponenten werden in Server und Klienten aufgeteilt, wobei jeder Klient auch ein Server sein kann. Ein Klient ist dabei eine Komponente, die einem Server einen Auftrag schickt und auf das Ergebnis wartet. Ein Server stellt seine Dienste über Schnittstellen bereit, die von außen angesprochen werden. Die Kommunikation kann dabei lose gekoppelt ablaufen. Wichtig hierbei ist, dass auf einen Verlust der Nachrichten korrekt reagiert wird. So sollte ein nochmaliges Senden der Anfrage nicht geschehen, wenn der Auftrag schon bearbeitet wurde. Der Server könnte sonst zum Beispiel eine Flugbuchung doppelt durchführen, die nicht gewollt ist.

Für die Kategorisierung der einzelnen Komponenten hat sich eine 3-Schichten Architektur als funktional erwiesen. Diese unterteilt die Software, wie in Kapitel 3.1.1 beschrieben. Die Unterteilung findet in die folgenden Bereiche statt.

1. User-Interface Ebene: Beinhaltet die Darstellung für den Benutzer.
2. Verarbeitungsebene: Stellt die Businesslogik und bereitet Datenein- und ausgabe auf.
3. Datenebene: Stellt den Zugriff auf die persistenten Daten sicher.

Da diese Bereiche in der Regel den Softwarekomponenten entsprechen spricht man von logischer oder auch vertikaler Verteilung der Anwendung. Welche dieser Schichten beim Klienten und beim Server implementiert sind ist unterschiedlich. Daraus ergeben sich die in Abbildung 1 dargestellten Aufteilungsmöglichkeiten. Bei (a) spricht man vom Thin-Client und

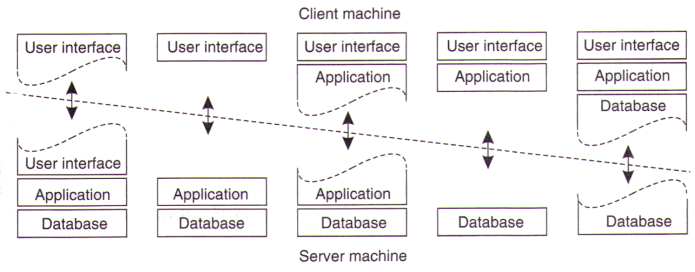

Abbildung 1: Klient und Server [Quelle:Tanenbau und van Steen (2007)]

bei (e) vom Fat-Client.

3.2.2 Dezentrale Architektur

Eine Dezentrale Architektur wird in peer-to-peer Systemen eingesetzt. Ein offensichtliches Ziel ist die Verteilung der Last auf viele Teilnehmer. Diese horizontale Verteilung bedingt, dass sich die Teilnehmer alle gleich verhalten - die gleiche Anfrage erzeugt bei unterschiedlichen Servern die identische Antwort. Sie implementieren sowohl den Klient als auch den Serverteil einer Anwendung und werden deswegen als Servent bezeichnet. Bei diesem Konzept kommt zusätzlich ein Verwaltungsaufwand für die logische Anordnung der Teilnehmer hinzu. Es muss darauf geachtet werden, welcher Knoten für welche Anfrage verantwortlich ist. Dies geschieht durch Überlagerung des physikalischen Netzwerkes mit einem logischen. Damit können alle Teilnehmer in einem Ring, oder anhand ihrer Position, in Gruppen geordnet werden. Dabei kennt jeder Teilnehmer die anderen, oder weiß einen Teilnehmer, der den zu erreichenden kennt. Es ist also möglich, einen Weg durch das Netzwerk zu finden, indem man die verantwortlichen Knoten anspricht. Bei dieser Variante spricht man von einer strukturierten peer-to-peer Architektur, während die unstrukturierte versucht, einen zufälligen Graphen als logische Schicht über das Netzwerk zu legen. Diese sorgt dafür, dass jeder zumindest seine Nachbarn kennt, also einen Ausschnitt des gesamten Netzwerkes. Das Routing von einem Teilnehmer zum anderen findet also zufällig statt (flooding), da es keine Hierarchie gibt. Um diesen Missstand zu beheben, werden einige Knoten mit besonderen Rechten und Pflichten ausgestattet. Diese ermöglichen es, gezielter nach Verbindungen zu fragen, da diese „Superknoten“ zum Beispiel eine Liste ihnen zugeordneter Unterknoten führen kann. Dies führt unweigerlich zu einer hybriden Architektur, wie sie im folgenden beschrieben wird.

