Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

André Goldflam

Evaluation von Flex 2.0 für die Entwicklung kollaborativer Anwendungen

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

André Goldflam

Evaluation von Flex 2.0 für die Entwicklung kollaborativer Anwendungen

Bachelorarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Angewandte Informatik
Studienrichtung Softwaretechnik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Jörg Raasch

Abgegeben am 20. Juni 2008

André Goldflam

Thema der Bachelorarbeit

Evaluation von Flex 2.0 für die Entwicklung kollaborativer Anwendungen

Stichworte

Adobe Flex, Rich Internet Application, Collaborative Computer Supported Cooperative Work, Entwurfsprinzipien, synchrones Schreiben

Kurzzusammenfassung

Im Rahmen dieser Arbeit wurde die Evaluation von Adobe Flex 2.0 anhand des Systementwurfs einer kollaborativen Desktopanwendung durchgeführt. Die Evaluation zeigte, dass Flex nicht die Anforderungen des Systementwurfs erfüllen kann. Dies bedeutet nicht, dass Flex für die Entwicklung von kollaborativen Anwendung grundsätzlich untauglich ist, sondern dass bei dem Entwicklungsprozess nicht alle aus dem Bereich der Desktopanwendungen üblichen Entwurfsprinzipien auch in Flex angewendet werden können.

André Goldflam

Title of the paper

Evaluation of Flex 2.0 considering the development of collaborative applications

Keywords

Adobe Flex, Rich Internet Application, Collaborative Computer Supported Cooperative Work, design principles, synchronous editing

Abstract

In the context of this thesis, Adobe Flex 2.0 was evaluated by a system design for a possible collaborative desktop application. The evaluation showed, that flex does not meet the requirements posed by the system design. This does not mean though, that Flex is not suited for the development of collaborative applications in general, but that one cannot expect, that all the design principles known from development of desktop applications can be applied to the development with Flex as well.

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Kai von Luck und Herrn Prof. Dr. Jörg Raasch für ihre Betreuung und Unterstützung.

Zusätzlich möchte ich mich bei meiner Familie, meiner Freundin und meinen Studienkollegen für ihre Unterstützung während dieser Arbeit bedanken.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

1.1. Motivation

Obwohl die Vision der computergestützten gemeinsamen Arbeit zwar schon in den 80er Jahren beschrieben wurde, kann bisher noch nicht davon gesprochen werden, dass die mit dieser Vision verbundenen Aufgaben wirklich gelöst werden konnten. Gerade die orts- oder auch zeitunabhängige Gruppenarbeit ist immer noch ein sehr wichtiges Forschungsgebiet, da mit jedem technologischen Fortschritt neue Möglichkeiten entstehen, um Gruppenarbeit auf die verschiedensten Arten besser zu unterstützen. Im Laufe der Globalisierung wird diesem Thema eine zunehmende Bedeutung beigemessen. Der rapiden Verbreitung des Internets und der Entwicklung von immer schnelleren Breitbandverbindungen ist es zu verdanken, dass mittlerweile fast jede Art der Kommunikation durch den Computer unterstützt werden kann. Dabei reicht die Palette von Internetkonferenzen über Webcam oder Instant Messenger, Chats, Wikis, Blogs bis hin zu Foren, in denen fachliche aber auch allgemeine Themen diskutiert werden können. Im geschäftlichen Bereich wird dabei immer mehr Mobilität von den einzelnen Beteiligten gefordert. Oft besteht ein Team aus Spezialisten, welche gemeinsam eine bestimmte Aufgabe lösen, wobei sich manche Beteiligte im Laufe der Zusammenarbeit nie persönlich treffen.

Eine andere parallele Entwicklung zeichnet sich im Bereich der Webanwendungen ab. Sogenannte Rich Internet Applications sollen die Lücke zwischen den traditionellen Web- und Desktopanwendungen schließen und die Vorteile von beiden Anwendungstypen, Ortsunabhängig- und Benutzerfreundlichkeit, miteinander vereinen. Für den Endbenutzer äußert sich die Entwicklung von Rich Internet Applications darin, dass er über das Internet Dienste, wie z.B. Email- oder Terminmanagement mit einem ähnlichen Komfort wie auf einem Desktop nutzen kann. Im geschäftlichen Bereich wird den Rich Internet Applications noch aus einem weiteren Grund viel Beachtung geschenkt, dieser wird in der Theorie als „Change- and Configuration Management“ genannt. Dabei handelt es sich um das Problem, Änderungen an der Software durch Updates oder Konfiguration effizient zu gestalten. Dieses Thema ist mit enormen Zeit- und Kostenaufwendungen für Unternehmen verbunden. Befindet sich z.B. ein Außendienstmitarbeiter geschäftlich im Ausland, um vor Ort einen Kunden zu betreuen, kann der Ausfall seines Laptops zu hohen Zeit- und dementsprechend auch Geldeinbußen führen, davon abgesehen, dass dem Kunden nicht der gewünschte Service

zuteil werden kann. Der Ersatz des Laptops ist dabei kein großes Problem, die Software kann eventuell aus dem Internet geladen werden. Die Konfiguration der frisch installierten Software stellt nun für einen Mitarbeiter, der diese Software nur bedient und niemals selbst installiert hat, schon größere Probleme dar. Der folgende Prozess, in möglichst kurzer Zeit eine kompetente Fachkraft zu erreichen, welche bei der Konfiguration die nötige Hilfestellung leisten kann, muss von Firmen unter beträchtlichem Aufwand gesichert werden.

Verknüpft man nun das Konzept des computergestützten Arbeitens mit den Möglichkeiten der Rich Internet Applications, erscheint es denkbar, benutzerfreundliche, kollaborative Anwendungen zu entwickeln, die den Beteiligten nicht nur Orts- und Zeit- sondern zudem auch noch Systemunabhängigkeit, zusätzlich zu den Vorteilen aus dem Bereich des „Change- and Configuration Management“, bieten.

1.2. Zielsetzung und Problematik

Im Markt der Rich Internet Application Frameworks stellen die größten Konkurrenten Google, Microsoft und Adobe dar. Die Frameworks haben unterschiedliche Reifegrade und verfolgen eigene Ansätze. Nachdem bereits Virginio Carfagno in seiner Diplomarbeit das Google Web Toolkit hinsichtlich grafiklastigeren Anwendungen evaluierte (Carfagno (2007)) und sich Mykhaylo Kabalkin mit der Migration einer Desktopanwendung auf eine Rich Internet Application auf Basis des Echo2 Frameworks beschäftigte (Kabalkin (2008)), ist der Autor dieser Arbeit davon überzeugt, dass sich eine Untersuchung des Flex 2.0 Frameworks von Adobe bezüglich der Entwicklung von kollaborativen Anwendungen als interessant und nützlich erweist.

Um Flex für Rich Internet Applications hinsichtlich der Nutzbarkeit für kollaborative Anwendungen zu bewerten, soll in dieser Arbeit zunächst ein kollaboratives System entworfen werden. Rich Internet Applications sollen die Lücke zwischen Internet- und Desktopanwendungen schließen, daher soll auch Flex den Anforderungen einer Desktopanwendung genügen. Der Entwurf dieses Systems soll sich nach gängigen, aus der Praxis bekannten Mustern richten. Es gilt also das Design einer kollaborativen Desktopanwendung zu entwickeln, die Anforderungen abzuleiten und diese dann auf Umsetzbarkeit für das Framework Flex zu testen.

Es ergibt sich die folgende Fragestellung:

> Kann anhand von Flex 2.0 eine Rich Internet Application entwickelt werden, wobei diese den Anforderungen einer kollaborativen Desktopanwendung genügt, welche auf Basis von gängigen Architektur- und Entwurfsmustern erstellt wurde?

Das Ziel dieser Arbeit liegt zum einen darin, das Design einer kollaborativen Anwendung zu entwickeln und zum anderen, die Mindestanforderungen dieses Designs zu bestimmen und auf Umsetzbarkeit in Flex zu prüfen.

1.3. Inhaltlicher Aufbau der Arbeit

In dem Einleitungskapitel (Kapitel 1) wurden die Motivation des Autors sowie die Problematik und das Ziel dieser Arbeit vorgestellt.

Im Grundlagenkapitel (Kapitel 2) sollen folgende für das Verständnis der Arbeit nötigen Themen behandelt werden: Traditionelle Webanwendungen, der Begriff Web 2.0, Rich Internet Applications und gängige Frameworks sowie abschließend Computer Supported Cooperative Work.

In Kapitel 3 soll die Vision der Referenzanwendung und die daraus resultierenden funktionalen und nicht-funktionalen Anforderungen beschrieben werden. Aus diesen Anforderungen sollen jene abgegrenzt werden, welche als Mindestanforderung der Anwendung identifiziert wurden und solche, die nicht für einen Minimalentwurf notwendig sind.

Das Design und der Entwurf der Referenzanwendung sowie die getroffenen Entscheidungen sollen in Kapitel 4 beschrieben werden. Dabei soll auf die Architektur- und Entwurfsmuster eingegangen werden, die in dem Design Anwendung finden. Am Ende des Kapitels soll ein Anforderungskatalog erstellt werden, welcher als Grundlage für die Machbarkeitsstudie dienen soll.

Im Flex-Kapitel (Kapitel 5) soll das nötige Verständnis des Flex Frameworks dargestellt werden, welches zur Überprüfung des Anforderungskataloges notwendig ist. Dabei sollen die historische Entwicklung, die programmiersprachlichen Grundzüge, spezifische Konzepte wie Datenbindung und Remote-Kommunikation sowie die Rahmenbedingungen bei der Softwareentwicklung anhand des Frameworks abhängig von Entwicklungsumgebung, Debugging, Logging und Unit-Testing dargestellt werden.

In Kapitel 6 soll der Anforderungskatalog aus dem Designkapitel anhand von Machbarkeitstests und auf Basis des Flex-Kapitels hinsichtlich der Umsetzbarkeit geprüft werden. Könnte eine der Anforderungen nicht erfolgreich umgesetzt werden, soll, wenn möglich, eine Alternative vorgestellt und eventuelle Vor- und Nachteile geschildert werden. Am Ende des Kapitels soll ein bewertendes Gesamt-Fazit erfolgen, welches Flex hinsichtlich des Anforderungskataloges beurteilt und diskutiert.

Zum Schluss soll in Kapitel 7 eine Zusammenfassung und ein Ausblick in die Zukunft erfolgen.

2. Grundlagen

2.1. Rich Internet Applications

Zuerst werden die klassischen Webanwendungen vorgestellt. Da mittlerweile der Begriff des Web 2.0 fälschlicherweise oft mit Rich Internet Applications gleichgesetzt wird, erfolgt eine Begriffsklärung. Abchließend wird auf Rich Internet Applications eingegangen.

2.1.1. Traditionelle Webanwendungen

Das Internet ist zu einem der wichtigsten Medien avanciert, wobei Webanwendungen aus der Internetlandschaft kaum wegzudenken sind. Der Erfolg lässt sich weitestgehend auf die hohe Erreichbarkeit potenzieller Interessenten zurückzuführen. Ein gutes Beispiel hierfür ist Amazon.com, ein Internetgeschäft, welches von Anfang an keinen physischen Laden besaß. Ein großer Teil des Erfolges von Amazon besteht in der Einfachheit, wie Besucher die gewünschte Ware finden können.

Dabei ist das weit bekannte Client-Server-Modell die Grundlage für die traditionellen Webanwendungen. Als wichtigstes Werkzeug zur Erstellung von Webanwendungen gilt HTML. Um die Anwendungen interaktiver zu gestalten, können über Module serverseitige Skripte oder dynamische Seiten erstellt werden.

Ein Nachteil des traditionellen Modells ist die Wartezeit auf dem Server, wenn eine neue Seite angefordert wird. Dies ist auf das synchrone request-reply-Verhalten zurückzuführen (Abbildung 2.1).

Zum einen werden viele redundante Daten (z.B. die Layoutinformationen einer Seite) ausgetauscht und zum anderen wird der Besuch einer Seite eher als „start and stop“ wahrgenommen. Gerade in zeitsensitiven Branchen wie in der Finanzwelt sind diese Anwendungsverhalten unerwünscht. Durch eine flächendeckende Abdeckung mit Breitbandanschlüssen und moderneren Computern erwarten die Nutzer des Internets auch mehr von Webanwendungen. Rich Internet Applications sollen nun diese Lücke zwischen Desktop-Applikationen und Webanwendungen füllen.

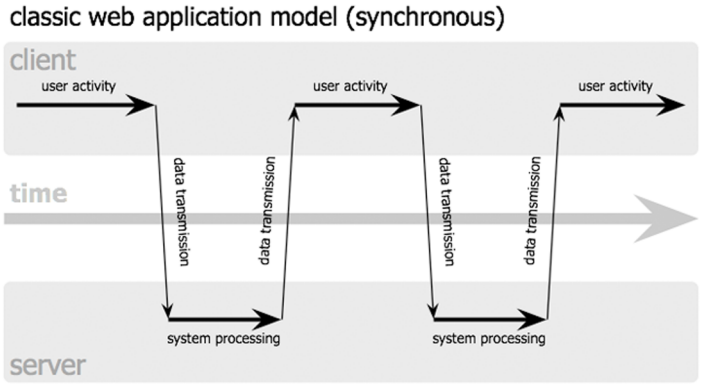

Abbildung 2.1.: Synchrone Kommunikation, (Quelle: Garrett (2005))

Obwohl diese neue Entwicklung oft in Zusammenhang mit dem Begriff Web 2.0 gebracht wird, um zu unterstreichen, dass es sich um „neuere“ und verbesserte Webentwicklung handelt, trifft dies meistens nicht zu (vgl. Nezlek (2007) S.414).

2.1.2. Web 2.0

Wie schon angedeutet, umfasst der Begriff Web 2.0 weit mehr als nur eine Technologie, welche Internetanwendungen benutzerfreundlicher gestaltet. Tim O´Reilly stellte im Laufe der ersten Web 2.0-Konferenz eine Liste von Charakteristiken zusammen, welche das Konzept des Web 2.0 verständlicher machen. Basierend auf dem von Tim O´Reilly erstellten Dokument (vgl. O'Reilly (2005)) lauten die Charakteristiken wie folgt:

The Web As Platform: Im „Web 2.0“-Konzept wird das Web mit einer Plattform gleichgestellt. Die PC-Plattform erlaubt z.B. die Ausführung einer Anwendung. Analog dazu ermöglich es das Web, über jedes Endgerät mit einem Web Browser auf Webanwendungen, die über die Plattform „Web“ zur Verfügung gestellt werden, zuzugreifen.

Harnessing Collective Intelligence: Der große Erfolg der Web 2.0-Webanwendungen liegt in der Nutzung der kollektiven Intelligenz. Dabei wird Benutzern die Möglichkeit geboten, sich an der Inhaltserzeugung und -erweiterung zu beteiligen. Durch die Beteiligung von immer mehr Nutzern erhöht sich entsprechend der Netzwerk-Effekt, dieser Effekt wird von O´Reilly als Schlüssel zur Marktdominanz im Web bezeichnet wird.

Wikipedia[1] setzte Eric Raymond's Maxime (ursprünglich in Bezug auf Open Source), „with enough eyeballs, all bugs are shallow“, hinsichtlich der Erstellung von Inhalten

[1] http://www.wikipedia.org/

um. Bei Wikipedia kann jeder Nutzer jeden Eintrag der Online-Enzyklopädie verändern.

Der Internethandel Amazon[2] verarbeitet hingegen die Verhaltensweise von Nutzern, um z.B. Käufern eines Artikels weitere Artikel vorzuschlagen, welche bereits andere Nutzer in Verbindung mit diesem Artikel gekauft haben. Weiterhin wird den Nutzern ermöglicht, Bewertungen und Erfahrungsberichte von Artikeln zu erstellen, welche anderen Amazon-Kunden als Kaufberatung zu Rate ziehen können.

Data is the Next Intel Inside: Die von den Nutzern zusammengetragenen Daten sind einzigartig und haben daher einen hohen Wert für ein Unternehmen. Ein Kennzeichen von Web 2.0 Anwendungen ist es, dass sie die Nutzung dieser einzigartigen Daten nicht verhindern, sondern im Gegenteil die Erweiterung und Verknüpfung mit anderen Inhalten fördern. Je mehr Benutzer die Daten verwenden und verfeinern, desto wertvoller werden sie.

End of the Software Release Cycle: Web 2.0-Anwendung stellen meist einen Service zur Verfügung wie z.B. Google seine Suche[3] oder Flickr[4] die Möglichkeit, Online-Photoalben zu erstellen. Der Service wird dabei während des laufenden Betriebes weiterentwickelt und der Benutzer kann in den Prozess meist passiv eingebunden werden. Dabei wird die Akzeptanz von Funktionalitäten anhand von Benutzerverhalten bewertet. Es können nicht genutzte oder fehlerhafte Funktionalitäten ohne die Auslieferung einer neuen Version entfernt werden. Bei Desktopanwendungen sind alle Veränderungen der Software an die Auslieferung einer Version gebunden und bedeuten zusätzlich auch einen Aufwand für den Nutzer.

Bei der Entwicklung von Web 2.0 Anwendungen werden die Dienste oft als „perpetual Beta“ entwickelt. Während der Dienst sich noch in der Entstehung befindet, wird er bereits Nutzern angeboten. Ein Beispiel ist z.B. Google Docs[5]. Manche Web 2.0 Anwendungen verstehen sich selbst jahrelang als Beta.

Lighweight Programming Models: Lightweight Programming Models unterstützen die lose Kopplung von Systemen und vereinfachen anderen Anwendungen die Integration dieser Dienste. O´Reilly verdeutlicht den Nutzen, anhand der zwei Varianten der Webservices von Amazon. Die erste Variante nutzt die strenge Formalien von SOAP, die zweite Variante hingegen wird einfach als XML über HTTP verbreitet. Der zweite Ansatz ist auch als „Representational State Transfer“ (REST) bekannt. Die strengere SOAP-Variante wird dabei von 5 Prozent genutzt (hauptsächlich von Amazon und Geschäftspartnern), während 95 Prozent den einfacheren REST-Dienst nutzen.

[2] http://www.amazon.com/
[3] http://www.google.com/
[4] http://www.flickr.com/
[5] http://docs.google.com/

Software Above the Level of a Single Device: Die Dienste einer Web 2.0 Anwendung sind nicht nur auf die PC-Plattform beschränkt, sondern können von den verschiedensten Endgeräten mit einem Internetzugang genutzt werden. Es ist daher keine spezielle Software nötig, um diese Dienste in Anspruch zu nehmen. Der mobile Markt gilt als sehr interessant, da durch stetige technologische Entwicklungen auch mehr Geräte an die Plattform Web angeschlossen werden.

Rich User Experiences: Über die Verwendung von JavaScript und DynamicHTML (DHTML) wurde bereits schon in traditionellen Webanwendungen ein höherer Grad an Bedienungskomfort erreicht. Allerdings konnte erst durch die Kombination von Technologien, welche später unter dem Begriff AJAX zusammengefasst wurden, eine Web Anwendung mit einer ähnlichen komfortablen Benutzerschnittstelle wie die einer Desktopanwendung entwickelt werden. Auf den Begriff AJAX wird im nachfolgenden Abschnitt eingegangen.

2.1.3. Rich Internet Applications (RIAs)

Der Begriff Rich Internet Applications wurde erstmals 2002 von Macromedia (mittlerweile Adobe), in Bezug auf die mit Flash erstellbaren Anwendungen und deren grafischen Benutzeroberfläche eingeführt. Rich Internet Applications folgen einem Modell, bei dem ein Teil der Informationen auf dem Benutzerrechner verarbeitet wird und der verbleibende Teil auf dem Server. Die Anfrage wird gestellt und der Server antwortet nach einer unbestimmten Zeit, also asynchron (Abbildung 2.2). Während dieser Zeit kann der Benutzer weiter die Applikation verwenden wie bei einer lokale Anwendung (vgl. Loosley, S. 4-5).

Rich Internet Applications können den Benutzer bei seinen Aktionen unterstützen. Obwohl der Schwerpunkt auf einer möglichst zeitnahen Ausführung von Aktionen liegt, kann der Benutzer über den Fortschritt eines sehr lang andauernden Prozesses informiert werden. Da nur bestimmte Teildaten vom Server angefordert werden, reduziert sich der Netzwerk-Verkehr. Es werden nicht mehr alle Informationen einer Seite immer wieder gesendet, sondern nur die gewünschten Teilinformationen. Diese Herangehensweise erlaubt kleinere, inkrementelle Updates, wodurch der erstellte Client sehr „leicht“ wird. Dadurch könnten auch Benutzer mit niedriger Bandbreite in den Genuss von RIAs kommen (vgl. Nezlek (2007) S.416).