3.2.3 Hybride Architektur

Bisher haben wir reine Klient Server und peer-to-peer Architekturen kennen gelernt. Eine Zusammenführung beider ermöglicht es, die Vorteile zu vereinen. So ist es möglich, ein einfaches Auffinden der Funktionen zu erreichen ebenso wie eine Lastverteilung. Um dies zu erreichen werden in einem verteilten System bestimmte Knoten mit besonderen Privilegien ausgestattet. Diese verwalten den Zugriff auf die eigentlichen Ressourcen, indem sie die Anfragen selbst beantworten oder an weitere Knoten delegieren. Für den Klienten sieht dies wie eine reine Klient Server Architektur aus, intern findet allerdings eine Lastverteilung statt. Ein Beispiel im peer-to-peer Bereich ist Bittorrent. Bei diesem Filesharingsystem werden .torrent Dateien aus dem Internet herunter geladen. Diese enthalten Ansprechpartner in dem Netzwerk, die Kenntnisse über die gewünschte Datei haben. Das Programm zum Herunterladen dieser Datei spricht nun den so genannten Tracker an. Dieser verfügt über eine Liste von anderen Klienten, die die angeforderte Datei, oder Teile von ihr, zur Verfügung stellen. Dieses Vorgehen ist vergleichbar mit einem Schneeballsystem. Der Ausgangspunkt (Tracker) kennt einige wenige Klienten, diese wiederum kennen andere Klienten, die die glei-

che Datei besitzen oder haben wollen usw. Die Stärke des Systems liegt also in der Masse der teilnehmenden Klienten, während die Server lediglich Listen von Referenzen verwalten müssen und nicht aktiv am Austausch der Daten beteiligt sind.

4 Szenarien für kollaboratives Arbeiten

An dieser Stelle werden einige Anwendungsfälle vorgestellt, die im Bereich des verteilten kollaborativen Arbeitens benötigt werden. Diese Szenarien werden folgend in Bezug auf die vorgestellten Architekturen bewertet. Abschließend werden einige existierende Lösungen vorgestellt.

4.1 Blackboard

Für kollaboratives Arbeiten ist es wünschenswert, dass sich Personen schnell und unkompliziert treffen können. Eine Variante, die Meetings zu unterstützen, geschieht durch das Bereitstellen von für jeden zugängliche Ressourcen. In diesem Beispiel soll diese Ressource eine elektronische Tafel sein. Diese Tafel kann durch einen Beamer oder einen großen Monitor realisiert werden. Der grundlegende Gedanke ist nun, dass alle Teilnehmer direkt Zugriff auf diese „Tafel“ haben. Über ihre Arbeitsplätze können sie neue Elemente erstellen oder vorhandene verändern. Das Blackboard dient demnach der Darstellung gemeinsam erarbeiteter Inhalte, wobei die räumliche Nähe der Teilnehmer dafür sorgt, dass wenig Konflikte bei der Bearbeitung entstehen.

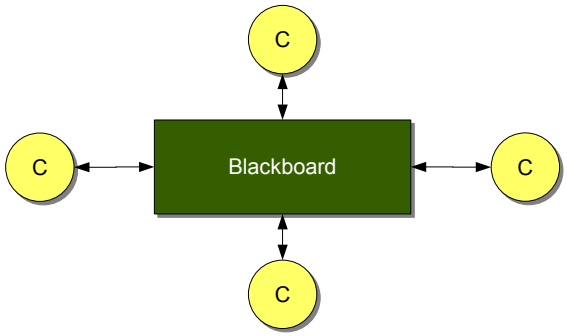

Abbildung 2: Schematisch Darstellung: Blackboard

4.2 Brainstorming

Konflikte können gänzlich vermieden werden, wenn ein Moderator die Diskussion führt oder es zu einem Zeitpunkt nur eine Person gibt die das Blackboard bearbeiten darf. Dies wird

in dem Szenario „Brainstorming“ verdeutlicht. Wie aus Abbildung 3 ersichtlich, gibt es einen ausgezeichneten Teilnehmer, der direkten Zugriff auf die globale Ressource hat. Die anderen können an ihrem Arbeitsplatz Daten vorbereiten und sie zur Verfügung stellen. Das Zusammenführen der Daten der einzelnen Teilnehmer wird dann von dem Moderator vorgenommen. Ein Brainstorming läuft ohne Unterstützung durch Computer sehr ähnlich ab. Zumeist wird die Diskussion geführt und Ideen aus der Gruppe der Teilnehmer an einem Punkt gesammelt und dargestellt.