Rich Internet Applications lassen sich in drei verschiedene Typen kategorisieren, abhängig davon, wie sie entwickelt und verteilt werden:

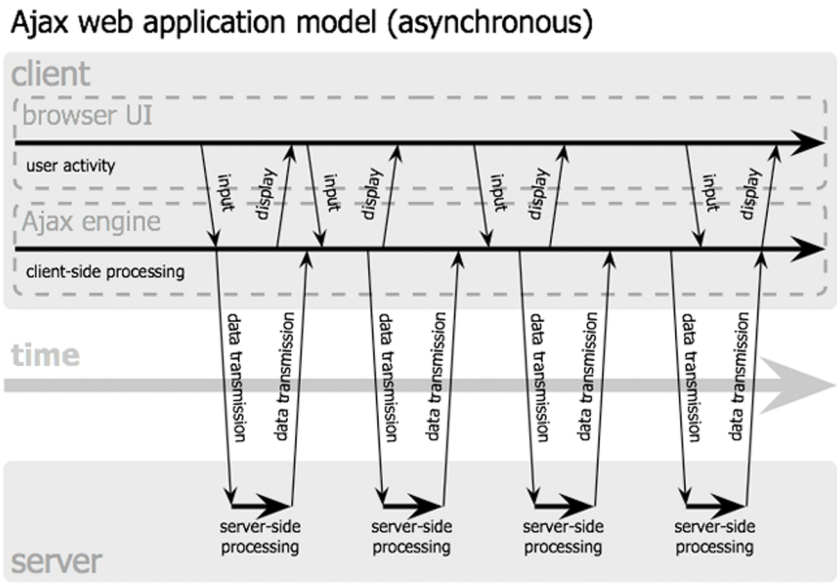

Abbildung 2.2.: Asynchrone Kommunikation, Quelle: Garrett (2005)

Plug-in und Stand-alone Eine Anwendung wird für eine dafür vorgesehene Plattform oder Umgebung erstellt und entweder als eine eingebettete Lösung verteilt oder als eine Stand-alone Anwendung vom Browser aus gestartet. Der große Vorteil des Plug-in basierten Ansatzes ist die einfache Entwicklung. Da man eine stabile Umgebung (auch „Sandbox“ genannt) vorfindet, kann man davon ausgehen, dass die Anwendung auf verschiedensten Systemen ohne Probleme lauffähig sein wird, solange das entsprechende Plug-in vorhanden ist. Wichtig ist dabei, dass die Umgebung Auskunft über die verfügbaren Dienste erteilen kann. Ein Beispiel hierfür ist Adobes Flex 2, welches auch in dieser Bachelorarbeit eingesetzt wird. Ein Beispiel für eine Stand-alone Lösung ist Java Web Start (vgl. Nezlek (2007) S.415).

Asynchronous JavaScript and XML (Ajax) Diese Anwendungen werden auch skript-basierte Rich Internet Applications genannt. Dabei kommt eine Kombination von XHTML/HTML, CSS, DOM und Javascript zum Einsatz. HTML und CSS werden zur Präsentation genutzt, durch JavaScript können asynchrone Anfragen an den Server gestellt und durch DOM-Scripting aktuelle Objekte manipuliert werden. Der Vorteil besteht darin, dass keine zusätzliche Software installiert werden muss. Leider haben einige dieser Technologien Kompatibilitätsprobleme in Bezug auf die Systemumgebung und den Browser. Da nicht einmal JavaScript vorausgesetzt werden kann, sollte eine Webanwendung erst traditionell erstellt werden, um die Grundfunktion zu garantieren. Sollte Javascript nun verfügbar sein, kann die Anwendung angereichert werden. Anzumerken ist, dass ein nicht ganz unerhebliches Wissen notwendig ist, um die vorhandenen Inkompatibilitäten zwischen den Browser

zu umgehen. Um eine bessere Kompatibilität zu erreichen, kommen Frameworks, wie z.B. das Google Web Toolkit, vermehrt zur Anwendung (vgl. Nezlek (2007) S.415).

Browser-basiert Bei diesem Ansatz baut die Benutzerschnittstelle auf XML auf. XML erlaubt dem Entwickler, die benötigten Elemente und Interaktionen in einem deklarativen Format zu definieren. Der größte Vorteil bei dieser Lösung ist, dass bestehende Standards wie CSS 1 & 2, DOM Level 1 & 2 und Javascript 1.5+ unterstützt werden und somit Plattformunabhängigkeit erreicht wird. Ein Beispiel für diesen Ansatz ist die Sprache XUL der Mozilla Foundation (vgl. Nezlek (2007) S.415).

2.2. Rich Internet Application-Frameworks

Da Flex nicht das einzige Framework im Bereich der Rich Internet Application ist, erfolgt hier eine kurze Vorstellung weiterer Frameworks. Es werden dabei zwei Java-basierte und ein Microsoft-Ansatz vorgestellt.

2.2.1. Ajax

Der Begriff Ajax wurde bereits in Abschnitt 2.1.3 erläutert. Es werden nun zwei Frameworks vorgestellt, welche es erlauben, eine Ajax-Anwendung anhand von Java zu entwickeln. Bei beiden Frameworks ist es dabei nicht nötig, JavaScript selbst programmieren zu können. Das nötige JavaScript wird über einen Compiler erzeugt.

Google Web Toolkit (GWT)

Das GWT ist ein Open Source Framework, basierend auf Java, welches dem Entwickler die Erstellung von Ajax-Webanwendungen ermöglicht. Die in Java geschriebene Anwendung wird über den GWT-Compiler in Browser-konformes HTML und JavaScript übersetzt und kann anschließend verteilt werden (vgl. Kabalkin (2008), S.23).

Architektur Abbildung 2.3 stellt die vier Hauptkomponenten des GWT dar.

GWT Java-to-JavaScript Compiler Übersetzt die Java-Komponenten nach JavaScript. Der GWT-Compiler wird dazu verwendet, um die Anwendungen im Webmodus laufen zu lassen. Hierbei wird die Anwendung als reines JavaScript und HTML kompiliert und dient zum Deployment.

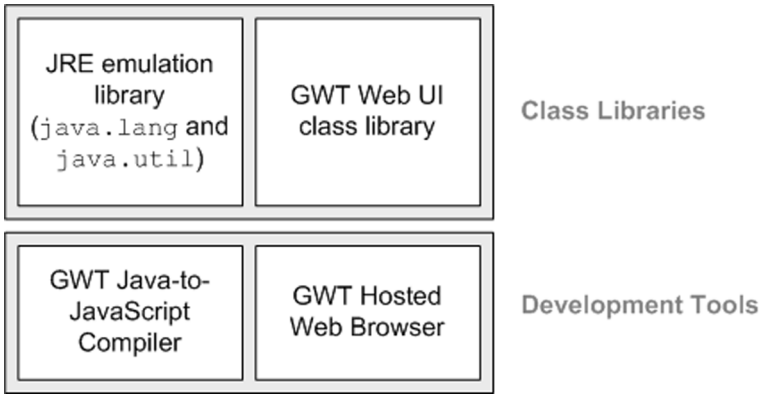

Abbildung 2.3.: Architektur des GWT,Quelle: Google

GWT Hosted Web Browser Erlaubt das Ausführen einer Anwendung im Hostedmodus. Dies bedeutet, dass der Code nicht übersetzt wird, sondern innerhalb der virtuellen Java Maschine läuft. Dies ermöglicht dem Entwickler, java-typische Debuggingunterstützungen zu nutzen.

JRE emulation library Enthält die Implementation der meistgenutzten Klassen der Java Standard Bibliotheken. Die emulation library umfasst dabei die meisten Klassen des java.lang-Paketes und einen Teil der Klassen des java.util-Paketes.

GWT Web UI class library Diese Bibliothek stellt dem Entwickler Komponenten für die Erzeugung von Benutzerschnittstellen zur Verfügung, z.B. enthält sie Buttons, Textboxen, Bilder oder Text.

Nach Kabalkin (2008) stellt die lückenhafte Unterstützung der Pakete einen großen Nachteil des GWTs dar. Weiterhin evaluierte Virginio Carfagno in seiner Diplomarbeit das GWT unter anderem hinsichtlich der Umsetzung einer grafisch mittelmäßigen 2D-Anwendung, wobei er zu dem Schluss kam, dass das GWT wegen Browserinkompabilitäten dieser Anforderung noch nicht genügt (vgl. Carfagno (2007) S. 87).

Echo2

Echo 2 ist ebenfalls ein Open Source Framework für die Entwicklung von Rich Internet Applications und von dem Unternehmen NextApp[6] entwickelt. Ein kommerzielles Plug-in für die Entwicklungsumgebung Eclipse namens EchoStudio2 wird von NextApp für das Framework angeboten. Das Plug-in erlaubt dabei die visuelle Entwicklung von Echo2-Anwendungen und ist mit einem User Interface Toolkit wie z.B. Java Swing zu vergleichen. Über den Einsatz

[6] http://www.nextapp.com/

von AJAX wird die Funktionalität dem einer Desktopanwendung angeglichen (vgl. Kabalkin (2008), S.27).

Architektur Abbildung 2.4 stellt die Echo2-Architektur dar. Es lassen sich drei getrennte Module identifizieren: Das Application Framework, die Web Rendering Engine und der Web Application Container. Zwischen den Modulen herscht nur ein geringer Grad von Kopplung. Die einzige Abhängigkeit besteht bei dem Web Application Container Modul, welches von dem Web Rendering Engine und dem Application Framework Modul abhängig ist, diese Abhängigkeit ist allerdings nicht bidirektional (vgl. NextApp (2007)).

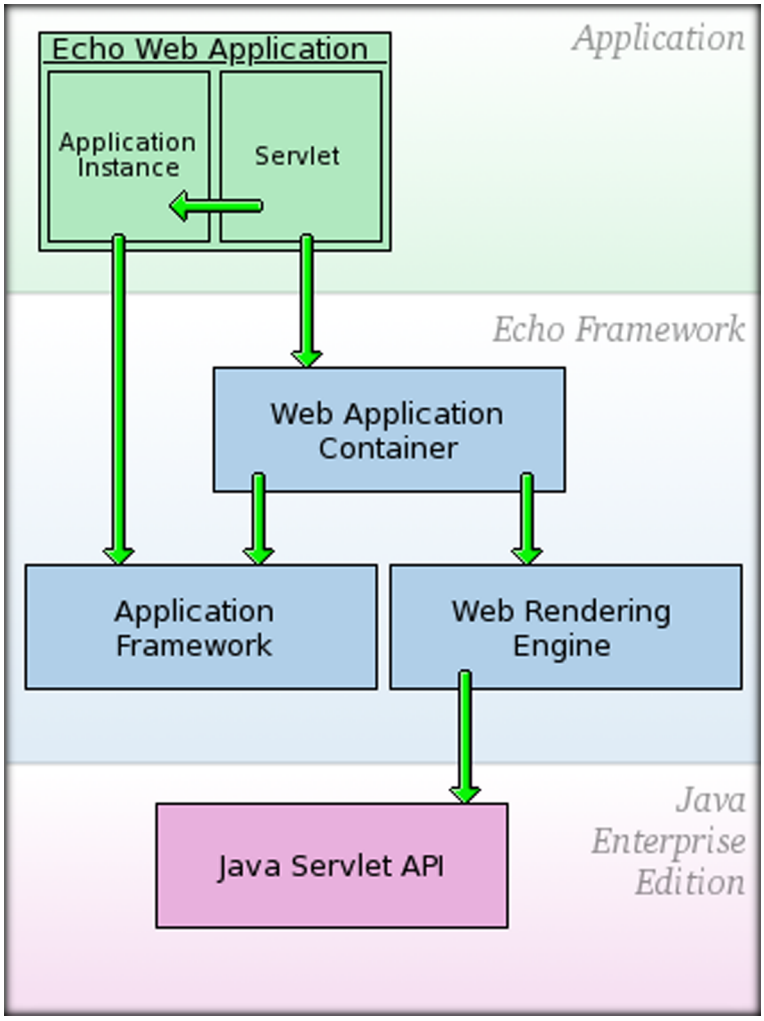

Abbildung 2.4.: Architektur des Echo2 Frameworks,Quelle: NextApp (2007)

Application Framework Das Application Framework Modul stellt eine API für die Verwaltung des Zustandes und der User Interfaces einer Anwendung zur Verfügung. Ein Entwickler kann über die Verwendung von Framework-Klassen (ApplicationInstance, Components, property objects) und die Nutzung der Event-/Listener APIS das Echo2 Framework ansprechen. Das Application Framework selbst stellt keine Funktionalität zur Verfügung, um Komponenten z.B. als HTML darzustellen oder um mit dem Browser zu kommunizieren. Es bietet allerdings Schnittstellen an, um eine Renderingkomponente zu integrieren. Dabei übernimmt der sogenannte Update Manager die Rolle eines Adapters. Änderungen, welche die UI-Komponenten betreffen, müssen der Renderingkomponenten mitgeteilt werden, ebenso müssen Benutzeraktionen welche von der Renderingkomponente stammen, an die Anwendung und dessen Komponenten weitergeleitet werden.

Web Rendering Engine Die Web Rendering Engine ist ein Satz von Werkzeugen, welche die Basis für Client-/Serverinteraktionen bereitstellt. Die Engine besteht aus einem serverseitigen, geschrieben in Java anhand der Verwendung der Java Servlet-API, und einem clientseitigen Teil, welcher in JavaScript geschrieben ist.

Server Engine Der serverseitige Part der Web Rendering Engine stellt ein HttpServlet zur Verfügung, welches dazu genutzt wird, um Clientanfragen zu verarbeiten. Wenn das Servlet eine HTTP-Anfrage erhält, wird es diese an das zugehörige Service-Objekt weiterleiten. Über Echo können dabei unzählige Service-Implementierungen wie z.B. das Darstellen des User-Interfaces als ein HTML-Dokument genutzt werden.

Client Engine Der clientseitige Teil der Web Rendering Engine ist eine JavaScript-Anwendung, welche innerhalb des Webbrowsers des Benutzers läuft. Die Client Engine ist dafür verantwortlich, Verbindungen mit der Server-Engine aufzubauen und beide zu synchronisieren, wenn der Benutzer Operationen auslöst.

Web Application Container Der Web Application Container ist eine Erweiterung der Web Rendering Engine. Diese ist für die Synchronisierung und Darstellung eines User Interfaces verantwortlich, welches mit dem Application Framework erstellt wurde. Als eine Erweiterung der Web Rendering Engine, bietet sie Erweiterungen für Server- und Client Engine, dementsprechend enthält sie serverseitigen Java- und clientseitigen JavaScript-Code.

2.2.2. Microsoft Silverlight 1.0

Silverlight ist eine cross-Browser, cross-Plattform Implementation des .NET-Frameworks, um Rich Internet Applications zu entwickeln. Um den hohen Grad von Kompatibilität zu erreichen, wird ein Plug-in eingesetzt, in welchem Silverlight-Anwendungen ausgeführt werden

können. Über XAML, eine deklarative Markupsprache, können Benutzerschnittstellen auf Basis der Windows Presentation Foundation beschrieben und erstellt werden. Die Windows Presentation Foundation stellt dabei grafische Komponenten und Effekte zur Verfügung, welche über HTML nicht realisiert werden könnten. Über die Verwendung von XAML wird eine Trennung von Oberflächen- und Anwendungslogik erreicht. Hinsichtlich der Anwendungslogik unterstützt Silverlight Version 1.0 nur die Verwendung von JavaScript. Erst Version 2.0, welche allerdings noch in der Beta-Phase steckt, kann das .NET-Framework für die Umsetzung genutzt werden. (vgl. Microsoft)

Architektur Abbildung 2.5 stellt die Architektur von Silverlight dar. Es wird dabei nur auf die Kernkomponenten eingegangen.

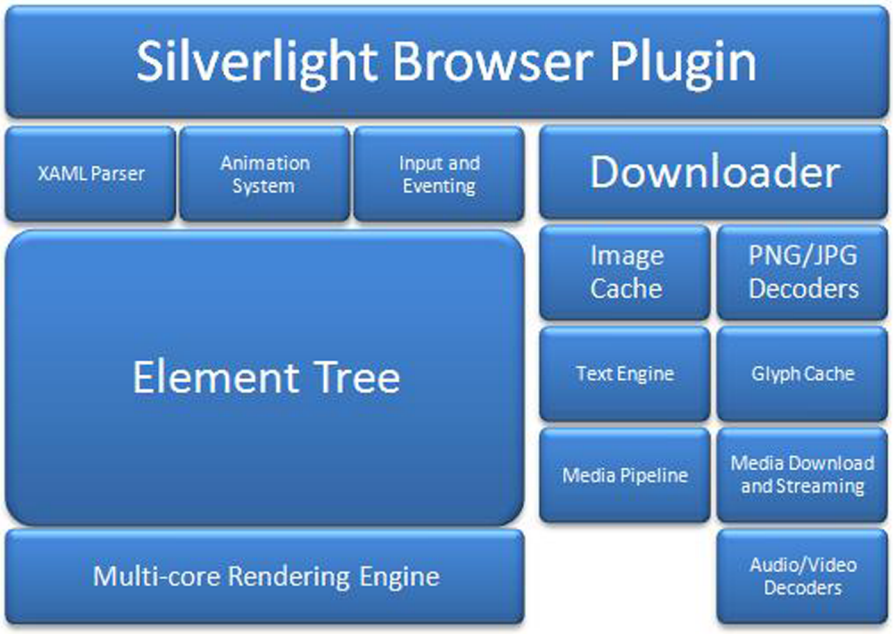

Abbildung 2.5.: Silverlight Architektur, (Quelle: Microsoft (2008))

Browser Plug-in Das Plug-in ermöglicht es, Silverlight-Anwendungen in verschiedensten Browsern auszuführen. Das Plug-in bietet hauptsächlich folgende Dienste an:

- Setzen oder Ändern des XAML-Inhalts, welcher durch die Laufzeitumgebung ausgeführt wird.
- Objekte aus der Laufzeitumgebung abfragen, damit diese anhand von JavaScript manipuliert werden können.
- Inhalte stückweise nachladen.

XAML-Parser Der Parser wird genutzt, um aus XAML einen Element Tree zu erstellen

Animation System Über Animation können grafische Komponenten bewegt, transformiert oder auch interaktiv werden. Um einer Komponente Interaktivität hinzuzufügen, wird sie mit einem Ereignis verbunden.

Input and Events Diese Komponente stellt verschiedene Ereignisstypen zur Verfügung, um dem Entwickler die Möglichkeit zu geben, auf Anwendungsereignisse und Benutzeraktionen zu reagieren.

Downloader XAML, Bilder, Schriften und andere Daten können, wenn nötig, über den Downloader heruntergeladen werden. Daher muss bei der Instanziierung der Anwendung nicht der gesamte Inhalt vorhanden sein.

Element Tree Der Element Tree stellt die Objekte dar, aus denen eine Silverlight-Anwendung besteht. Um einen Element Tree zu erstellen, kann der XAML-Parser verwendet werden. Änderungen an dem Element Tree können über JavaScript oder das Animation System durchgeführt werden.

Rendering Engine Die Rendering Engine ist für das eigentliche Zeichnen der Inhalte des Element Trees zuständig.

2.3. Computer Supported Cooperative Work (CSCW)

Der Bereich des CSCW ist ein interdisziplinärer Forschungsbereich bestehend aus Informatik, Psychologie, Soziologie und weiteren. CSCW umfasst die theoretischen Grundlagen, bzw. die Methodologien für Gruppenarbeit und deren Unterstützung durch einen Rechner (vgl. Borghoff (1998), Seite 92). Dabei steht bei CSCW die Kommunikation von Mensch zu Mensch im Vordergrund, welche technologisch unterstützt wird. Da der Schwerpunkt auf der menschlichen Interaktion liegt, ist CSCW eine Schnittmenge aus mehreren Forschungsbereichen.

Weiterhin muss der Begriff der „Gruppe“ noch genauer definiert werden. Als Gruppe wird eine Menge von Personen bezeichnet, welche sich zur Lösung eines gemeinsamen Problems zusammengeschlossen haben. Um dieses Problem zu lösen, ist der Informationsaustausch unter den Gruppenmitgliedern zwingend erforderlich (vgl. Borghoff (1998), S. 93).

Groupware umfasst die Systeme, welche Gruppenarbeit unterstützen. Allerdings wird der Begriff Groupware mittlerweile eher in Verbindung mit Kalender-, Kontakt- und Projektmanagementsystemen wie Lotus-Notes in Verbindung gebracht. Diese Anwendungen stellen nur einen Teilbereich der Groupware-Systeme dar. Um Missverständnisse zu vermeiden, sollen

Anwendungen, die in den Bereich der Groupware fallen, in dieser Arbeit unter dem Begriff der CSCW-Systemen behandelt werden.

CSCW-Systeme unterscheiden sich von den leicht missverständlichen Groupware-Lösungen in dem Sinne, dass ein CSCW-System eine wirklich gemeinsame Umgebung bereitstellt. Bei der gemeinsamen Umgebung ist es wichtig, dass ein Gruppenbewusstsein vorhanden ist. Dies äußert sich z.B. bei einem Mehrautorensystem darin, dass dem Benutzer angezeigt wird, wie viele Autoren gerade gleichzeitig an einem Dokument arbeiten und dass der Benutzer über Änderungen automatisch informiert wird. Dies steht im Gegensatz zu einem Textsystem, bei dem mehrere Benutzer nacheinander dasselbe Dokument verändern, da die Aktionen an sich isoliert von einander bleiben (vgl. Borghoff (1998), S. 96).

CSCW-Systeme werden entweder nach Raum und Zeit, nach funktionalen Aspekten oder nach dem 3K-Modell klassifiziert. Nach Johansen (1988) wird bei der Klassifikation nach Raum und Zeit zeitlich zwischen synchroner sowie asynchroner Kommunikation unterschieden. Synchron bedeutet in diesem Fall, dass die Teilnehmer gleichzeitig interagieren wie z.B. bei einem Telefonat, und asynchron bedeutet, dass die Kommunikation zeitlich versetzt erfolgt wie z.B. per E-Mail. Die räumliche Verteilung wird dabei anhand der Örtlichkeit der Gruppenmitglieder unterschieden. Grudin (1994) erweitert die Raum-Zeit-Matrix noch um den Punkt der Planmäßigkeit.

Abbildung 2.6 gibt für die verschiedenen Klassifikationen jeweils ein Beispiel für die Zusammenarbeit. Aus den Klassen können sich bereits für die entsprechenden Systemkomponenten einige Anforderungen ableiten lassen, ein vollständiges CSCW-System sollte allerdings mindestens die vorhersehbaren Klassen einschließen.

Die Klassifizierung nach Raum und Zeit soll in Abschnitt 3.4 erfolgen, nachdem die funktionalen und nicht-funktionalen Anforderungen definiert wurden.

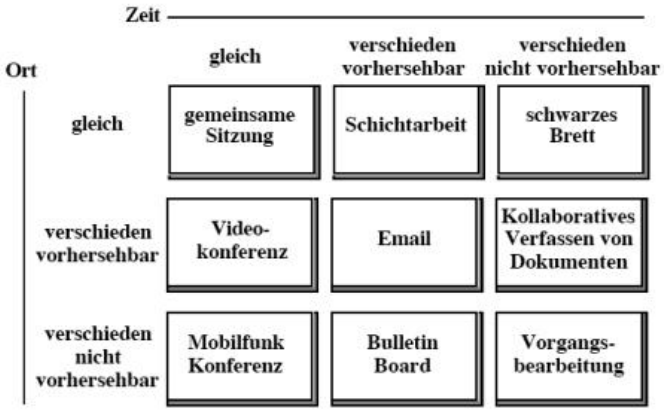

Abbildung 2.6.: CSCW-Klassifizierung, aus

Neben der Klassifizierung nach Raum und Zeit gibt es nach Ellis u. a. (1991) eine funktionelle Klassifizierung:

- **Nachrichtensysteme**: Diese Systeme dienen Gruppen dazu, Textnachrichten, Graphiken und mittlerweile auch Videos asynchron auszutauschen. In diesen Bereich fallen E-Mail-, Foren- und Blogsysteme, aber heutzutage auch Video-Plattformen wie z.B. YouTube.
- **Gruppeneditoren**: Ein Gruppeneditor ermöglicht es einer Gruppe ein gemeinsames Dokument zu verfassen. Dabei arbeiten die einzelnen Teilnehmer nicht isoliert voneinander, sondern es existiert ein wirkliches Gruppenbewusstsein. Die Editoren lassen sich in Realzeiteditoren und asynchrone Editoren unterteilen.
- **Gruppenunterstützungssysteme und elektronische Sitzungsräume**: Diese Systeme unterstützen eine Gruppe bei der Lösung von unstrukturierten Problemen. Dabei ist es das Ziel die Entscheidungsfindung effizienter zu gestalten. Gruppenunterstützungssysteme werden oft als elektronischer Sitzungsraum realisiert.
- **Konferenzsysteme**: Sie werden nach Ellis u. a. (1991) in 3 Kategorien unterteilt: Realzeitrechnerkonferenz, Telekonferenz und Desktopkonferenz. (vgl. Borghoff (1998), S.124) führt noch eine weitere Art auf, die Nicht-Realzeitrechnerkonferenz: Diese beschreibt eine Konferenz, welche mit Hilfe asynchroner Kommunikation, meist per Email, abgehalten wird. Dabei werden die Emails durch ein System gefiltert, sortiert oder verwaltet und der laufenden Konferenz zugeordnet.
- **Agentensysteme**: Sie nehmen aktiv an der Gruppenarbeit teil. Intelligente Agenten werden bereits in Computerspielen häufig eingesetzt, eignen sich aber auch für andere Aufgaben. Sie nehmen stellvertretende Funktionen ein wie z.B. ein ModeratorAgent, welcher eine Diskussion leitet und darauf achtet, dass alle Beteiligten gleichberechtigt zu Wort kommen können.
- **Koordinationssysteme**: Sie dienen der Koordinierung der Gruppenteilnehmer, um das gemeinsame Ziel strukturierter und effizienter zu erreichen. Man unterscheidet dabei 4 Systemarten: formularbasiert, prozedurorientiert, konversationsorientiert und kommunikationsorientierte Systeme.

Eine weitere Klassifizierung nach dem so genannten 3K-Modell wurde von Teufel S. (1995) durchgeführt. Es siedelt ein CSCW-System innerhalb der drei Schwerpunkten an: Kommunikation, Koordination und Kooperation. Dabei steht bei der Kommunikation die Verständigung von Personen im Mittelpunkt. Die Koordination hat die effiziente Organisation von Ressourcen und von einander abhängigen Tätigkeiten als Schwerpunkt. Die Kooperation unterstützt die Benutzer bei der Verfolgung eines gemeinsamen Zieles. Grafisch lassen sich somit die CSCW-Systeme innerhalb eines Dreiecks entsprechend ihrer K-Orientierung platzieren.

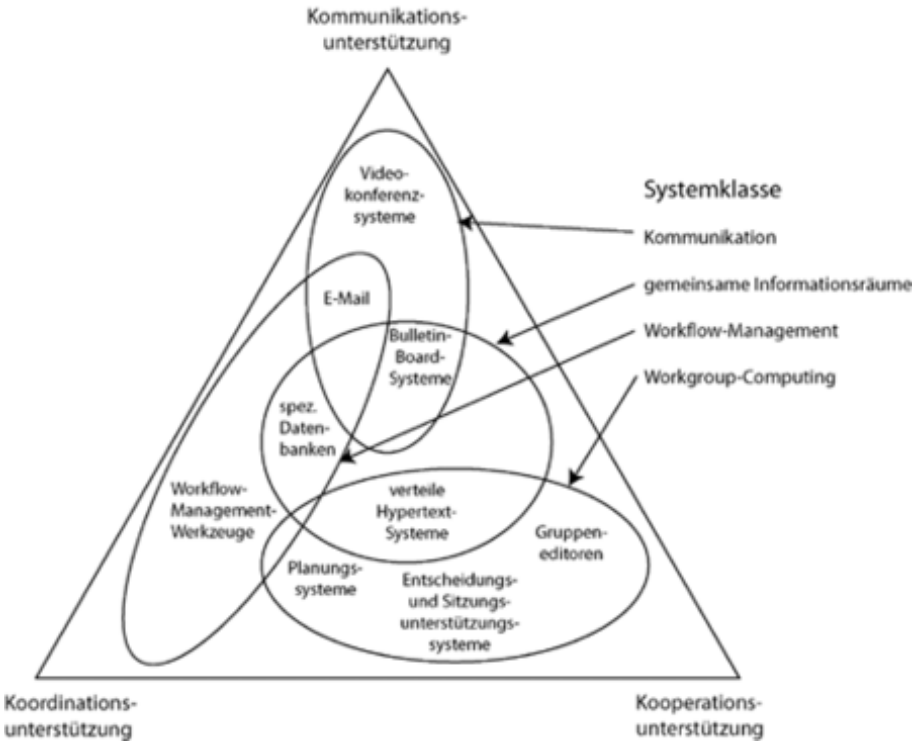

Abbildung 2.7.: 3K-Modell, Quelle: Teufel S. (1995)

3. Analyse

In diesem Kapitel soll die kollaborative Anwendung vorgestellt werden, nach welcher die Referenzarchitektur im Kapitel Design entworfen werden soll. Dabei soll eine Vision der kollaborativen Anwendung, die nötigen Anhaltspunkte für funktionale und nicht-funktionale Anforderungen liefern. Zur Definition der Anforderungen soll dabei auf die Verwendung von Anwendungsfällen verzichtet werden, da die kollaborative Anwendung nicht das Ergbniss eines Beratungsprozesses zwischen Software-Architekt und Kunde ist, sondern als Rahmen für die Referenzarchitektur dienen soll. Einige Grundanforderungen wurden bereits im Einführungskapitel in Form einer Fragestellung definiert und sollen in den Anforderungsabschnitten aufgegriffen werden. Die funktionalen und nicht-funktionalen Anforderungen sollen beschrieben und, wenn möglich, mit einem tabellarischen Ablauf verdeutlicht werden.

3.1. Vision

Aus der Fragestellung in Abschnitt 1.2 geht hervor, dass es sich um eine kollaborative Anwendung handeln soll. An dieser Stelle soll die Funktionalität des Systems weiter ausgeführt werden, um ein in sich sinnvolles und geschlossenes System als Basis für den Entwurf der Referenzarchitektur zu gewährleisten.

3.1.1. Anwendungsbeschreibung

Bei dem System soll es sich um ein Dokumentenverwaltungssystem handeln, welches einem Benutzer erlaubt, Dokumente zeitlich synchron mit mehreren Beteiligten gleichzeitig zu editieren. Dokumente sollen sich in Schrift- und Grafikdokumente unterteilen. Bei einem Schriftdokument soll es sich um ein strukturierbares Dokument handeln, welches anhand von Strukturelementen und Textabschnitten aufgebaut ist. Bei Schriftdokumenten soll erkenntlich sein, welche Textabschnitte und Strukturelemente in Bearbeitung sind. Zudem sollen Benutzer Textabschnitte und Strukturelemente kommentieren können. Das Schriftdokument soll anzeigbar sein. Es soll mit der Anwendung möglich sein, kleine Skizzen zu erstellen. Zudem sollen vorhandene Grafiken bearbeitet und gespeichert werden können.

Um das gemeinsame Arbeiten an einem Dokument zu erleichtern, sollen die Benutzer während der Zusammenarbeit per Webcam miteinander kommunizieren können. Zudem sollen sonstige Informationen in Form von Dateien schnell zwischen den Benutzern ausgetauscht werden können, indem sie einem virtuellen Raum hinzugefügt und dort von jedem Teilnehmer abgerufen werden können.

Weiterhin soll die Anwendung als eine soziale Plattform fungieren. Die Benutzer sollen nicht nur von der Existenz anderer Benutzer erfahren, während sie miteinander zusammenarbeiten. Es soll ein Austausch über die Kooperation hinaus möglich sein. Um dies zu unterstützen, soll ein Nachrichtensystem integriert werden.

Die Software soll auf verschiedensten Geräten mit verschiedenen Betriebssystemen lauffähig sein.

3.2. Funktionale Anforderungen

Es wurden folgende funktionale Anforderungen bestimmt, um die aus der Anwendungsbeschreibung in Abschnitt Vision vorgestellte Anwendung realisieren zu können:

F:D:A1 Dokument erstellen

F:D:A2 Dokument löschen

F:SD:A1 Schriftdokument anzeigen

F:GD:A1 Grafikdokument bearbeiten

F:S:A1 Strukturelemente erstellen

F:S:A2 Strukturelemente bearbeiten

F:S:A3 Strukturelemente verschieben

F:S:A4 Strukturelemente sperren

F:S:A5 Strukturelemente kommentieren

F:S:A6 Strukturelemente löschen

F:T:A1 Textabschnitt erstellen

F:T:A2 Textabschnitt bearbeiten

F:T:A3 Textabschnitt verschieben

F:T:A4 Textabschnitt sperren

F:T:A5 Textabschnitt kommentieren

F:T:A6 Textabschnitt löschen

F:DT:A1 Daten hochladen

F:DT:A2 Daten speichern

F:KW:A1 Kommunikation über Webcam

F:KN:A1 Kommunikation über Nachrichtensystem

Es folgen die Beschreibungen der funktionalen Anforderungen für Dokumente, Schriftdokumente, Grafikdokumente, Strukturelemente, Dateien und Kommunikation. Die Beschreibungen der funktionalen Anforderungen von Textabschnitten finden sich im Anhang im Abschnitt A.1 wieder, da sie sich nur minimal von denen der Strukturabschnitte unterscheiden.

Als Akteure der Ablauftabellen repräsentiert „B“ den Benutzer und „S“ das System.

3.2.1. Dokument erstellen (F:D:A1)

In dieser Ablauftabelle wird exemplarisch ein Schriftdokument erstellt. Die Erstellung eines Schriftdokumentes unterscheidet sich dabei von der Erzeugung eines Grafikdokumentes nur durch dokumententypspezifische Grundeinstellungen.

B	öffnet den Knoten in dem das Dokument angelegt werden soll
B	öffnet das Kontextmenü des Knotens und wählt den Punkt „Neues Dokument erstellen“ und wählt den Punkt „Schriftdokument erstellen“
S	öffnet ein Menü um die Grundeinstellungen eines Schriftdokumentes fetzulegen (siehe Grundeinstellung Dokument , Abschnitt 3.2.2)
B	wählt die gewünschten Einstellungen aus und bestätigt den Dialog
S	erstellt ein neues Dokument und fügt eine Verbindung zu dem gewählten Knoten hinzu
S	führt alle Operationen gemäß der Grundeinstellungen aus
B	sieht das neu angelegte Dokument in der Übersicht

3.2.2. Grundeinstellung Dokument

Die Grundeinstellungen beinhalten den Namen und eine einfache Rechtevergabe über das Dokument. Dabei wird das Dokument einer Gruppe zugeordnet, welche uneingeschränkten Zugriff auf das Dokument hat. Jeder Benutzer, egal welcher Gruppe er angehört, kann jeder

Zeit alle Dokumente namentlich in der Verzeichnisstruktur sehen, um Konflikte durch Löschen zu vermeiden. Es wird dabei von einer „gutwilligen“ Arbeitsumgebung ausgegangen, bei der die verschiedenen Benutzer voneinander nichts zu verbergen haben.

3.2.3. Dokument löschen (F:D:A2)

B	öffnet den Knoten in dem das Dokument liegt
B	öffnet das Kontextmenü des Dokuments und wählt den Punkt „Dokument löschen“
S	prüft ob das Dokument gelöscht werden kann, führt Löschung durch oder gibt Fehlermeldung aus

3.2.4. Schriftdokument anzeigen (F:SD:A1)

Es wird ein Fenster geöffnet in dem das komplette Schriftdokument dargestellt wird.

3.2.5. Grafikdokument bearbeiten (F:GD:A1)

Abschnitt „Element synchron bearbeiten“ (3.2.12) stellt den generellen Ablauf des synchronen Bearbeitens dar und muss zuerst durchlaufen werden. Danach kann der Benutzer die Grafik anhand eines einfachen Editors bearbeiten.

3.2.6. Strukturelement erstellen (F:S:A1)

Ein Strukturelement kann entweder ein Eltern-Strukturelement besitzen, also ist es ein Kindelement eines anderen Knotens, oder es ist ein Wurzel-Strukturelement des Dokuments.

B	öffnet das Kontextmenü des Elements und wählt „Strukturelement hinzufügen“ aus
S	Strukturelement-erstellen-Dialog erscheint
B	wählt einen Strukturelement-Typ aus, legt den Namen fest und klickt auf „Erstellen“
S	fügt das Strukturelement hinzu

3.2.7. Strukturelement bearbeiten (F:S:A2)

Abschnitt „Element synchron bearbeiten“ (3.2.12) stellt den generellen Ablauf des synchronen Bearbeitens dar und muss zuerst durchlaufen werden. Danach kann der Benutzer die synchron editierbaren Elemente des Strukturelements, den Namen, in einem Editor bearbeiten. Der Editor erlaubt es, den Text zu formatieren.

3.2.8. Strukturelement verschieben (F:S:A3)

Es ist möglich, ein Strukturelement samt seinen Kindelementen einem anderen Strukturelement oder dem Schriftdokument zuzuordnen. Dabei kann es zu einem Wurzel- oder zu einem neuen Kindelement eines anderen Strukturelementes werden. Eine Verschiebung darf nur stattfinden, wenn das Strukturelement und keins der Kindelemente sich in Bearbeitung findet.

B	öffnet das Kontextmenü des Elements und wählt „verschieben“ aus
S	prüft, ob ein Element zur Zeit bearbeitet wird, falls ja, erfolgt eine Meldung
S	Strukturelement-verschieben-Dialog erscheint
B	wählt das gewünschte Oberelement aus und klickt auf „verschieben“
S	verschiebt das Strukturelement

Der Strukturelement-Verschieben-Dialog wird als eine Baumstruktur des Dokumentes realisiert, wobei das zu verschiebende Element samt seiner Kindelemente nicht angewählt werden kann.

3.2.9. Strukturelement sperren (F:S:A4)

Ein Benutzer kann ein Strukturelement sperren, solange nicht bereits ein anderer Benutzer eine Sperre auf das Element besitzt. Diese Sperrung bewirkt, dass kein anderer Benutzer das Strukturelement oder eines der Kindelemente bearbeiten kann. Befindet sich das Strukturelement oder eines seiner Kindelemente in Bearbeitung, kann keine Sperre gelegt werden. Es ist möglich, ein Strukturelement zu sperren, auch wenn eines seiner Kindelemente bereits von einem anderen Benutzer gesperrt wurde. In diesem Falle bleibt die Sperre des Kindelements unangetastet und fällt erst bei einer Aufhebung der Sperre unter die Sperre des Elternelements. Der Benutzer kann einen Grund für die Sperre und einen voraussichtlichen Zeitpunkt eintragen, an dem der Bereich wieder frei verfügbar sein wird. Es wird eine Nachricht an alle Personen geschickt, welche das Dokument bearbeiten können. Es können weiterhin Kommentare für die Elemente erstellt werden.

3.2.10. Strukturelement kommentieren (F:S:A5)

1	B	klickt auf das Kommentar-Symbol neben dem zu kommentierenden Strukturelement
2	S	es erscheint die Kommentarübersicht; falls noch kein Kommentar existiert, folgt direkt Schritt 4
3	B	es ist möglich, einen komplett neuen Kommentar zu erstellen oder einen Kommentar zu einem vorherigen Kommentar zu schreiben
4	S	das Übersichtsfenster wird abgeblendet und es erscheint das Kommentar-Erstellen-Fenster.
5	B	fügt seinen Kommentar in das entsprechende Feld ein und klickt auf „Kommentar einfügen“

3.2.11. Strukturelement löschen (F:S:A6)

Ein Strukturelement kann nur gelöscht werden, wenn keins der Kindelemente in Bearbeitung oder gesperrt ist.

B	öffnet das Kontextmenü des Strukturelements und wählt „löschen“ aus
S	überprüft, ob das Strukturelement gelöscht werden darf, falls nicht, wird eine Fehlermeldung angezeigt
B	sieht die Übersicht des Schriftdokuments

3.2.12. Element synchron bearbeiten

1	B	wählt das zu bearbeitende Element, öffnet das Kontextmenü und wählt „bearbeiten“ aus
2	S	falls es eine laufende Arbeitssitzung gibt, tritt der Benutzer dieser bei (Abschnitt Element synchron bearbeiten: Arbeitssitzung beitreten, 3.2.14), ansonsten wird eine Arbeitssitzung für das Element erstellt (Abschnitt Element synchron bearbeiten: Arbeitssitzung erstellen, 3.2.13)

3.2.13. Element synchron bearbeiten: Arbeitssitzung erstellen

Ein Benutzer möchte ein Element bearbeiten und es existiert keine Arbeitssitzung, daher wird eine Arbeitssitzung für ihn angelegt. Andere Benutzer sehen bereits in der Übersicht, dass das Element zur Zeit bearbeitet wird.

3.2.14. Element synchron bearbeiten: Arbeitssitzung beitreten

Der Benutzer tritt der Arbeitssitzung bei. Anderen Benutzern wird die Anwesenheit des neuen Benutzers durch eine Nachricht mitgeteilt. Ein Symbol verdeutlicht, wer bei der Arbeitssitzung zugegen ist.