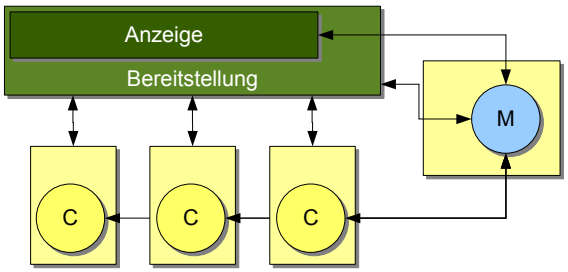

Abbildung 3: Schematische Darstellung: Brainstorming

4.3 Ad-Hoc

Dieses Szenario benutzt keinerlei globale Ressourcen. Vielmehr geht es hierbei um den spontanen Zusammenschluss diverser Klienten zu einer Gruppe. Jeder stellt den anderen seine Ressourcen zur Verfügung und macht sie somit global nutzbar. Dies sorgt dafür, dass die Meetings an beliebigen Orten stattfinden können und jedem Teilnehmer die erarbeiteten Daten zur Verfügung stehen. Der Vorteil dieses Verfahrens ist, dass jeder seine eigene Arbeitsumgebung benutzen kann und selbst entscheidet, welche Dienste angeboten werden. Jeder Teilnehmer kann dabei alle anderen Dienste nutzen.

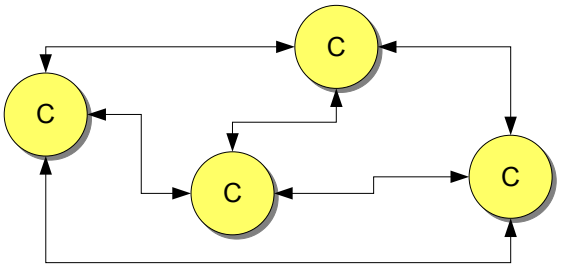

Abbildung 4: Schematische Darstellung: Ad-Hoc

4.4 Allgemeine Probleme

In allen Anwendungsfällen des gemeinsamen Arbeitens treten ähnliche Probleme auf. Zum einen gilt es, gleichzeitig benutzte Ressourcen zu schützen und zum anderen die Synchronität zwischen lokalen und globalen Inhalten zu gewährleisten.

Für den gemeinsamen Zugriff auf globale Inhalte haben sich Sperren als praktikable Lösung etabliert. Bei diesen muss allerdings bei unterschiedlichen Anwendungsfällen überlegt werden, welche Bereiche gesperrt werden. Wenn ein Benutzer zugreift, sperrt man in einem Dokument nur das Zeichen, welches der Nutzer gerade ändert, oder den ganzen Absatz exklusiv. Der Verwaltungsaufwand wächst, je kleiner die zu sperrenden Bereiche werden. Dafür kann das Dokument flexibler bearbeitet werden, da nur kleinste Teile nicht von anderen bearbeitet werden können. Eine weitere Möglichkeit ist die Arbeit mit Transaktionen. Ein Klient führt die Änderungen lokal durch und übergibt diese zurück an das globale Dokument. Hierbei ist die Strategie entscheidend, die eine Lösung von Konflikten bei einer Transaktion ermöglicht. Es kann zum Datenverlust kommen, wenn diese Strategie zum Beispiel immer die letzte Änderung als korrekt interpretiert, da so von anderen zuvor getätigte Anpassungen überschrieben werden.