3.2.15. Element synchron bearbeiten: Arbeitssitzung verlassen

Der Benutzer verlässt die Arbeitssitzung. Der Benutzer, der zuletzt die Sitzung verlässt, bestimmt, welche temporären Daten noch gespeichert werden sollen (Siehe Abschnitt: Daten speichern, 3.2.16). Wenn der letzte Benutzer die Arbeitssitzung verlässt, wird das Element gespeichert. Sollten alle Benutzer die Verbindung abbrechen, wir die Arbeitssitzung geschlossen und das Element in dem zuletzt vorhandenen Zustand gespeichert.

3.2.16. Daten hochladen (F:DT:A1)

Ein Benutzer kann Daten mit anderen Benutzern teilen, welche mit derselben Arbeitssitzung verbunden sind. Zusätzlich können Daten zu Dokumenten, Strukturelementen und Textabschnitten hinzugefügt werden, ohne dass sie zur Zeit in Bearbeitung sind. Dabei kann er im Vorwege entscheiden ob die Daten permanent oder nur für die Dauer der Arbeitssitzung gespeichert werden sollen. Wenn es sich um permanente Daten handelt, werden diese sofort gespeichert. Die Daten werden den Benutzern in einem extra Fenster dargestellt. Es wird dabei zwischen dauerhaften und temporären Daten visuell unterschieden.

3.2.17. Daten speichern (F:DT:A2)

Die Daten werden dem Element zugeordnet und gespeichert.

3.2.18. Kommunikation über Webcam (F:KW:A1)

Während einer Arbeitssitzung können die Benutzer sich über eine Webcam verständigen.

3.2.19. Kommunikation über Nachrichtensystem (F:KN:A1)

Über das Nachrichtensystem können die Benutzer miteinander in Kontakt treten. Zusätzlich kann das System über Nachrichten die Benutzer über für sie relevante Vorgänge informieren.

3.3. Nicht-Funktionale Anforderungen

Die Fragestellung aus 1.2 schreibt vor, dass die Rich Internet Application einer Desktopanwendung genügen soll. Da Rich Internet Applications einen zentralen Server nutzen, soll es sich bei dem kollaborativen System um eine Client-Server-Anwendung handeln. Flex fällt in die Kategorie der Plug-in Rich Internet Applications, daher muss der Benutzer das Flash Plug-in Version 9 für seinen Web Browser installiert haben. Zusätzlich zu dem Plug-in muss der Internetanschluss des Benutzers über ausreichend Bandbreite verfügen, um einen normalen Betrieb der Anwendung zu gewährleisten. Da die objektorientierte Programmiersprache Java in der Softwareentwicklung als solides Werkzeug gilt, soll das Design der Anwendung in Java umsetzbar sein. Betrachtet man die Vision der Anwendung des Abschnitts 3.1, soll ein System erstellt werden, welches die synchrone Bearbeitung von Dokumenten erlaubt. Während der Bearbeitung ist es wichtig, dass die Änderungen eines Dokumentes direkt verarbeitet und angezeigt werden können, wie auch bei einer Desktopanwendung.

Folgende nicht-funktionale Anforderungen konnten identifiziert werden:

NF:A1 Design soll als Client-Server-Anwendung entworfen werden.

NF:A2 Ein Benutzer, der die Anwendung nutzt muss einen Web Browser mit installiertem Flash Plug-in (Version 9) besitzen.

NF:A3 Der Internetzugang des Benutzers soll über ausreichend Bandbreite verfügen, um einen normalen Betrieb der Anwendung zu gewährleisten.

NF:A4 Design soll in Java umsetzbar sein.

NF:A5 Änderungen sollen während einer Arbeitssitzung sollen direkt verarbeitet und angezeigt werden.

3.4. Klassifizierung der kollaborativen Anwendung

Die in dieser Arbeit vorgestellte Anwendung lässt sich in dieser Klassifikation nach Raum und Zeit (siehe Abbildung 2.6 auf Seite 25) bei „Ort - verschieden vorhersehbar“ und „Zeit - gleich“ positionieren, da der Schwerpunkt auf der räumlich unabhängigen, gleichzeitigen Bearbeitung eines Dokumentes liegt. Nach dem 3K-Modell, siehe Abbildung 2.7 auf Seite 27, lässt sich die Anwendung zwischen Kommunikations- und Koordinationsunterstützung einordnen, da das gemeinsame Editieren eine Dokumentes durch die Verwendung von Webcam und Nachrichtensystem unterstützt wird.

3.5. Abgrenzung

Da die die Evaluation von Flex Gegenstand dieser Arbeit ist, und Flex innerhalb einer Client-Server-Anwendung nur clientseitig eingesetzt wird, soll der Entwurf der Server-Anwendung nur die Komponenten und eine Übersicht über deren Dienste enthalten. Anschließen soll eine weitere Abgrenzung von funktionalen und nicht-funktionalen Anforderungen erfolgen und festgelegt werden, welche Anforderungen in den Entwurf des Systems einfließen.

Nicht-funktionale Anforderungen Aus Abschnitt 3.3 wird die Anforderung **NF:A5** nicht in die Überlegungen des Design einbezogen. Im Fall der Anforderung **NF:A5** wird dies mit dem Umstand begründet, dass die Architektur nicht umgesetzt wird sondern nur als Referenz für die Anforderungen dienen soll. Die Anforderung **NF:A5** kann daher nicht überprüft werden. Zudem sind die Einflussfaktoren der Anforderungen **NF:A5** stark system- und technologieabhängig.

Die nicht-funktionale Anforderung **NF:A3** wird vorausgesetzt. Es soll aber nicht weiter untersucht oder diskutiert werden, wie viel Bandbreite vorhanden sein muss, um den Betrieb einer kollaborativen Anwendung zu gewährleisten. Dieser Bereich könnte im Rahmen einer Untersuchung hinsichtlich der Qualitätsmerkmale von Rich Internet Applications geschehen.

Alle weiteren nicht-funktionalen Anforderungen sollen in den Entwurf der Referenzarchitektur einfließen.

Funktionale Anforderungen Bei der Anforderung **F:KW:A1** (3.2.18) soll grundsätzlich zu erkennen sein, wie die Kommunikation per Webcam in der Anwendung gehandhabt wird. Eine tiefgehende Behandlung des Themas Kommunikation per Webcam würde den Rahmen dieser Arbeit überschreiten, da die Thematik und die zur Verfügung stehenden Hard- und Softwaresysteme betrachtet werden müssten.

Die Anforderung **F:GD:A1** (Grafikdokument bearbeiten, 3.2.5) nutzt dieselbe Funktionalität wie die Anforderungen **F:S:A2**(siehe 3.2.7) und **F:t:A2**. Grafikdokumente sollen zwar in dem Entwurf vernachlässigt werden, dieser sollte aber so gestaltet werden, dass eine nächträgliche Erweiterung vorstellbar wäre.

Die weiteren funktionalen Anforderungen sollen für das Design berücksichtigt werden.

4. Design

Im folgenden Abschnitt soll eine kurze Übersicht und Einordnung der verwendeten Architekturmuster gegeben werden. Anschließend sollen das Schichtenmuster, die Aufteilung von Schichten bei Client-Server-Anwendungen, die Kommunikation zwischen Client und Server, der Aufbau der graphischen Benutzerschnittstelle des Clients und die verwendeten Entwurfsmuster beschrieben werden. Am Ende eines Architektur- oder Entwurfsmusters soll angegeben werden, wo es sich in dem Design der Anwendung wiederfindet.

Zuerst soll der Server und dann der Client anhand der verwendeten Schichten und den zugehörigen Komponenten dargestellt werden. Es soll bei den entsprechenden Komponenten ein Bezug zwischen den vorgestellten Architektur- und Entwurfsmusters hergestellt werden. Nach der Vorstellung von Client und Server soll anhand von Sequenzdiagrammen das Zusammenspiel der beiden Anwendungen verdeutlich werden. Am Schluss des Designkapitels sollen die Anforderungen aus dem Design abgeleitet und zusammengefasst werden.

4.1. Muster

In der Informatik kann man Muster in Entwurfs- und Architekturmuster unterteilen. Entwurfsmuster helfen dabei vor allem bei der Implementierung von Komponenten und sind mit wenigen Ausnahmen nicht auf Architekturebene erkennbar. In Abschnitt 4.6 werden die in diesem Design verwendeten Entwurfsmuster vorgestellt. Architekturmuster beschreiben dagegen die Relation und Interaktion von Komponenten. In einem System werden meist mehrere Architekturmuster zum Einsatz gebracht (vgl. Siedersleben (2003) S. 104).

Die in diesem Design zur Anwendung gekommenen Architekturmuster können nach Buschmann u. a. (2001) in drei Kategorien eingeteilt werden:

From Mud to Structure(„Struktur ins Chaos“) Muster in dieser Kategorie verhindern eine unkontrollierte Masse von Komponenten und Objekten. Im Speziellen unterstützen sie die Zerlegung eines Gesamtsystems in kooperative Subsysteme. Aus dieser Kategorie wird das Schichten-Muster eingesetzt.

Distributed Systems („Verteilte Systeme“) Aus dieser Kategorie wird das Broker-Muster verwendet. Es gewährleistet die Ortstransparenz innerhalb eines verteilten Systems.

Interactive Systems („Interaktive Systeme“) Interaktive Systeme unterstützen die Strukturierung der Mensch-Computer-Interaktion. Aus dieser Kategorie wird das Model-View-Controller-Muster genutzt.

4.2. Schichten

Die Schichtenbildung hat sich in der Vergangenheit als ein geeignetes Mittel zur Strukturierung bewiesen. Eine Schicht stellt eine Außensicht zur Verfügung und kapselt somit die Implementierung gegenüber der Außenwelt. Komponenten innerhalb der Schichten weisen einen ähnlichen Abstraktionsgrad auf (vgl. Starke (2005), Seite 122-124).

Die wichtigste strukturelle Charakteristik des Schichtenmusters ist, dass Schicht N nur die Dienste von Schicht N-1 in Anspruch nimmt. Jede Schicht schützt somit alle unter ihr liegenden Schichten vor einem direkten Zugriff und minimiert somit Abhängigkeiten (vgl Buschmann u. a. (2001), S. 34).

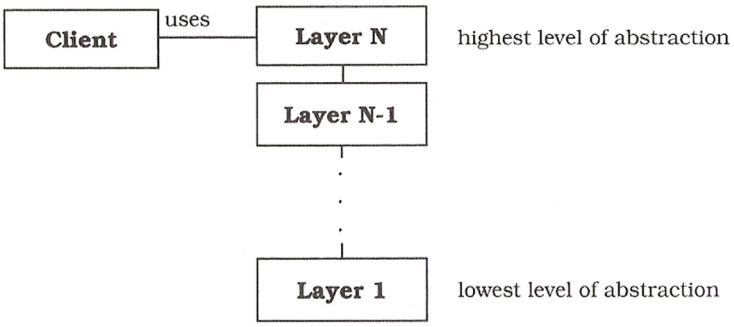

Abbildung 4.1.: Schichtendarstellung, (Quelle: Buschmann u. a. (2001))

Betrachtet man die Schichten etwas genauer, zeigt sich, dass sie komplexe Einheiten darstellen, die aus verschiedenen Komponenten bestehen. In der folgenden Abbildung 4.2 besteht jede Schicht aus 3 Komponenten. In der mittleren Schicht interagieren zwei Komponenten miteinander. Die Komponenten rufen die Dienste der anderen direkt auf, wie z.B. bei Component_2.1 und Component_1.1 (vgl Buschmann u. a. (2001), S. 35).

Nach (Starke (2005), S.122-123) bieten Schichten folgende Vorteile:

- Sie sind im Betrieb und bei der Erstellung unabhängig voneinander.

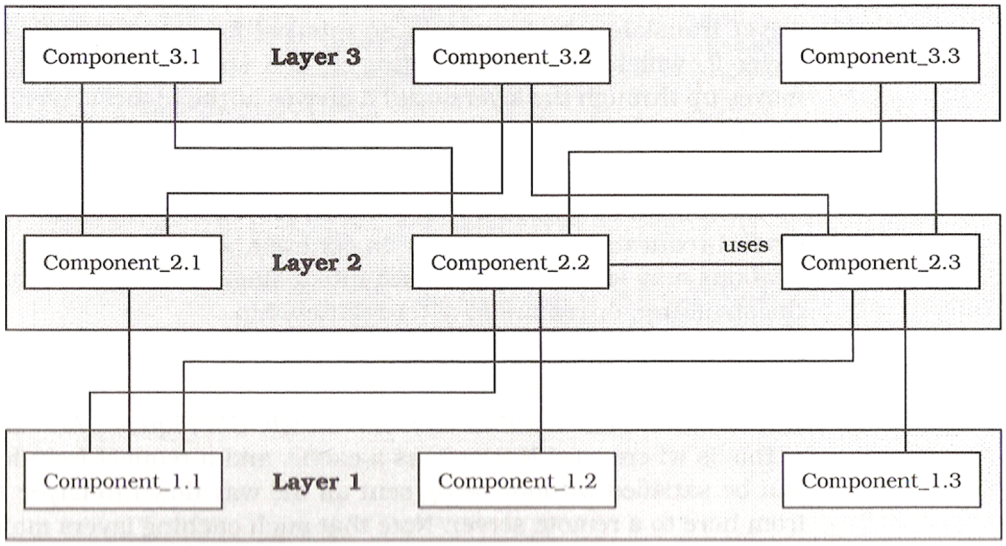

Abbildung 4.2.: tiefer Schichtendarstellung, (Quelle: Buschmann u. a. (2001))

- Es ist möglich, die Implementierung einer Schicht unter der Voraussetzung auszutauschen, dass die neue Implementierung die gleichen Dienste zur Verfügung stellt.
- Die Bildung von Schichten minimiert Abhängigkeiten von Komponenten.
- leicht verständliches Strukturkonzept.

Schichten bieten nach (Starke (2005), S.123) folgende Nachteile:

- Performance kann negativ beeinflusst werden, da Anfragen unter Umständen über mehrere Schichten gereicht werden müssen. Als eine Lösung zu diesem Problem wird eine so genannte Layer-Bridge eingeführt, welche die dazwischenliegenden Schichten überbrückt. Dabei wird der Performancevorteil durch eine zusätzliche Abhängigkeit erkauft.
- Manche Änderungen innerhalb eines Systems erzeugen innerhalb des Schichtenmusters einen Welleneffekt. Wenn nun ein neues Datenfeld in das System eingeführt wird und dieses zum einen angezeigt und auch gespeichert werden soll, zieht dies Änderungen an allen Schichten nach sich.

4.2.1. 3-Schichten-Architektur

Für Standard-Software wird die aus der Praxis gängige 3-Schichten-Architektur verwendet:

1. Präsentation
2. Fachdomäne: Ermöglicht Entwicklern, sich auf die fachlichen Aspekte zu konzentrieren
3. Infrastruktur: Kapselt technische Infrastruktur gegenüber der Fachdomäne. Wird typischerweise weiter untergliedert in:
   - Persistenz
   - Integration von Fremdsystemen
   - Sicherheit
   - Kommunikation und Verteilung

4.2.2. Anwendung der 3-Schichten-Architektur

Besonders interessant ist hier die Aufteilung der drei Schichten bei einer Client-Server-Anwendung. Sie wird in dem folgenden Kapitel 4.3 vorgestellt. Die Verteilung der drei Schichten auf die Client- und Serveranwendung wird in Abschnitt 4.3.1 beschrieben.

4.3. Client-Server-Anwendungen

Im Einführungskapitel (1.2) wird vorgeschrieben, dass es sich bei diesem verteilten System um eine Client-Server-Anwendung handeln soll.

Wie schon in Abschnitt 4.2.1 erwähnt, hat es sich in der Praxis als vorteilhaft erwiesen, eine Anwendung in drei Schichten zu unterteilen. Bei einer Client-Server-Anwendung lassen sich diese drei Schichten auf fünf verschiedene Arten sinnvoll trennen:

- Bei Abbildung 4.3(a) verbleibt nur der Teil des User-Interfaces auf dem Client, welcher die Eingabe verarbeitet. Die Steuerung wird vom Server aus kontrolliert.
- Abbildung 4.3(b) belässt das komplette User-Interface auf der Clientseite, wobei die Anwendung in eine grafische und eine verarbeitende Komponente getrennt werden kann.
- In Abbildung 4.3(c) enthält der Client einen Teil der Anwendung.
- Bei Abbildung 4.3(d) wird die gesamte Anwendung clientseitig ausgeführt. Die persistenten Daten werden auf dem Server gehalten.
- Zusätzlich wird bei Abbildung 4.3(e) ein Teil der Daten auf dem Client verwaltet.

Abbildungen 4.3(a)-(c) werden auch thin-Clients genannt, da die rechenintensiven Komponenten auf dem Server gelagert sind. Hingegen werden 4.3(d)-(e) fat clients genannt (vgl. Tanenbaum (2007) S. 38 - S.43).

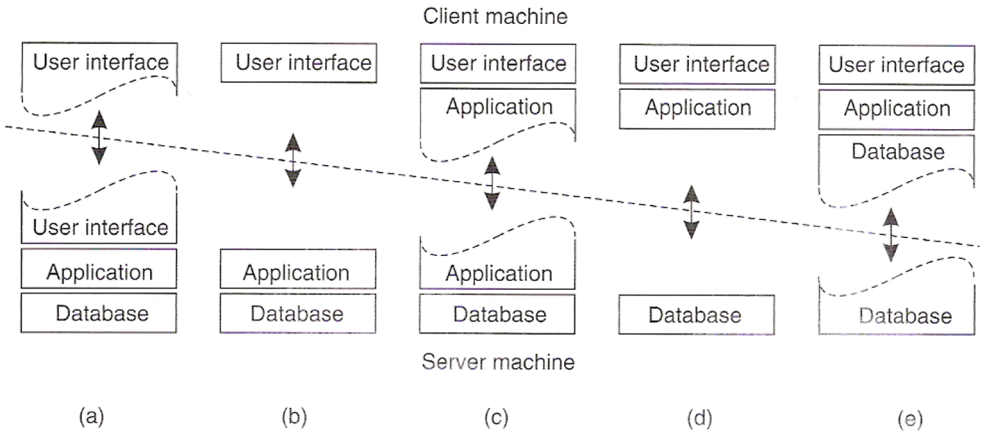

Abbildung 4.3.: Mögliche Schichtenaufteilung bei einer Client-Server-Anwendung, Quelle Tanenbaum (2007)

4.3.1. Verteilung der 3-Schichten innerhalb der Client-Server-Anwendung

In Kapitel 4.8.1 wird beschrieben, wo die in dieser Arbeit vorgestellte Anwendung positioniert werden kann.

4.4. Kommunikation zwischen Client und Server

Die Komponenten eines verteilten Systems müssen miteinander kommunizieren können. Es wird anschließend das Broker-Architekturmuster und dessen Anwendung innerhalb des Designs vorgestellt. Nachfolgend wird kurz auf Webservices und XML sowie Message Passing eingegangen, da beide Themen eine Relevanz im Bereich der verteilten Systeme darstellen.

4.4.1. Broker-Architekturmuster

Das Architekturmuster Broker wird dazu verwendet, um verteilte Softwaresysteme mit entkoppelten Komponenten zu strukturieren, welche über entfernte Aufrufe miteinander interagieren.

Wenn bei einem verteilten System die beteiligten Komponenten den Ablauf der Kommunikation selbst übernehmen, sind stärkere Abhängigkeiten und auch Einschränkungen die Folge. Das System würde z.B. abhängig vom Ort des Servers sein. Ein von dem System unabhängiger Mechanismus zur Verwaltung von Komponenten, ist zudem auch nötig. Innerhalb einer Anwendung sollte ein Objekt keine zusätzlichen Schnittstellen anbieten, welche sich nicht mit der eigenen Funktionalität befassen. Wie das Objekt implementiert wird und wo es zur Laufzeit existiert, sollte für den Entwickler keine Rolle spielen.

Dies kann über eine sogenannte Brokerkomponente erreicht werden, welche die Entkopplung von Clients und Servern unterstützt. Dabei registrieren sich die Server bei dem Broker, um deren Dienste Clients über Schnittstellen verfügbar zu machen. Clients hingegen können diese Schnittstellen nutzen, indem sie Anfragen an den Broker senden. Der Broker ist dafür zuständig, die Kommunikation zu organisieren, dies beinhaltet unter anderem das Weiterleiten von Anfragen an den richtigen Server sowie die Übertragung von Ergebnissen und Fehlern zurück an den Client. Über das Broker-Muster wird die Verteilung von Objekten für den Entwickler transparent. Dies wird erreicht, indem ein Objektmodell eingeführt wird, welches die Verteilung kapselt und somit versteckt.

Es lassen sich die folgenden Komponenten innerhalb des Broker-Musters identifizieren (siehe Abbildung 4.4), sie werden in der folgenden Übersicht kurz beschrieben und anschließend einzeln betrachtet.

Clients Nutzen Serverdienste über Broker.

Servers Bieten Serverdienste über Broker an.

Brokers Regeln Kommunikation zwischen Client, Server und entfernten Brokern.

Bridges kapseln systemspezifische Details bei Broker-Broker-Kommunikation.

clientseitige Proxies stellen Anfragen und verarbeiten Ergebnisse.

serverseitige Proxies verarbeiten Anfragen und stellen Ergebnisse.

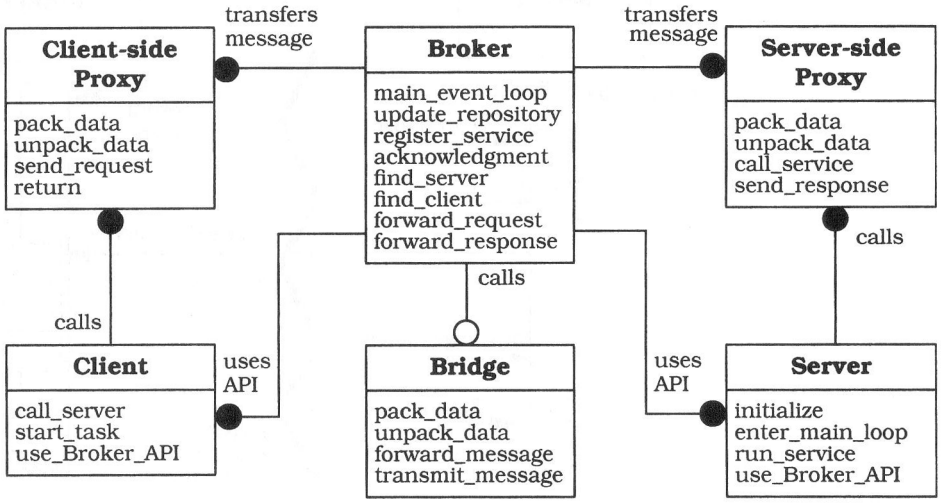

Abbildung 4.4.: Broker, Quelle: Buschmann u. a. (2001) S.107

Clients

Ein Client ist eine Anwendung, welche die Dienste eines Servers in Anspruch nimmt. Dabei stellt der Client die Anfrage an den Broker. Nachdem die Anfrage ausgeführt wurde, erhält der Client über den Broker die Antwort oder etwaige Fehler.

Servers

Ein Server bietet seine Dienste über Schnittstellen an. Diese Schnittstellen werden dabei nach einer interface definition language (IDL) oder einem binären Standard definiert, um sie verfügbar zu machen.

Brokers

Der Broker ist zuständig für die Übertragung der Anfragen von Clients an den Server sowie die daraus resultierenden Antworten. Dabei muss ein Broker die Empfänger der Antworten von vorher gestellten Anfragen zuordnen können. Zusätzlich bietet der Broker Schnittstellen an, damit Server sich beim Broker registrieren und Clients Servermethoden ausführen können.

Wird eine Anfrage über einen lokalen Broker an einen Server gestellt, wird diese wie bereits beschrieben verarbeitet. Falls der Server allerdings nur über einen entfernen Broker

zugänglich ist, muss der lokale Broker eine Verbindung zu dem entfernten Broker aufbauen und die Anfrage weiterleiten. Broker können also miteinander kommunizieren, wobei hier vernachlässigt werden soll, auf welche Weise diese Kommunikation im Einzelnen realisiert wird.

Bridges

Die Bridges sind Komponenten, welche die Kommunikation zwischen zwei Brokern in einem heterogenem Netzwerk transparent gestalten. Eine Bridge bildet dabei eine Schicht, die alle systemspezifischen Details kapselt.

clientseitige Proxies

Ein clientseitiger Proxy stellt die Schicht zwischen Client und Broker dar, über welche die Verteilungstransparenz erreicht wird. Die gekapselten Dienste umfassen die nötigen Mechanismen, um zwischen dem Client- und Brokerprozess zu kommunizieren, das Verpacken von Parametern und Antworten, sowie die Verwaltung des dafür nötigen Speichers.

serverseitige Proxies

Die serverseitigen Proxies verhalten sich analog zu den clienseitigen, wobei sie Anfragen empfangen, diese verarbeiten und den entsprechenden Dienst aufrufen. Zudem muss das Ergebnis wieder verarbeitet und zurückgeschickt werden.

(vgl. Buschmann u. a. (2001) S.99 - S.105)

Anwendung des Broker-Musters

Das Broker-Muster beschreibt einen allgemeinen Ansatz; im Bereich der Objektorientierung werden Object-Broker eingesetzt, um das Broker-Muster umzusetzen. Es ermöglicht innerhalb des Designs, alle Objekte unabhängig von deren Ort auf dieselbe Art zu behandeln. Im Java-Bereich wäre hier z.B. der Einsatz von RMI[1] oder auch CORBA[2] möglich. Beide Varianten haben sich in der Praxis bewährt. In Kapitel 4.8.2 wird beschrieben, wie der Object-Broker innerhalb der Anwendung für die Kommunikation zwischen Client und Server eingesetzt wird.

[1] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[2] http://www.corba.org/

4.4.2. Webservices und XML

Das im World Wide Web vorherrschende Kommunikationsmodell ist das schon im Abschnitt 2.1.1 vorgestellte request-reply-Verhalten. Dabei nimmt ein Client den Dienst eines Servers in Anspruch, indem er eine Anfrage an den Server stellt und wartet, bis dieser antwortet. Diese Art der Kommunikation wird auch synchron genannt. Diese Form der Interaktion ist angemessen, solange nicht dauerhaft eine bestimmte Information abgefragt werden muss. Eine dauerhafte Abfrage von Änderungen ist oft unnötig, da gar keine Änderungen vorliegen, und verbraucht zusätzliche Ressourcen (vgl Dustdar S. (2003),S. 229).

Der Webservice stellt eine Weiterentwicklung der klassischen Vorgehensweise dar. Es werden dabei Dienste eines Servers über den Webservice im Internet „nutzbar“ gemacht. Dabei wartet der Client nach dem Stellen einer Anfrage nicht auf die Antwort des Servers, sondern bestimmt eine Methode, welche für die Verarbeitung der Serverantwort zuständig ist, sobald diese eintrifft. Diese Weise der Kommunikation heißt asynchron. Webservices sollen beschrieben, veröffentlicht, gefunden und dynamisch aufgerufen werden können. Um Daten auszutauschen, werden XML-basierte Standards verwendet (vgl Dustdar S. (2003),S. 113).

Da es sich bei der asynchronen Kommunikation trotzdem um eine Verhaltensweise handelt, bei dem der Client immer zuerst eine Anfrage stellen muss, bevor der Server eine Antwort liefern kann, ist sie für die hiesigen Anforderungen nicht geeignet und wird nicht weiter beschrieben.

4.4.3. Message Passing

Der Begriff des Message Passing stammt aus dem Bereich der parallelen Verarbeitung, wobei ein System so in Komponenten aufgeteilt wird, dass es gleichzeitig ausgeführt werden kann, um bei Berechnungen eine bessere Auslastung des Gesamtsystems zu erreichen (vgl. Neary und Cappello (2002)).

Zusätzlich beschreibt es eine fundamentale Sichtweise: Objekte müssen auf irgendeine Weise miteinander kommunizieren können. Die Kommunikation erlaubt dabei Objekten, die Dienste eines anderen Objekts zu nutzen, zu informieren und sich zu synchronisieren. Dieser Kommunikationsmechanismus heißt Message Passing. Die Grundidee besteht darin, dass Objekte Nachrichten schicken und empfangen können, um Informationen bereitzustellen oder entgegenzunehmen. Dabei ist es wünschenswert, dass der Nachrichtenmechanismus die lose Kopplung der verschiedenen Objekte nicht einschränkt. Im Bereich des Message Passing lassen sich zwei grundsätzliche Kommunikationsansätze unterscheiden. Die erste stellt dabei die Kommunikation über einen zentralen Datenspeicher und die zweite den

Austausch von Nachrichten über einen Kommunikationsweg dar. Da im Rahmen der Objektorientierung das Austauschen von Nachrichten ein zentrales Konzept darstellt, ist die zweite Variante angebrachter, um zwei nebenläufigen Objekten die Kommunikation zu ermöglichen (vgl. Putzke-Röming u. a. (1998)).

4.5. Graphische Benutzerschnittstelle des Clients

Die graphische Benutzerschnittstelle wird anhand des Model-View-Controller-Architekturmusters (MVC) umgesetzt. Das Ziel ist die Entkopplung von der Darstellung (View), der Datenhaltung (Model) und der Ablaufsteuerung (Controller). Da es sich bei der in dieser Arbeit vorgestellten Anwendung um eine verteilte Anwendung handelt, kommt es oft zu Änderungen von außen (z.B. wenn durch andere Benutzer die ein Dokument bearbeiten) und zu möglichen Inkonsistenzen zwischen Darstellung und Datenhaltung. Diese können durch die Verwendung des MVC vermieden werden. Viele GUI-Frameworks basieren auf dem MVC-Muster, im Falle von Java würde dies z.B. auf die Swing-Bibliothek zutreffen (vgl. Starke (2005), S. 220 - S.221). Abbildung 4.5 verdeutlicht den Klassenaufbau des MVC-Modells.

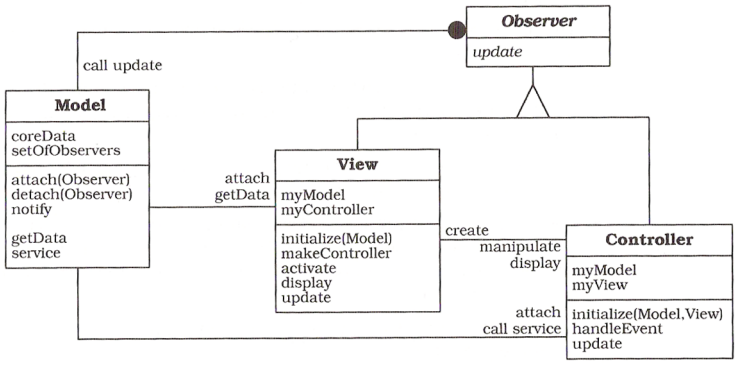

Abbildung 4.5.: MVC-Aufbau, (Quelle: Buschmann u. a. (2001) S.129)

Nach (Buschmann u. a. (2001) S. 124) gibt das MVC-Modell keine Auskunft darüber, wie eine effiziente Kommunikation zwischen den einzelnen Teilen gewährleistet werden kann. Dies lässt sich auf die einschränkende Interaktion zwischen Observable und Observer über die update-Methode zurückführen (Buschmann u. a. (2001) S. 143). Ein weiters Problem besteht in Programmiersprachen, welche keine Mehrfachvererbung unterstützen. Da die meisten Modelle selbst schon von einer anderen Klasse erben, stellt dies ein grundlegendes Problem dar. Innerhalb von objektorientierten Sprachen wurde deshalb ein flexiblerer Mechanismus des Beobachter-Musters eingeführt, die Kombination von Event und EventListener. Der Vorteil liegt darin, dass Event und Listener nicht gekoppelt sind. Bei mehreren Listenern und

Events würden die verschiedenen Listener eigene Implementierungen der Methoden für jedes Event besitzen und nicht nur eine große update-Routine, welche alle Änderungsfälle abdecken muss. Der Nachteil ist, dass ein Event an jeden Listener manuell gesendet werden muss, bei einem Observer wird dies hingegen automatisch über die Oberklasse abgewickelt. Da dies als ein hinnehmbarer Nachteil angesehen werden kann, wird in diesem Design ausschließlich auf die Implementierung von Event und EventListener gesetzt.

Bei objektorientierten Benutzerschnittstellen wird eine Oberfläche schnell sehr komplex und erfordert eine Erweiterung des MVC-Muster. Dabei setzt sich meistens eine View aus mehreren Subkomponenten zusammen. Diese sind für die Darstellung von verschiedenen Ausschnitten des Modells verantwortlich und sind demzufolge auch mit dem Ablauf verknüpft. Dabei wird diese Hierarchie der GUI-Komponenten anhand von so genannten MVC-Tripeln realisiert, wie in Abbildung 4.6 dargestellt. Dabei wird die Vater-Kind-Beziehung der graphischen Komponenten auf die Controller-Hierarchie übertragen. Es entsteht somit eine Verarbeitungskette, wobei der Controller des Hauptfensters die Wurzel darstellt, und die Anfragen bearbeitet, die keines der Kindelemente bearbeitet hat. Innerhalb eines Tripels erfolgt die Kommunikation nach dem MVC-Modell. (vgl. Starke (2005) S.224-225)

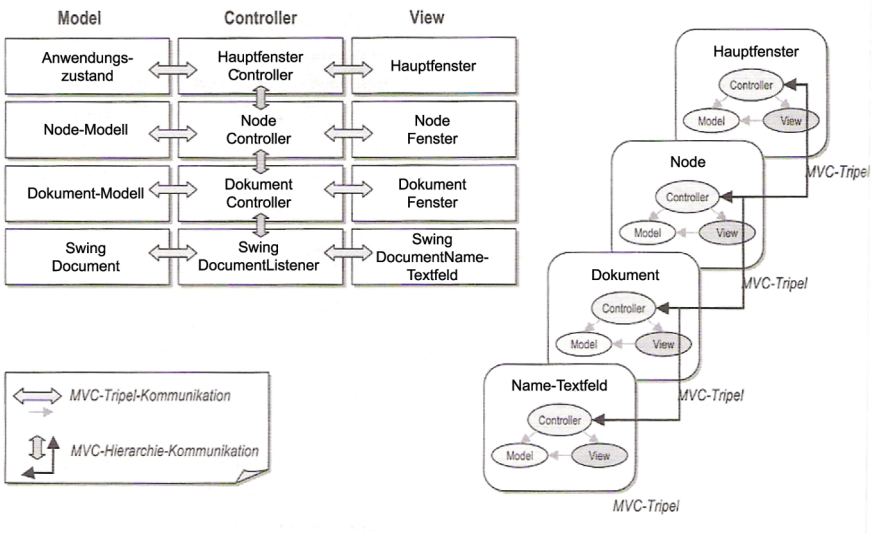

Abbildung 4.6.: MVC-Hierarchie, (vgl. Starke (2005) S.224)

4.5.1. Anwendung des MVC-Musters

Das MVC-Muster wird für den Aufbau der graphischen Benutzerschnittstelle des Clients genutzt, diese wird in Kapitel 4.8.3 näher beschrieben.

Das MVC-Muster wird für den Aufbau der graphischen Benutzerschnittstelle des Clients genutzt, diese wird in Kapitel 4.8.3 näher beschrieben.

4.6. Entwurfsmuster

In diesem Abschnitt sollen die Entwurfsmuster Fassade und Beobachter vorgestellt werden, welche in dem Design Anwendung finden.

4.6.1. Fassade („Facade“)

Ein System wird üblicherweise in Subsysteme unterteilt, um seine Komplexität zu reduzieren. Die Minimierung von Kommunikation sowie Abhängigkeiten zwischen Subsystemen gilt als ein Entwurfsziel des Design-Prozesses. Der Zweck der Fassade liegt darin, dass über sie eine einheitliche Schnittstelle zu einer Menge von Subsystemen angeboten wird. Die Nutzung wird dabei vereinfacht, da die Fassadenklasse eine abstrakte Schnittstelle definiert (Siehe Abbildung 4.7) (vgl. Gamma u. a. (2001) S.212 - S.222).

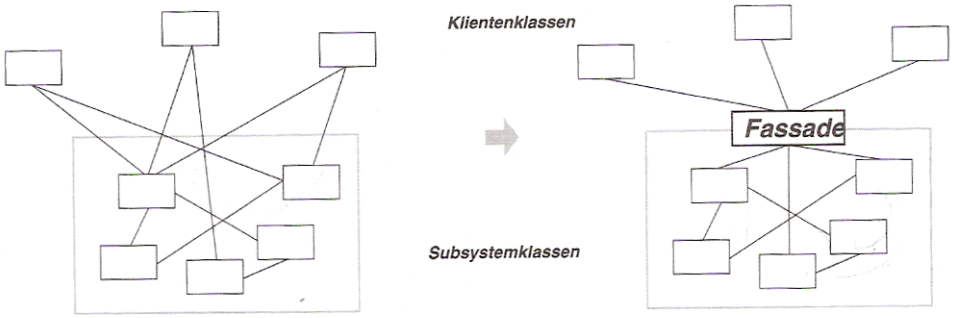

Abbildung 4.7.: Fassade, (Quelle Gamma u. a. (2001), S. 212)

Anwendung

In dem vorliegenden Design wird das Fassadenmuster dazu genutzt, um die Subsysteme in Schichten aufzuteilen. Die Fassade dient dabei als Eintrittspunkt zu der jeweiligen Subsystemschicht. Bei Abhängigkeiten zwischen Subsystemen werden diese verringert, da ausschließlich über ihre Fassade kommuniziert wird. Die Fassade kommt unter anderem in Abschnitt 4.8.5 zur Anwendung.

4.6.2. Beobachter („Observer“)

Durch die Teilung eines Systems in interagierende Komponenten tritt oft der Fall auf, dass eine Änderung eines Objektes an die mit ihm in Beziehung stehenden Objekte propagiert werden muss um die Konsistenz zu wahren. Das Beobachter-Muster sorgt dafür, dass die Änderung eines Zustandes eines Objektes dazu führt, alle abhängigen Objekte zu benachrichtigen und automatisch zu aktualisieren. Abbildung 4.8 zeigt die Konstruktion des Beobachtermusters.

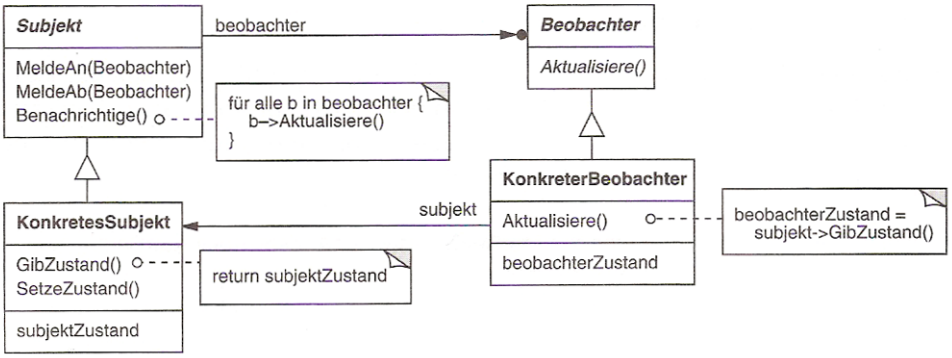

Abbildung 4.8.: Beobachter-Muster, (Quelle: Gamma u. a. (2001), S. 289)

Anwendung

In dem vorliegenden Design wird die graphische Benutzerschnittstelle anhand des MVC- (siehe Abschnitt 4.5) unter Verwendung des Beobachter-Musters, umgesetzt. Weiterhin wird das Beobachtermuster in dem Design explizit in Abschnitt 4.8.14 angewendet.

4.7. Server

Im Kapitel Server sollen die zwei Schichten, Geschäftslogik und Persistenz, sowie die in den Schichten enthaltenen Komponenten erläutert werden. Sie werden nur hinsichtlich ihrer Dienstleistungen beschrieben.

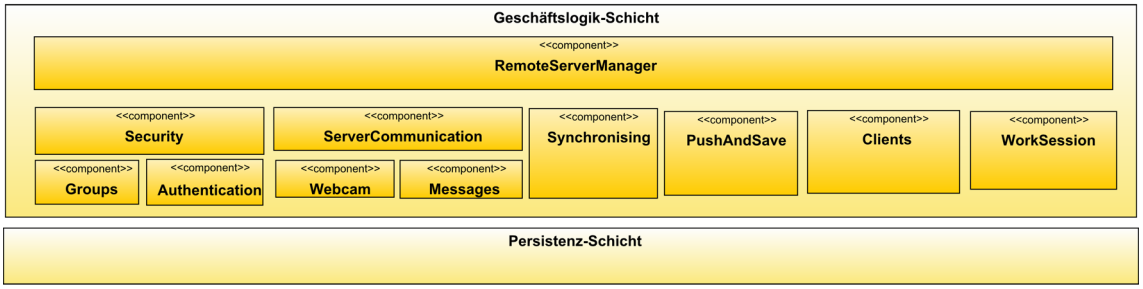

Abbildung 4.9.: Server-Schichtung und Komponentenaufbau

4.7.1. Geschäftslogik-Schicht

Der Server ist zuständig für Authentifizierung, Gruppenverwaltung, Kommunikation, Synchronisierung, Verwaltung von Arbeitssitzungen sowie das Sequentialisieren, Weiterleiten und Speichern von Datenänderungen durch die Clients.