Eine weitere Problematik ist die Synchronisation zwischen den globalen und den lokalen Datenbeständen. Sollte zum Beispiel ein Teilnehmer das Meeting frühzeitig verlassen speichert er den zu diesem Zeitpunkt aktuellen Stand lokal. Diesen kann er dort beliebig ändern, während die anderen Teilnehmer noch gemeinsam an dem Dokument arbeiten. Alle Teilnehmer sollen am Ende dennoch die selbe Version der Dokumente besitzen. Hier bedarf es ebenfalls sinnvoller Strategien für die Zusammenführung.

4.5 Existierende Lösungen

Die hier kurz vorgestellten Lösungen stellen einen Ausschnitt der vorhandenen Anwendungen dar. Die Auswahl ist nicht gewertet und greift lediglich unterschiedliche Architekturen auf.

Gobby ist eine Klient Server Anwendung, die es erlaubt, mit mehreren Benutzern simultan an Textdokumenten zu arbeiten. Der Server kann dabei an zentraler Stelle instanziiert werden, oder durch einen beliebigen Teilnehmer. Dieser so gestarteten Session können die anderen beitreten. Jeder bekommt eine Farbe zugewiesen, die die Änderungen am Dokument hervorheben. Verlässt ein Teilnehmer die Sitzung wird er weiterhin in der Benutzerliste aufgeführt und seine Änderungen auch mit seiner Farbe markiert. Screenshot siehe 5

GoogleDocs ist ein Online Dienst, der es erlaubt, Dokumente mit eingeladenen Personen zu bearbeiten. Der Vorteil hierbei ist, das keine Installation benötigt wird, sondern die im Browser integrierten Funktionen genutzt werden.

Carfagno (2007) hat im Rahmen seiner Diplomarbeit eine Browser Anwendung zum Erstellen von Mindmaps programmiert. Die Browserfähigkeit sorgt für eine große Kompatibilität

bei unterschiedlichsten Systemen, da AJAX heutzutage durch alle gängigen Browser unterstützt wird. Diese Anwendung kann von jedem Teilnehmer bereitgestellt werden, wodurch eine Arbeitsweise wie in Abschnitt 4.3 beschrieben, möglich ist.

Einen anderen Ansatz verfolgt das Johanson u. a. (2004) Projekt der Stanford Universität. Hier werden unterschiedlichste Devices mit Hilfe von großen Displays zur Zusammenarbeit verwendet. Diese Vorgehensweise entspricht dem Blackboard Szenario aus Abschnitt 4.1.

5 Technologien

An dieser Stelle werden einige Technologien vorgestellt, die beim Einsatz in kollaborativen Umgebungen sinnvoll erscheinen. Zu diesen gehören direkte Anwendungen als auch Software, die es ermöglicht eigene Anwendungen global nutzbar zu machen.

5.1 Remote Procedure Call

Das Verfahren, Aufgaben von entfernten Rechnern erledigen zu lassen, ist in fast jeder Klient Server Architektur anzutreffen. Aktuelle Vertreter dieses Vorgehens ist die Service Oriented Architecure. Die Daten werden dabei meist an einer globalen Stelle vorgehalten und über Services zur Verfügung gestellt. Damit kann eine globale Serverinstanz den Zugriff regulieren und die benötigten Sperrungen auf Datenebene vornehmen. Weiterhin können erarbeitete Dokumente dort abgerufen und bei Bedarf in unterschiedliche Ausgabeformate konvertiert werden. Der Verwaltungsaufwand liegt in der Wartung der globalen Instanzen über die ein Zugriff stattfindet. Werden hier Änderungen an der Software durchgeführt, benutzen ab diesem Zeitpunkt alle Klienten diesen Softwarestand. Der Zugriff auf die Schnittstellen kann aus beliebigen Anwendungen stattfinden, wodurch eine Plattformunabhängigkeit erreicht werden kann.

5.2 Code on Demand

Um eine Anwendung auf einem Klienten auszuführen, ist es sinnvoll, diese flexibel bereitzustellen. Der Klient lädt sich den aktuellen Stand und bringt diesen lokal zur Ausführung. Auch in diesem Fall findet die Softwarewartung zentral statt und muss nicht für jeden Klienten durchgeführt werden. Eine Erweiterung der angebotenen Anwendungen ist ebenfalls möglich ohne an den Klienten herantreten zu müssen, sie kann einfach global hinterlegt werden. Die bereitgestellten Anwendungen sollten dabei systemunabhängig funktionieren, damit die Ausführung in heterogenen Umgebungen gelingt.