Security

Die Security-Komponente enthält zwei Subkomponenten, `Groups` und `Authentication`. `Authentication` ist für die Anmeldung und die ständige Authentifizierung der Benutzer verantwortlich. `Groups` verwaltet die Gruppen innerhalb des Systems.

ServerCommunication

`ServerCommunication` beinhaltet die Subkomponenten `Webcam` und `Messages`. `Messages` ist für die Verteilung von Nachrichten an Benutzer zuständig. `Webcam` übernimmt die Zentralisierung von Videobroadcasts eines Benutzers, um das Video an weitere Benutzer zu streamen, welche an derselben Arbeitssitzung beteiligt sind.

Synchronising

Diese Komponente bietet die Möglichkeit, ein Dokument oder ein Knoten zu synchronisieren. Die Synchronisierung wird dabei über das Versions-Attribut der zu synchronisierenden Elemente abgehandelt.

PushAndSave

Die `PushAndSave`-Komponente nimmt alle Änderungen, des Systems entgegen, auch Änderungen welche innerhalb einer Arbeitssitzung entstehen. Permanente Änderungen werden gespeichert und an alle Clients verteilt, temporäre Änderungen wie z.B. bei einer Arbeitssitzung werden erst nach der Beendigung gespeichert und verteilt.

Clients

Verwaltet alle aktiv mit dem Server verbundenen Clients. Über diese Komponente kann mit den Clients kommuniziert werden.

Worksession

`Worksession` verwaltet die Arbeitssitzungen, in welchen die Benutzer Elemente gleichzeitig bearbeiten können.

4.7.2. Persistenz-Schicht

Die Persistenzschicht ist für die dauerhafte Speicherung der Objekte zuständig. Sie kapselt die technische Funktionalität zur Speicherung der Geschäfts-Daten. Es kann dabei auf ein Persistenz-Framework wie z.B. Hibernate zurückgegriffen werden. Für dieses Design soll die Annahme reichen, dass Objekte über den Server persistiert werden können.

4.8. Client

Im Kapitel Client soll die Client-Server-Anwendung klassifiziert und danach die Kommunikation zwischen Client uns Server geschildert werden. Anschließend sollen die drei Schichten, Präsentation, Kommunikation und Geschäftslogik und die dazugehörigen Komponenten beschrieben werden.

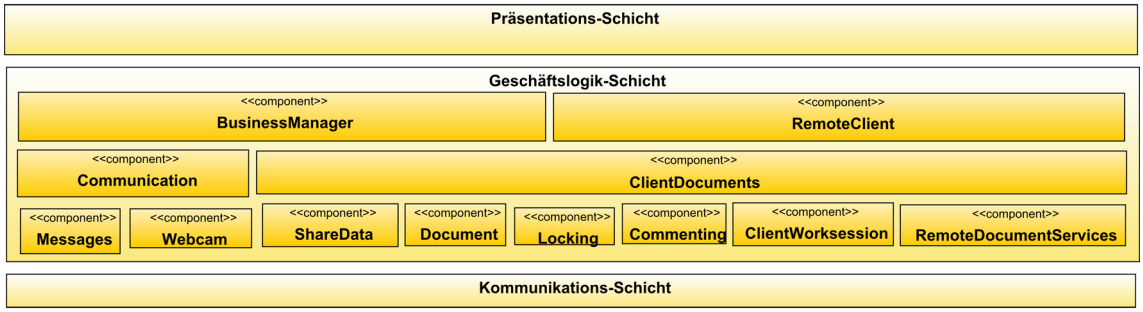

Abbildung 4.10.: Client-Schichtung und Komponentenaufbau

4.8.1. Einordnung der Client-Server-Anwendung

Die Anwendung kann in der Abbildung 4.3 auf Seite 42, aus dem Kapitel „Client-Server-Anwendungen“ unter dem Modell „c“ positioniert werden, da auf dem Server, sowie auf dem Client, ein Teil der Anwendung implementiert wird. Dabei läuft auf dem Client der Kern der Anwendung, der Dokumenten- und Texteditor. Der Server ist zum einen für zentrale Aufgaben wie Rechte- und Arbeitssitzungsverwaltung und zum anderen für die Sequentialisierung, Verteilung und Speicherung von Daten zuständig. Die Abbildung 4.10 zeigt die Client- und Abbildung 4.9 die Serverschichtung und beide verdeutlichen so den Schnitt innerhalb der Anwendung.

4.8.2. Kommunikation zwischen Client und Server

Die serverseitige Komponente RemoteServerManager fungiert dabei im Sinne des Broker-Musters als Server, wobei die Kommunikationsschicht des Clients als Client zu verstehen ist. Da eine bidirektionale Kommunikation zwischen Client und Server nötig ist, nimmt die RemoteClientkomponente des Clients innerhalb des Broker-Musters die Rolle des Servers ein und die Clientskomponente des Servers ist als Client zu verstehen.

4.8.3. Präsentations-Schicht

Die graphische Benutzerschnittstelle soll anhand des in Kapitel 4.5 vorgestellten MVC-Architekturmusters realisiert werden. Als gängige Praxis hat es sich erwiesen, das Architekturmuster auf der Basis eines MVC-Frameworks umzusetzen. Eine Ausarbeitung der graphischen Komponenten ist nicht Gegenstand dieser Arbeit, da das Design der Komponenten stark von der verwendeten Programmiersprache und dem spezifischen Framework abhängig

ist. Die Evaluation beschränkt sich dabei im Sinne der graphischen Benutzerschnittstelle auf eine mögliche Implementierung des MVC-Musters.

4.8.4. Kommunikations-Schicht

Diese Schicht kapselt die Details der Kommunikation zwischen Client und Server gegenüber den anderen Schichten.

4.8.5. Geschäftslogik-Schicht

Die Grafik in Abbildung 4.10 stellt die Komponenten der Geschäftslogik vor. Der *Business-Manager* dient als Fassade und fasst die Dienste der darunter liegenden Komponenten zusammen. Die Komponenten *Communication* und *ClientDocuments* sind ebenso Fassaden für die darunter liegenden Komponenten. *RemoteClient* kapselt die Dienste, welche vom Server genutzt werden können.

4.8.6. BusinessManager-Komponente

Die BusinessManager-Komponente dient als Fassade zu den darunterliegenden Komponenten Communication und ClientDocuments, dabei kapselt der BusinessManager allerdings nur die Funktionalitäten, welche der Benutzer über Interaktionen auslösen kann.

4.8.7. RemoteClient-Komponente

Die RemoteClient Komponente dient als Fassade und kapselt nur die Funktionalitäten der darunterliegenden Komponenten, Communication und ClientDocuments, welche über den Server aufgerufen werden können. Weiterhin stellt die RemoteClient-Komponenten die Funktionalität per Reflexion zur Verfügung, welche dazu benötigt wird, um Änderungen, welche durch den Server verteilt wurden, lokal auszuführen.

4.8.8. ClientDocuments-Komponente

Über die Fassade `iClientDocuments` kann auf die darunter liegenden Schnittstellen `iCommentManager`, `iLockingManager`, `iDocumentManager`, `iShareDataManager`, `iClientWorksessionManager`, `iClientWorksessionManager` und `RemoteDocumentServices` zugegriffen werden.

Die Komponenten `CommentManager`, `LockingManager` und `ShareDataManager` sind sehr ähnlich aufgebaut, da sie im Grunde einen vergleichbaren Dienst zur Verfügung stellen. Alle drei Komponenten verwalten zusätzliche Informationen über die Elemente, welche die entsprechenden Schnittstellen implementieren.

Es werden zuerst die verschiedenen Manager-Komponenten und anschließend die Dokumentenstruktur dargestellt.

4.8.9. DocumentManager

Der DocumentManager ist für die Verwaltung, den Aufbau und die Übersetzung von Dokumenten zuständig.

Um Dokumente zu organisieren, kann über `Nodes`(Abbildung 4.11) eine simple Baumstruktur aufgebaut werden. Die `Nodes` ohne Elternnode bilden dabei die Wurzel des Baumes und somit die oberste Ebene der Verzeichnisstruktur. Ein `Node` kann beliebig viele Kindnodes und `Documents` beinhalten. Die Node-Klasse ist generisch und wird auch von dem `MessageManager` (4.8.17) zur Organisation der Nachrichten genutzt. Alle Benutzer, unabhängig davon welcher Gruppe sie angehören, sehen alle `Nodes` und die darin enthaltenen `Documents`. Wird ein `Node` gelöscht, werden alle seine `Documents` und Kindnodes mitgelöscht. Ein Benutzer muss dabei Mitglied in allen Gruppen sein, zu denen die `Documents` gehören.

Weiterhin bietet der `DocumentManager` die Möglichkeit, die Struktur- und Inhalts-Elemente eines Textdokumentes zu erstellen und zu verwalten. Über den `iDocumentRenderer` kann das Textdokument in verschiedene Formate übersetzt werden.

4.8.10. CommentManager

Der CommentManager stellt die Schnittstelle zwischen kommentierbaren Elementen (`Documents` und `TextComponent`) dar. Er ermöglicht für diese Elemente, Kommentare neu zu erstellen, editieren und auf Kommentare zu antworten. Wenn ein Kommentar gelöscht wird, werden alle dazugehörigen Kommentare mitgelöscht.

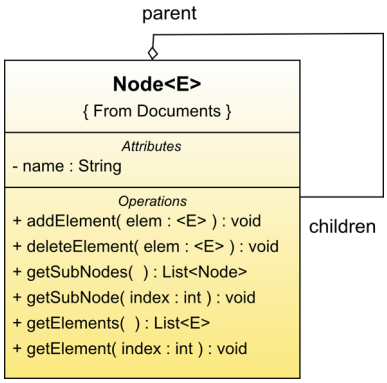

Abbildung 4.11.: Node-Klasse

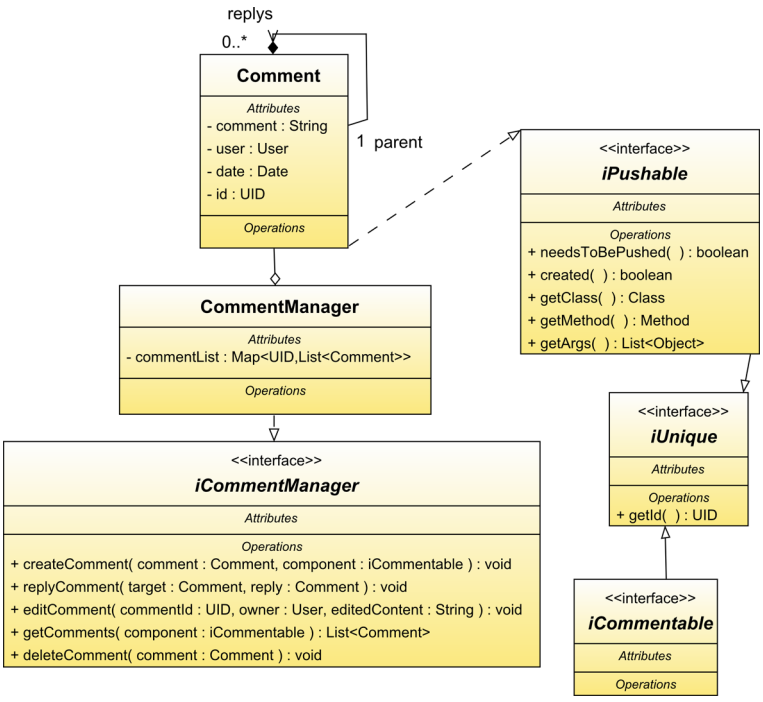

Abbildung 4.12.: CommentManager-Übersicht

4.8.11. LockingManager

Der `LockingManager`(Abbildung 4.13) ist für die Sperrvergaben von `Document` und `TextComponent` zuständig. Dabei wird nur die Möglichkeit zur Bearbeitung gesperrt, nicht aber der Lese-Zugriff. Ein Benutzer kann über die Methode `aquireLock()` eine Sperre auf ein derzeit nicht gesperrtes Element legen. Bei einem `StructureComponent` werden alle Kindelemente mitgesperrt. Sollte bereits ein Kindelement von einem anderen Benutzer gesperrt worden sein, bleibt dieses unberührt. Der Benutzer kann weiterhin das gesperrte Kindelement bearbeiten. Sobald er die Sperre abgibt, fällt das entsprechende Kindelement unter die Sperre des übergeordneten Elternelementes.

Über die Methode `canAccess()` wird bestimmt, ob der anfragende Benutzer das `Document` oder die `TextComponent` editieren darf. Da Verschachtelungen der sperrbaren Elemente möglich sind (wie z.B. bei einem `StructureComponent`), muss überprüft werden, ob eventuell bereits eine Sperre auf einem der Eltern-Objekte gelegt wurde.

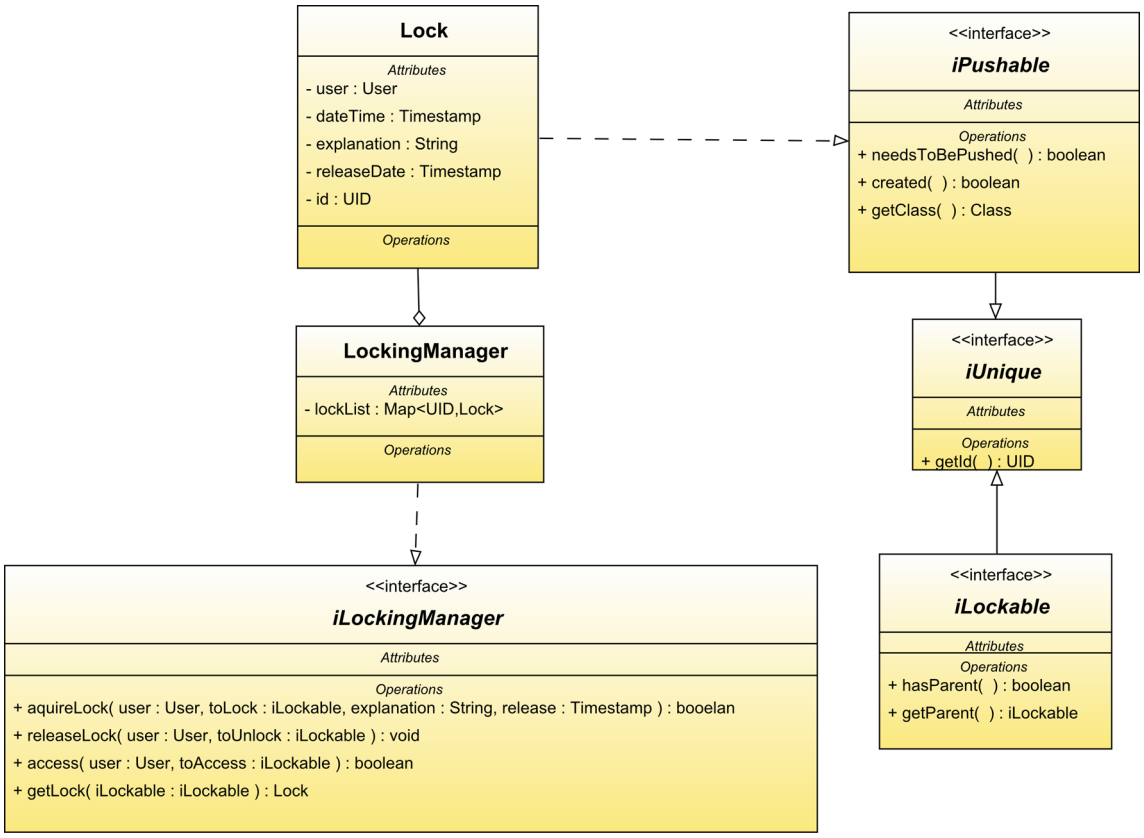

Abbildung 4.13.: LockingManager

4.8.12. ShareDataManager

Der ShareDataManager(Abbildung 4.14) erlaubt es dem Benutzer, über die Methode `uploadData()` Daten hochzuladen. Es können entweder Daten für `Document` und `TextComponent` permanent gespeichert oder für eine laufende Arbeitssitzung zwischenzeitlich gespeichert werden. Über die Methode `saveData()` können die temporären Daten einer Arbeitssitzung persistiert werden.

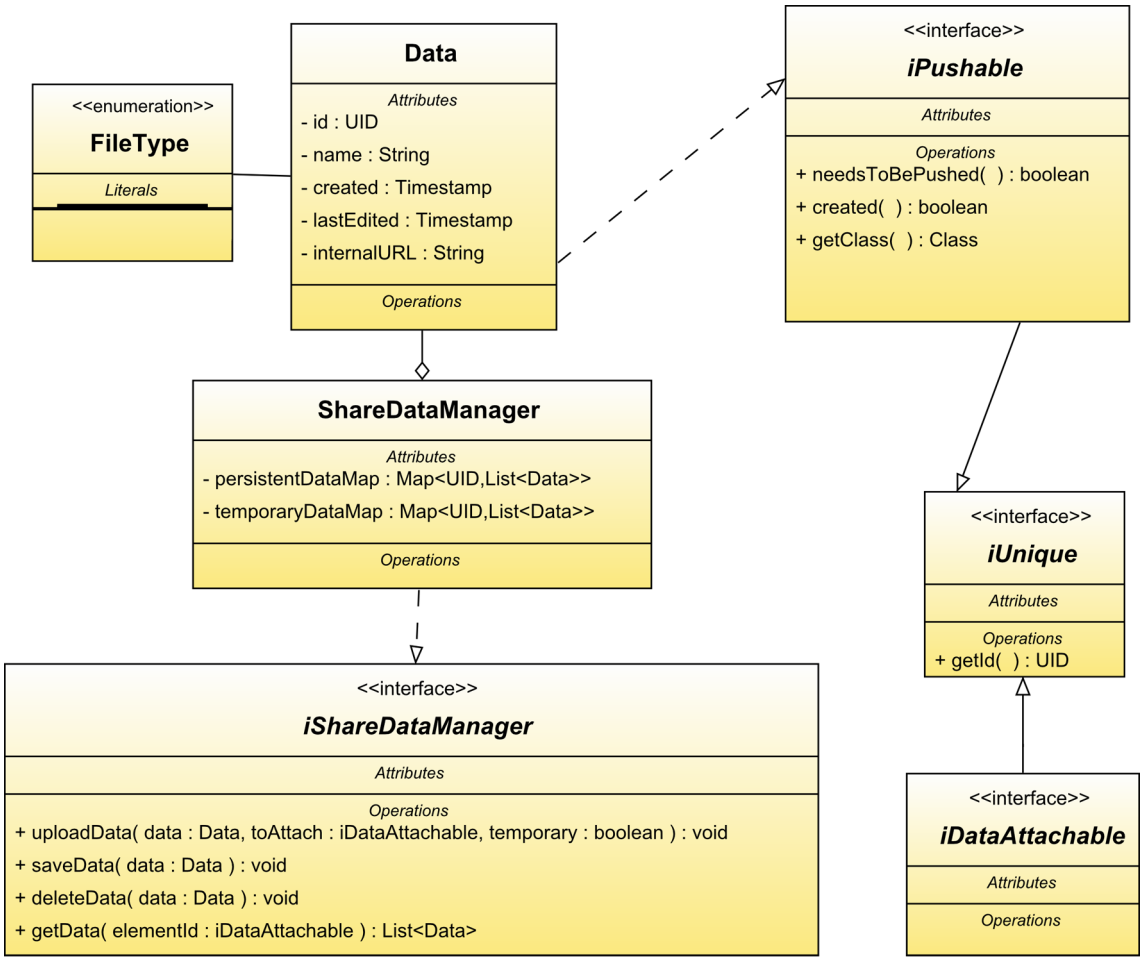

Abbildung 4.14.: ShareDataManager

4.8.13. ClientWorksessionManager

Der `ClientWorksessionManager` stellt Methoden zur Verfügung, um eine synchron editierbare Komponente (die `Content`- und die `StructurComponent`-Klasse) gemeinsam

bearbeiten zu können. Generell ist es möglich, dass in einer Sitzung nicht nur ein Attribut bearbeitet werden kann, dies tritt allerdings in diesem Design nicht auf. Über die Methode `joinSession()` kann einer Arbeitssitzung für eine synchron editierbare Komponente beigetreten werden. Dabei wird automatisch über den Server eine Arbeitssitzung erstellt, falls keine existieren sollte. Es ist dabei nicht möglich, an mehreren Sitzungen gleichzeitig teilzunehmen.

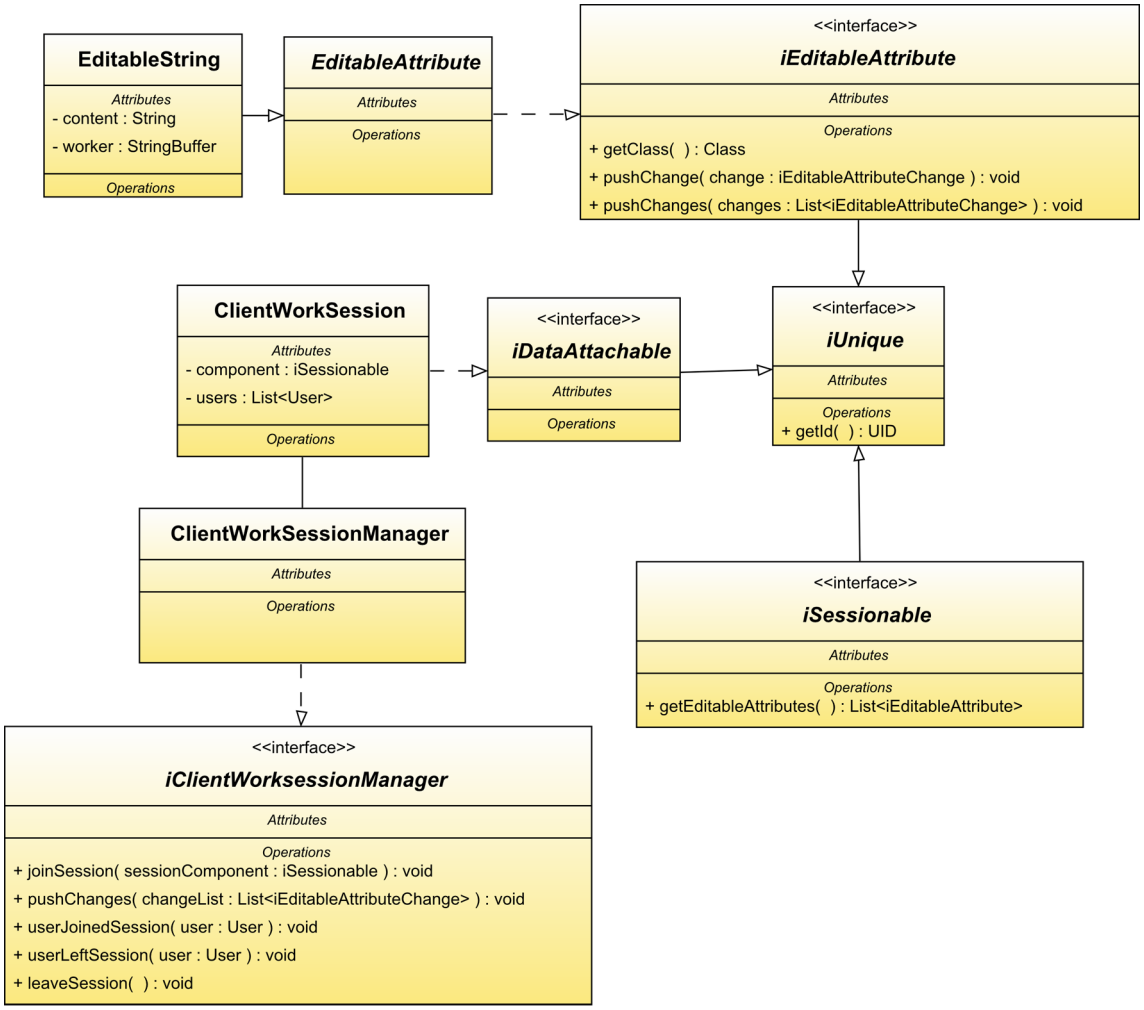

Abbildung 4.15.: ClientWorkSessionManager

Änderungen durch andere Benutzer werden über die Methode `pushChanges()` übertragen. Dabei wird eine Liste mit `iEditableAttributeChanges` als Argument mitgegeben. Diese Änderungen werden über den `worker` aus dem `EditableAttribute` ausgeführt. Dabei arbeit der `worker` direkt auf dem Model des Attributes. Die Eingabe des Benutzers

über die grafische Oberfläche und die Änderungen von anderen Benutzern unterscheiden sich nur in der Quelle der Aktion, funktionieren aber im Grunde genommen gleich.

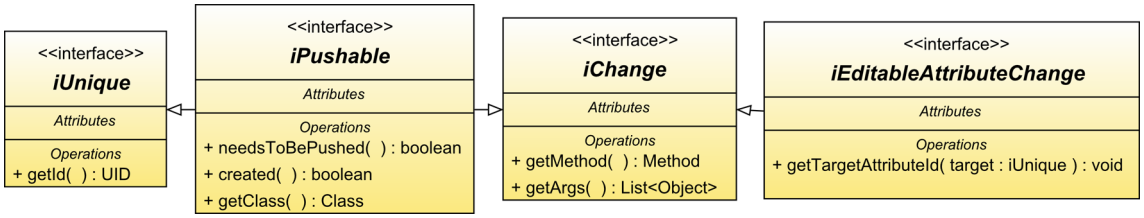

Abbildung 4.16.: Pushable-Schnittstelle und direkt davon abhängige Schnittstellen

4.8.14. RemoteDocumentServices

Diese Komponente bietet zum einen alle serverseitigen Dienste an, auf welche die ClientDokuments-Komponente zugreifen kann, um ein einheitliches Interface zu garantieren und zum anderen wird über RemoteDocumentServices die Synchronisierung der Client- und Serverdaten gewährleistet. Soll eine serverseitige Methode aufgerufen werden, leitet RemoteDocumentServices die Anfrage an die Kommunikationsschicht weiter. Um Änderungen an den lokalen Daten an den Server weiterzuleiten, wird das Beobachter-Muster verwendet. Die Objekte, die zwischen Clients automatisch aktualisiert werden müssen, sind nur lose mit diesem Synchronisierungs-Mechanismus gekoppelt, da Änderungen über Ereignisse an alle Beobachter übermittelt werden. Die RemoteDocumentServices-Komponenten überwacht dabei alle Objekte, bei denen Änderungen an den Server weitergeleitet werden müssen. Da der Begriff der Synchronisierung bereits in einem anderen Kontext verwendet wird, soll diese Art der Synchronisierung als Push-Mechanismus benannt werden.

4.8.15. Dokumentenstruktur

Ein kollaboratives Textdokument gliedert sich in seinen fachlichen Inhalt, die Meta-Daten (Strukturierung und Layout) und kollaborative Informationen wie z.B. Editierrechte, Kommentare etc. . Revout (2008) beschreibt zwei Möglichkeiten, die drei Teile auf eine Dokumentenstruktur abzubilden. Entweder können alle Komponenten in einer Struktur vereint werden, oder die Komponenten werden voneinander getrennt. Eine Trennung macht insofern Sinn, dass ein höherer Grad an Flexibilität hinsichtlich Formatierung und Darstellung erreicht werden kann. Bei einer Trennung können zusätzlich die kollaborativen Informationen leichter vor dem Benutzer verborgen bleiben (vgl. Revout (2008) S.61 - S.62).

Aufgrund der höheren Flexibilität wird eine Trennung der drei Komponenten vorgenommen. Das Dokumentenformat soll dabei unabhängig sein. Dies erfordert eine Übersetzungsinstanz, welche das Dokument in ein entsprechendes Format transformiert.

Klassenaufbau

Die Klasse `TextDocument` besteht aus einem `iDocumentLayout` und mehreren `Textcomponent`-Objekten und stellt ein Schriftdokument dar. Die Klasse `Textcomponent` ist die abstrakte Definition einer Textkomponente. Dabei erben `StructureComponent` und `Content` von `Textcomponent`. `StructureComponent` kann `Content` sowie `StructureComponent`-Objekte enthalten, somit ist es möglich, `TextDocument` gemäß einer Baumstruktur zu gliedern. Diese Baumstruktur spiegelt die logische Struktur des Schriftdokuments wieder. Die Klasse `Content` besitzt nur eine Eigenschaft, `content`, und repräsentiert einen Textabschnitt. Die Eigenschaft `name` der `StructurComponent`-Klasse hält z.B. den Kapitel- oder den Sektionsnamen, der Enum `StructureType` legt den entsprechenden Strukturtyp fest. `StructurComponent` ist als das Strukturelement zu verstehen. Beide Attribute sind vom Typ `EditableString` und können innerhalb einer Arbeitssitzung gleichzeitig mit anderen Benutzern bearbeitet werden. Mehr hierzu in Abschnitt 4.8.13.

Informationen über Zugriffsrechte, Kommentare, Daten und Arbeitssitzungen werden von den jeweiligen Komponenten über Schnittstellen entkoppelt und anderweitig verwaltet. Abbildung 4.17 zeigt den Aufbau der Klassenhierarchie. Die Schnittstellen sind dabei aus Gründen der Übersichtlichkeit nur als Icon sichtbar. Auf ihre Funktionalität wird im Abschnitt 4.8 weiter eingegangen.

Dokumentenformat

Um sich nicht auf ein Format zu beschränken, wurde der Aufbau des Schriftdokuments abstrahiert. Um ein Schriftdokument nun in ein Format zu übersetzen, wird das Schriftdokument an einen „Übersetzer“ gegeben. Für jedes Format muss ein eigener Übersetzer existieren. Abbildung 4.18 zeigt einen `WebRenderer`, der für die Ausgabe im Web als HTML zuständig ist und einen `LatexRenderer`, welcher das Schriftdokument in das Latex-Format übersetzt.

Kollaborative Informationen

Zugriffsrechte innerhalb des Dokumentes Die Zugriffsrechte stellen innerhalb eines kollaborativen Systems ein wichtiges Thema dar. Dabei hat die Zugriffsrechtestrategie auch

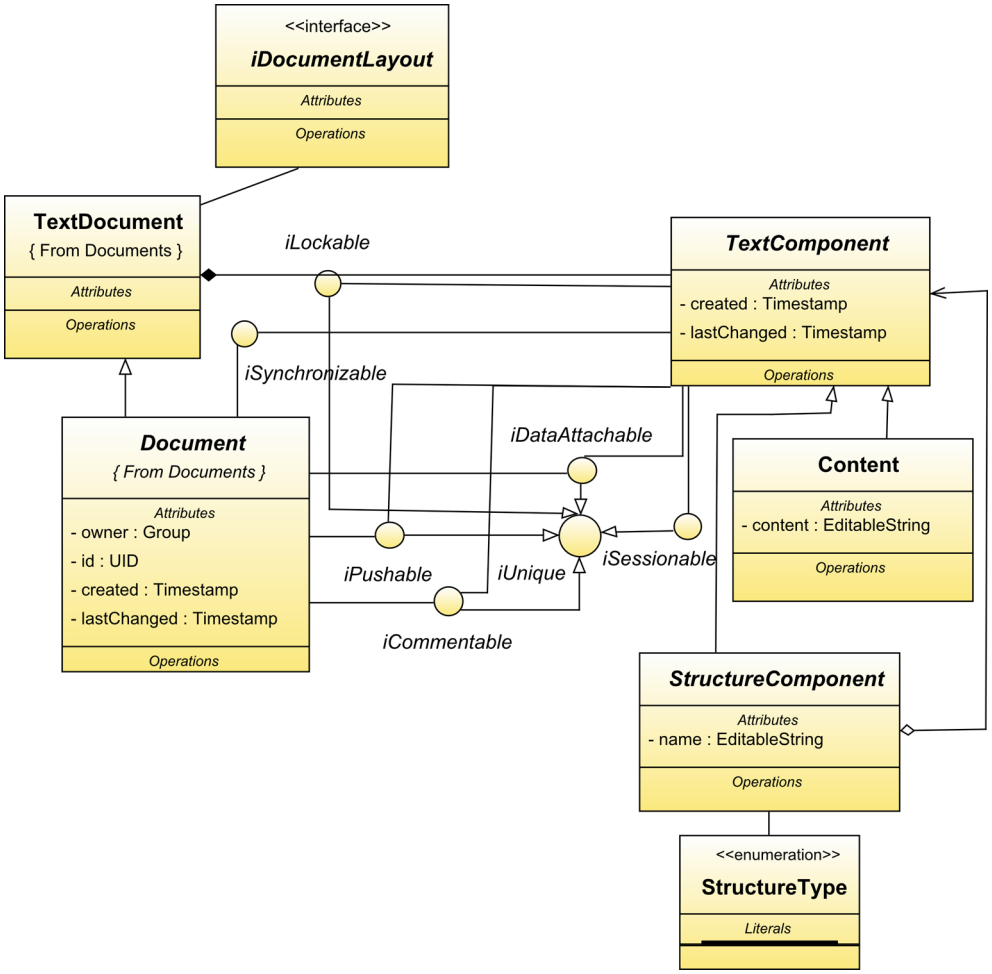

Abbildung 4.17.: Dokumentenstruktur

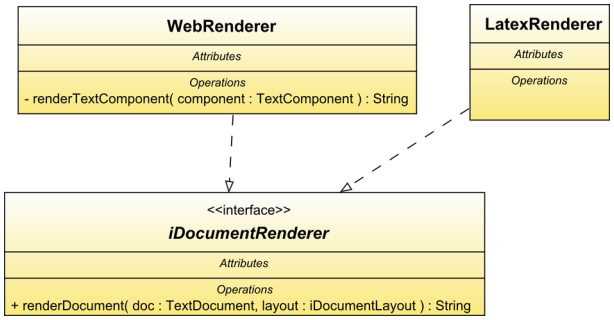

Abbildung 4.18.: Dokumentenübersetzer

Einfluss auf die Dokumentenstruktur. Als Gruppe wird eine Menge von Benutzern definiert, welche ein gemeinsames Ziel verfolgt. Das kollaborative Dokument ist dementsprechend einer Gruppe zugeordnet. Bei der Erstellung eines Dokumentes kann dem Dokument nur eine Gruppe zugeordnet werden, bei welcher der Benutzer, der das Dokument erstellt, ein Teilnehmer ist. Innerhalb der Gruppe, welche ein Dokument bearbeitet, wird von einer gutwilligen Umgebung ausgegangen. Die Benutzer arbeiten zusammen, ohne einander schaden zu wollen. Jeder Benutzer kann alle Teile des Textdokumentes einsehen.

Die Zugriffsrechte beschränken sich hierbei auf das Erstellen, Editieren und Löschen von Textabschnitten. Da es sich bei dieser Anwendung um einen synchronen Texteditor handelt und sämtliche Änderungen über den Server sequentialisiert werden, ist es nicht nötig, Textabschnitte zu sperren, um mögliche Konflikte zu vermeiden. Eine Zugriffsrechte-Sperre kann trotzdem sinnvoll sein, wenn ein Benutzer einen Abschnitt ohne mögliche „Störungen“ von anderen Benutzern bearbeiten möchte. Dies ist z.B. sinnvoll, wenn ein Benutzer einen kompletten Abschnitt in Ruhe umstrukturieren möchte. Eine ausführliche Bearbeitung des Themas Berechtigungsstrukturen ist bei Mund (2006) zu finden.

Wie schon in Abschnitt 4.8.15 angedeutet, wird die Sperrung von Dokumentteilen über eine Schnittstelle geregelt. Dabei können sperrbare Objekte über den LockingManager (Abschnitt 4.8.11) verwaltet werden.

Kommentare und Daten Die Kommentare sowie die verknüpften Daten der Dokumententeile können über die jeweiligen Manager verwaltet werden.

4.8.16. Communication-Komponente

Der CommunicationManager kapselt die Dienste des Message- und des WebcamManagers.

4.8.17. MessageManager

Wie in Abschnitt 4.8.9 bereits beschreiben, wird die Node-Klasse dazu verwendet, um eine Baumstruktur zu erstellen. Die Nachrichten sind den Nodes zugeordnet. Der MessageManager(Abbildung 4.19) erlaubt es, Nachrichten an einzelne oder mehrere Benutzer des Systems zu senden. Um die Organisation von Nachrichten zu vereinfachen, wird durch den MessageSorter ein Ziel-Node für ein- und abgehende Nachrichten definiert. An dieser Stelle könnte ein komplexes Regelwerk die Automatisierung der Nachrichtenverteilung übernehmen wie z.B. bei Microsoft Outlook. In diesem Design wird davon aus Zeitgründen

abgesehen.

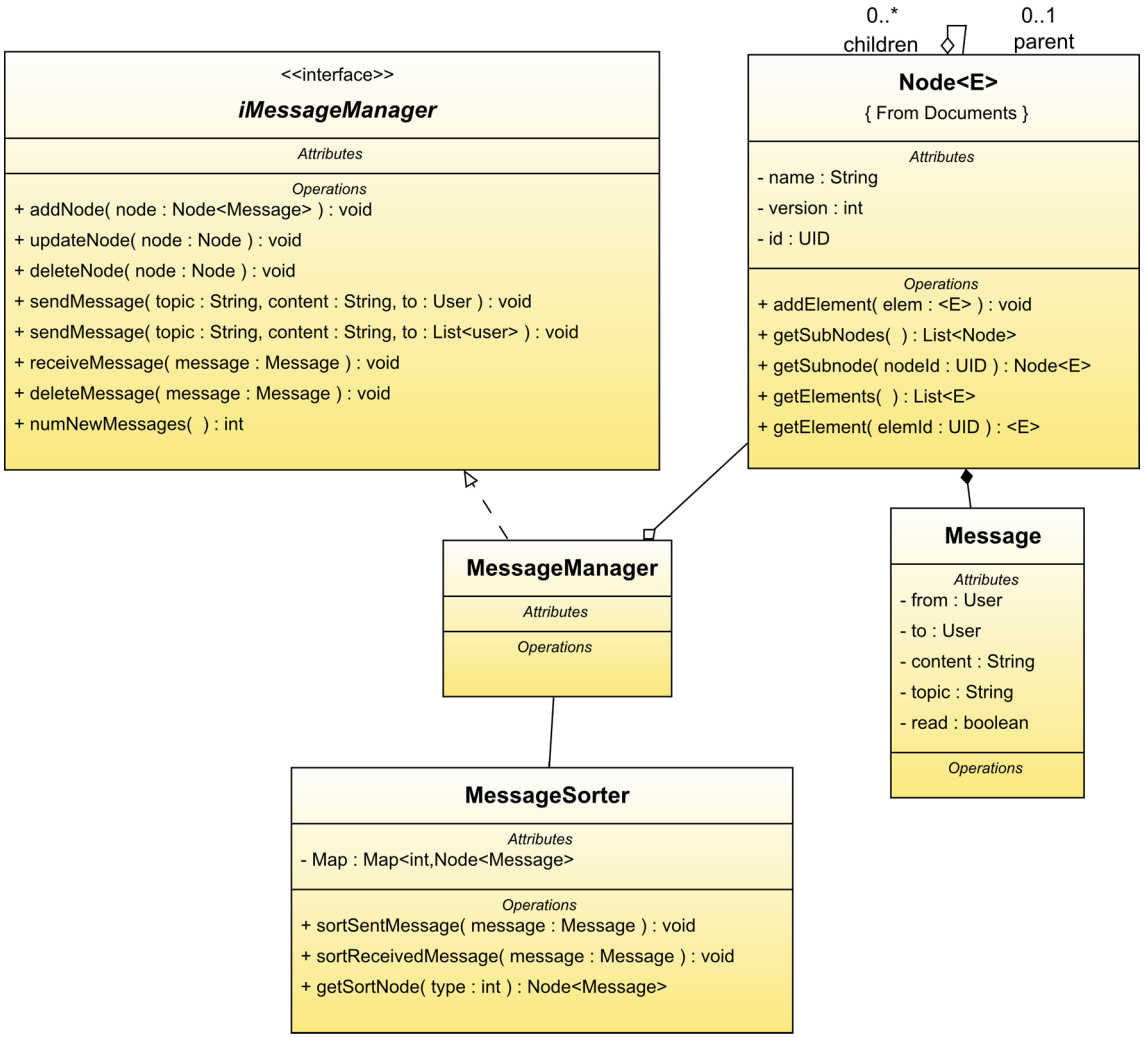

Abbildung 4.19.: MessageManager

4.8.18. WebcamManager

Die Kommunikation per Webcam wird im Allgemeinen über einen Stream realisiert, welcher an den Server geschickt und von dort aus weiter an alle an der Session teilnehmenden Benutzer weitergeleitet wird. Diese Komponente wird nur oberflächlich und der Vollständigkeit halber aufgeführt.

4.9. Zusammenspiel der einzelnen Komponenten

Ausgewählte funktionale Anforderungen sollen in Form von Sequenzdiagrammen dargestellt werden, um das Zusammenspiel der verschiedenen Komponenten zu verdeutlichen. Dabei soll beim Sequenzdiagramm für die Anforderung Dokument erstellen (F:D:A1) der Push-Mechanismus des Servers vorgestellt werden, welcher in Abschnitt 4.8.7 beschrieben wurde. In den folgen Diagrammen soll dieser Prozess vereinfacht dargestellt werden. Weiterhin wird die Funktionsweise der Anwendung anhand des Strukturelementes exemplarisch dargestellt.