5.3 Lokale Applikationen

Um eigene Anwendungen global zugänglich zu machen muss eine Infrastruktur bereitstehen, die die Anwendungen für unterschiedliche Ausgabemedien aufbereiten kann. Ebenso muss die Eingabeschnittstelle die Befehle an die lokal laufende Anwendung weiterreichen. Die Infrastruktur fungiert als Zwischenschicht und sorgt dafür, dass jeder Klient die Daten auf seine Bedürfnisse zugeschnitten, aufbereitet bekommt.

6 Zusammenfassung und Ausblick

Alle in dieser Arbeit vorgestellten Konzepte haben ihren Vorteil beim Einsatz in kollaborativen Arbeitsumgebungen. Am sinnvollsten erscheint dennoch der Einsatz einer hybriden Architektur, da diese die Vorteile der zentralen und dezentralen Architektur nutzen kann. Die zentrale Komponente kann dabei die Verwaltung der Software und Benutzer bewältigen, während die dezentralen Systeme, zum Beispiel die Notebooks der Teammitglieder, gemeinsam an Projekten arbeiten können, deren Ressourcen auf einem Monitor zusammengefasst werden.

Die Datenaufbewahrung und Verfügbarkeit der Softwarekomponenten sollte auf einem global zugänglichen System abgelegt werden. Dies ermöglicht den Zugriff auf die Daten von entfernten Rechnern aus und somit die Nachbearbeitung am eigenen Arbeitsplatz. Ideal wäre es zudem, wenn neben den Daten auch die Software zur Bearbeitung dort angesiedelt wäre. Die Systemverwalter müssten die Software so nur an einem Ort ändern und es gäbe bei den Mitarbeitern keine Versionskonflikte.

Für dieses Vorgehen ist AJAX einem mögliche Technologie. Sie bietet die Möglichkeiten lokal und gleichzeitig online zu arbeiten. So erweitert sie normale Webanwendungen die, dank standardkonformer Browser, auf fast jedem Gerät wiedergegeben werden können, um die Flexibilität des Offline arbeitens. Weiterhin kann die AJAX Software, sowie der Prototyp von Carfagno (2007) als reines Offline Programm zur Verfügung gestellt werden. Diese Funktion, erweitert um die Möglichkeit der lokalen Speicherung von Änderungen, zum Beispiel in einer Datenbank im Browser (siehe Derby) und einer nachfolgenden Synchronisation zwischen lokalen und globalen Daten, würde ein sinnvolles Arbeiten im Team unterstützen.

Als nächsten Schritt würde die Entwicklung eines solchen Prototyps anstehen. Als Beispiel könnte ich mir ein Spiel wie Jawbreak (siehe http://www.bigfrog.net/jawbreaker/) oder Sudoku als Multiplayer Webanwendung vorstellen. Besonderes Augenmerk ist hierbei sicherlich auf die Kleinstgeräte wie Smartphones oder PDAs zu richten, die ressourcenschonend arbeiten sollten.

A Anhang

Literatur

[Derby] URL http://www.ibm.com/developerworks/db2/library/techarticle/dm-0607cline/. – Stand 200-07-29

[Bittorrent] *Bittorrent*. online. – URL http://www.bittorrent.org/protocol.html. – Stand 2007-07-28

[Globole] *Globule: an Open-Source Content Distribution Network*. online. – URL http://www.globule.org/. – Stand 2007-07-27

[Gobby] *Gobby: a collaborative text editor*. – URL http://gobby.0x539.de. – Stand 2007-07-29

[GoogleDocs] *Google Docs*. – URL http://docs.google.com. – Stand 2007-07-29

[Carfagno 2007] CARFAGNO, Virginio: *Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/diplom/carfagno.zip. – Stand 2007-07-29

[Gamma u. a. 2005] GAMMA, E. ; HELM, R. ; JOHNSON, R. ; VLISSIDES, J.: *Design Patterns*. Addison-Wesley, 2005. – ISBN 0-201-63361-2

[Hollatz 2007] HOLLATZ, Dennis: *Einsetzbarkeit von AJAX-basierten Applikationen für kooperatives Arbeiten*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2007. – URL http://users.informatik.haw-hamburg.de/~ubicomp/arbeiten/bachelor/hollatz.pdf. – Stand 2007-07-29

[Johanson u. a. 2002] JOHANSON, Brad ; FOX, Armando ; WINOGRAD, Terry: The Interactive Workspaces project: experiences with ubiquitous computing rooms. In: *Pervasive Computing, IEEE Volume 1, Issue 2* (2002), S. 67–74. – URL http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/7756/21806/01012339.pdf

[Johanson u. a. 2004] JOHANSON, Brad ; FOX, Armando ; WINOGRAD, Terry: The Stanford Interactive Workspaces Project. (2004). – URL http://hci.stanford.edu/cstr/reports/2004-05.pdf. – stand 2007-07-29

[Tanenbau und van Steen 2007] TANENBAU, Andrew S. ; STEEN, Maarten van: *Distributed Systems - Principles and Paradigms*. Pearson, 2007. – ISBN 0-13-239227-5

A.1 Abbildungen

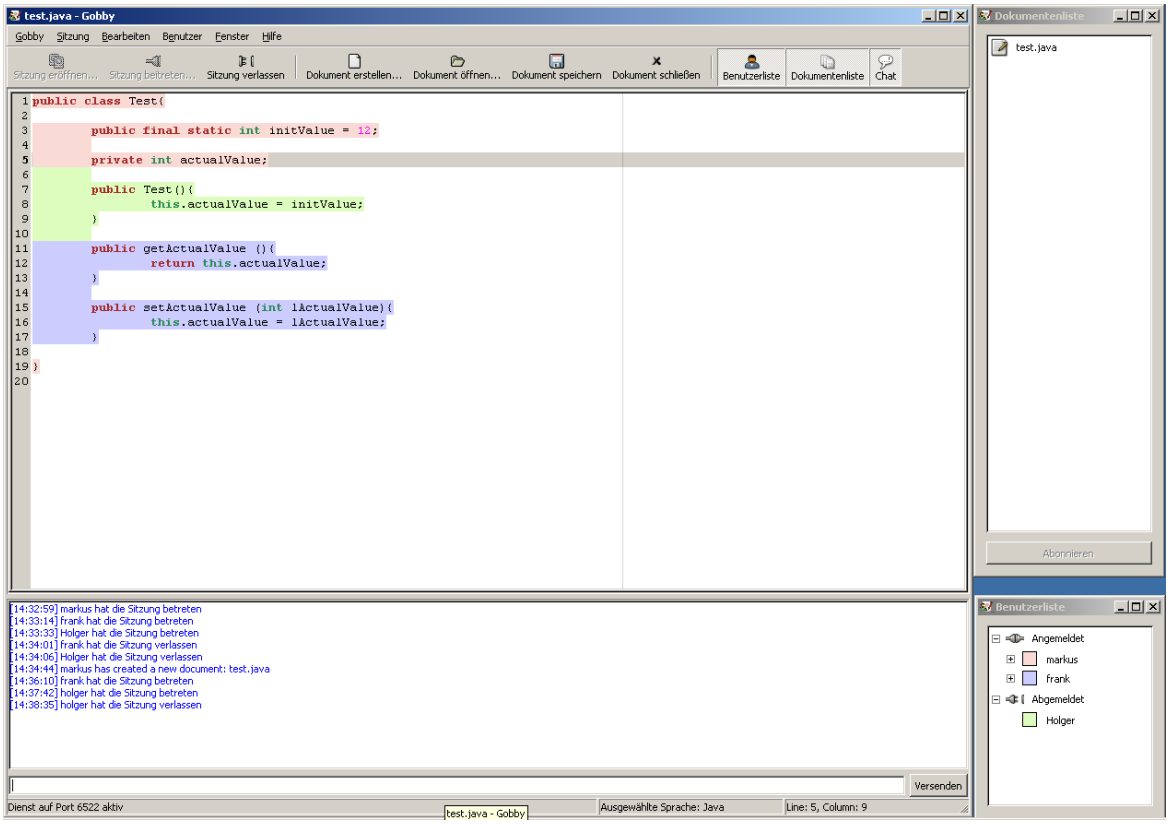

Abbildung 5: Screenshot von Gobby