4.9.1. Dokument erstellen

Ein Dokument zu erstellen, lässt sich in drei Phasen aufteilen: `TextDocument`-Objekt erstellen, `TextDocument`-Objekt über den Push-Mechanismus synchronisieren und die Neuigkeit an Interessierte übermitteln.

Der Benutzer klickt auf einen Knoten und wählt im Kontextmenü „Schriftdokument erstellen“. Es wird ein Event erzeugt, welches dem Kontextmenüelement „Schriftdokument erstellen“ zugeordnet ist. Das Event fordert über den Server die Gruppen an, denen der Benutzer angehört. Nach der Serverantwort wird das Fensterobjekt erstellt, welches die Eingaben für die Dokumentenerstellung entgegennimmt. Sobald der Benutzer auf „Dokument erstellen“ klickt, wird durch einen entsprechenden Listener das `TextDocument`-Objekt erstellt und synchronisiert, indem ein `DocumentCreatedEvent` geschickt wird.

Bei einem neu erstellten `PushableObject`, wie dem `TextDocument`, steht das `needsToBePushed`-Flag auf **`true`**. Der zuständige Listener prüft, ob das Objekt synchronisiert werden muss. Da dies der Fall ist, wird es an den Server übergeben und das `needsToBePushed`-Flag auf **`false`** gesetzt. Der Server leitet die Weitergabe des Objektes an ClientB ein, wo es eingefügt wird. Das Einfügen erzeugt ein `documentCreatedEvent`. Da bei diesem Aufruf des Listeners das `needsToBePushed`-Flag von `TextDocument` auf **`false`** steht, erfolgt keine weitere Verteilung. Wichtig hierbei ist, dass jede Änderung und jede Erstellung eines Objektes erst an den Server geschickt und anschließend an alle Clients verteilt wird, auch an den Client, der die Synchronisierung selbst verursacht hat. Abbildung 4.20 zeigt dabei den vereinfachten Ablauf des Vorgangs.

Nun werden Nachrichten an alle Benutzer geschickt, die Teil der Gruppe sind, welcher das Dokument zugeordnet ist.

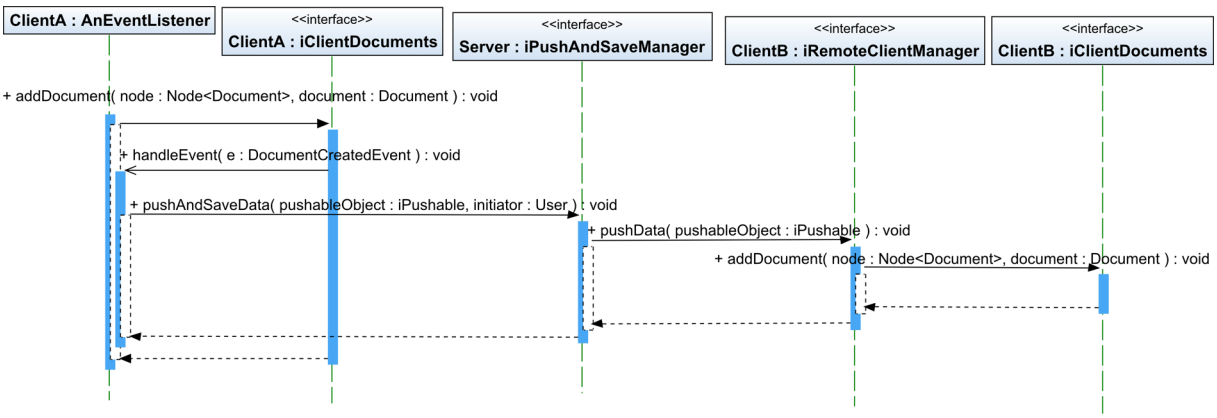

Abbildung 4.20.: Dokument erstellen

4.9.2. Strukturelement erstellen

Ausgangspunkt des Ablaufes ist, dass der Benutzer einem Dokument ein neues Strukturelement hinzufügen möchte. Der *EventListener* wird dabei durch den klick auf den Knopf „Strukturelement erstellen“ angestoßen und erstellt das Fenster `CreateStructureElementView` um die Eingaben des Benutzers entgegenzunehmen (Siehe Abbildung 4.21). Wenn der Benutzer auf den „erstellen“Knopf klickt, wird der zuständige EventListener die Daten in das Modell einfügen. Das Einfügen der Daten erzeugt wiederum ein Event, das veranlasst, dass das gerade erstellt Objekt an den Server übergeben wird. Der Server speichert das Objekt und verteilt es falls nötig weiter. Der Ablauf des push-Mechanismus wird im Abschnitt „Dokument erstellen“ 4.9.1 erläutert. Abschließend wird wieder die `DocumentView`, das Fenster für die Dokumentenübersicht, angezeigt.

4.9.3. Strukturelement bearbeiten

Ein Strukturelement zu bearbeiten lässt sich in drei Abschnitte unterteilen: Einer Arbeitssitzung beitreten, eine Änderung durchführen und die Arbeitssitzung wieder verlassen.

ClientA möchte ein Strukturelement bearbeiten. Da es noch keine existierende Arbeitssitzung für die Komponente gibt, wird vom Server eine neue erstellt und ClientA tritt dieser Arbeitssitzung bei. ClientB möchte wiederum dasselbe Objekt bearbeiten und tritt der bereits existierenden Arbeitssitzung ebenfalls bei. Der Server leitet die Information, dass ClientB der Arbeitssitzung beigetreten ist, weiter. Der Beitritt löst bei ClientA einen Event aus welcher die Anzeige aktualisiert, ClientA sieht nun, dass ClientB der Sitzung beigetreten ist.

Abbildung 4.22 verdeutlicht den Ablauf des Push-Mechanismus bei einer Modifikation eines Attributes. ClientA gibt nun einen Buchstaben in das Bearbeitungsfeld ein. Diese Eingabe

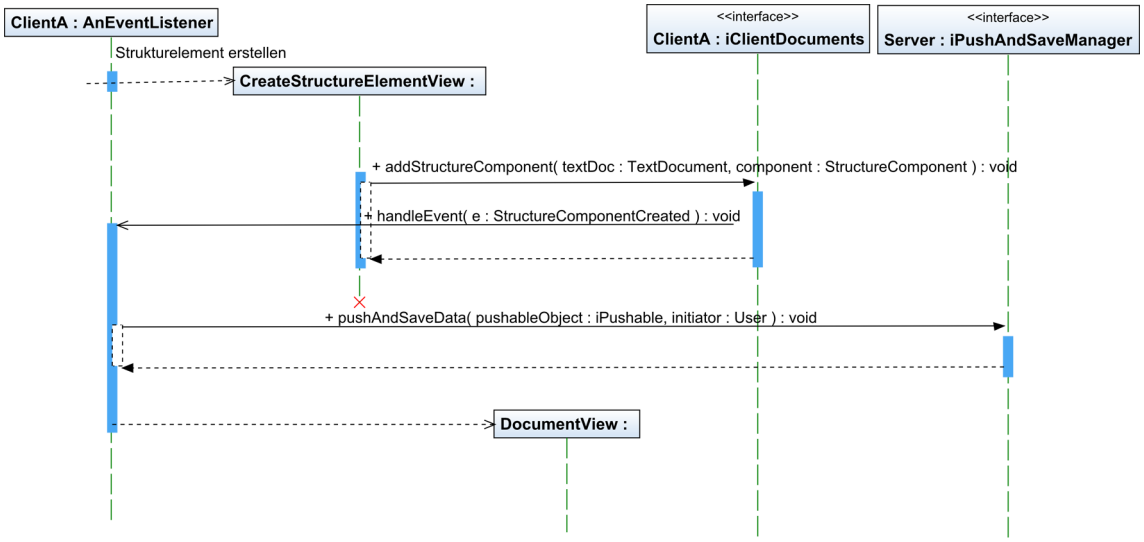

Abbildung 4.21.: Strukturkomponente erstellen

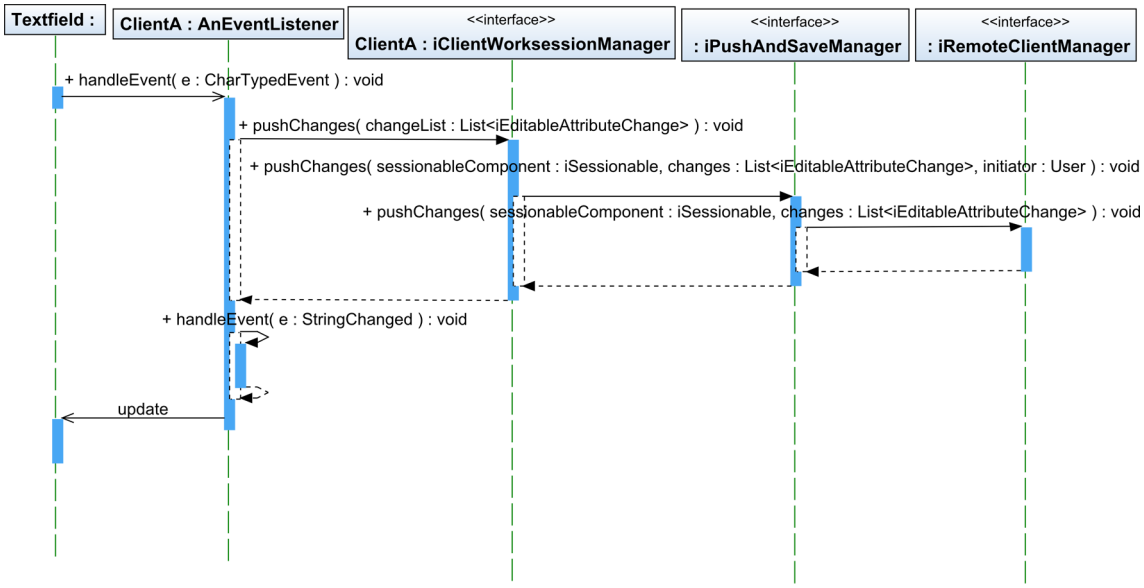

Abbildung 4.22.: Strukturelement bearbeiten: Verbreitung von Änderungen

erzeugt einen Event, welcher die Änderungen, die in der `ChangeList` festgehalten sind, an den Server weitergibt. Von dort aus werden sie weiter an alle Clients verteilt, auch an den Client, der die Änderung verursacht hat. Der eigentliche Event der graphischen Komponente, welcher durch die Eingabe erzeugt wurde, wird blockiert, da die Änderungen selbst am Modell ausgeführt werden. Die Benutzeraktionen werden also über die Arbeitssitzung umgeleitet.

Verlässt nun ClientB die Sitzung, ist der Ablauf ähnlich wie beim Beitreten der Arbeitssitzung. Sobald der letzte Benutzer die Arbeitssitzung verlässt, wird der aktuelle Stand des Strukturelements gespeichert und die Änderung mit allen Clients über den Push-Mechanismus synchronisiert. Sollten während der Arbeitssitzung Daten hochgeladen worden sein, wird beim Benutzer, welcher als letztes die Arbeitssitzung verlässt, ein Dialog angezeigt über den er eventuell noch temporäre Daten für eine permanente Speicherung markieren kann. Nachdem die gewünschten Daten gespeichert und verteilt worden sind, verlässt der Benutzer die Arbeitssitzung und der Server schließt diese ab.

4.9.4. Strukturelement sperren

Der Benutzer klickt das „sperren“ Symbol eines Strukturelementes. Das Fenster *CreateLockView* erscheint und der Benutzer gibt die relevanten Informationen ein. Nachdem der Benutzer auf den „Sperren“-Knopf des Fensters drückt, wird ein Event erzeugt. Dieser Event löst die Methode `aquireLock` aus, um eine Sperre anzufordern. Bevor die Sperre gewährt werden kann, muss überprüft werden, ob evtl. bereits eine Sperre auf ein Elternelement gelegt wurde. Ist dies nicht der Fall, wird eine Sperre für das Element erstellt und anschließend verteilt.

4.9.5. Strukturelement kommentieren

Dieser Fall läuft ähnlich ab, wie das Erstellen eines Strukturelementes. Das Fenster `CreateCommentView` erscheint, wenn man auf „einen Kommentar hinzufügen“ neben dem Strukturelement klickt. Nachdem alle Eingaben getätigt worden sind, klickt der Benutzer auf „erstellen“Ės wird ein `CommentCreatedEvent` geschickt, der Kommentar erstellt und der Push-Mechanismus zur Synchronisierung eingeleitet. Abbildung 4.24 stellt den Ablauf dar.

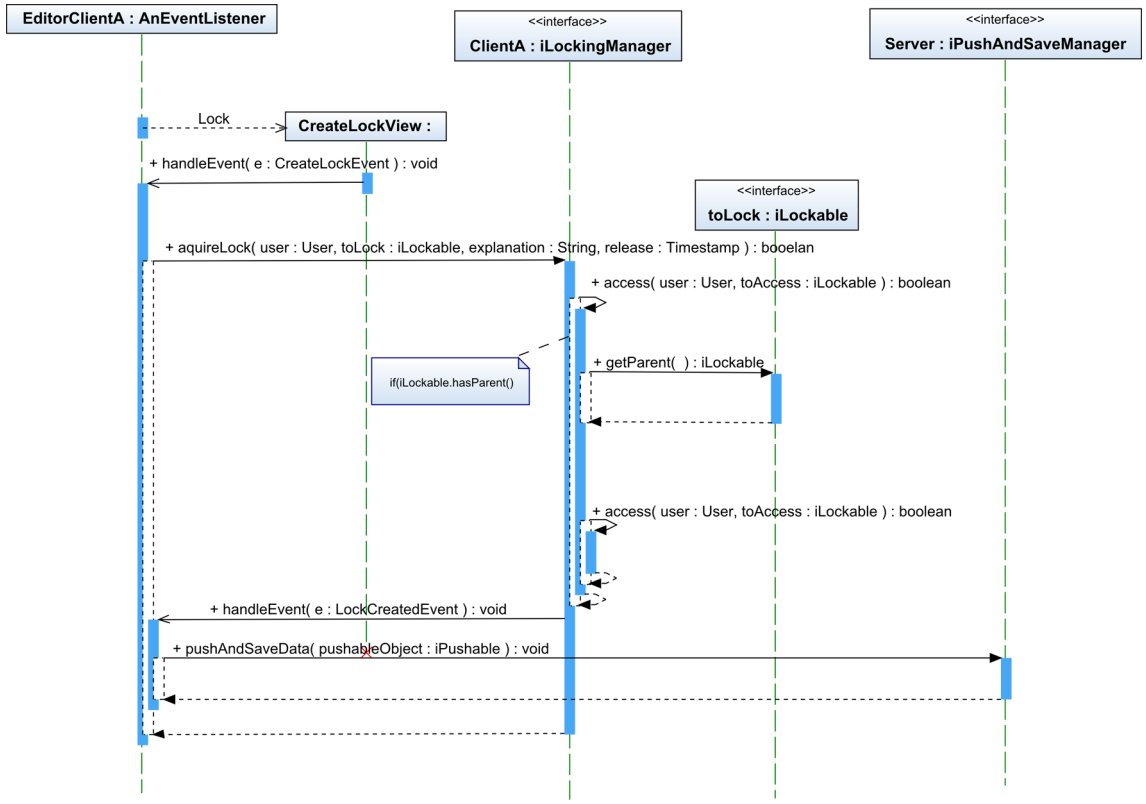

Abbildung 4.23.: Strukturelement sperren

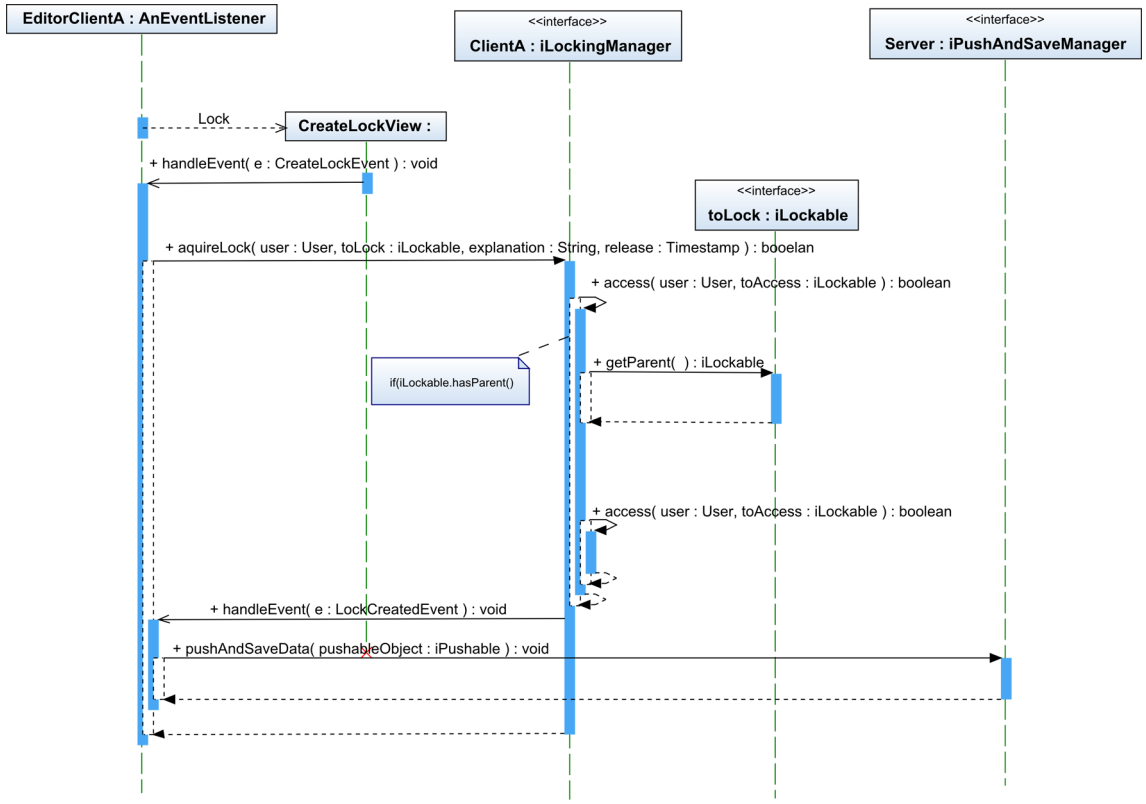

Abbildung 4.24.: Strukturelement kommentieren

4.10. Anforderungen

Aus der vorgestellten Referenzarchitektur sollen nun die Anforderungen abgeleitet werden, welche Flex hinsichtlich der Umsetzung erfüllen muss. Dabei sollen zuerst die Anforderungen aus den Architektur-, dann aus den Entwurfsmustern und abschließend die Anforderungen aus den Spracheigenschaften dargestellt werden.

4.10.1. Anforderungen aus Architekturmustern

Zuallererst muss eine Trennung in Schichten sinnvoll möglich sein, zudem müssen diese Schichten entsprechend der Referenzarchitektur auf Client und Server verteilt werden können. Um die Kommunikation zwischen Client und Server auf die vorgestellte Weise zu gewährleisten, muss einen Objekt-Broker genutzt werden. Die Benutzerschnittstelle wird nach dem MVC-Muster umgesetzt.

- Umsetzung der Schichtenarchitektur. Die Beschreibung der Schichten und die Kategorisierung der Client-Server-Anwendung erfolgte in Abschnitt 4.8.1.
- Umsetzung des Broker-Muster, in diesem speziellen Fall soll ein Objekt-Broker genutzt werden. Die Kommunikation anhand des Brokers wurde in Abschnitt 4.8.2 erläutert.
- Umsetzung der clientseitigen graphischen Benutzerschnittstelle anhand des MVC-Musters. Die graphische Benutzerschnittstelle wurde in Abschnitt 4.8.3 beschrieben.

4.10.2. Anforderungen aus Entwurfsmustern

Es ließen sich folgende Entwurfsmuster anhand der Referenzarchitektur identifizieren und müssen dementsprechend umsetzbar sein:

- Fassade, wird unter anderem in Abschnitt 4.8.5 verwendet.
- Beobachter, wird zur Umsetzung des MVC-Musters (Siehe Abschnitt 4.8.3) und zur Realisierung des Synchronisierungs-Mechanismus genutzt (Siehe Abschnitt 4.8.14).

4.10.3. Anforderungen aus Spracheigenschaften

Auf unterster Ebene liegen die Spracheigenschaften, die nötig sind, um die oben genannten Muster wie in diesem Design vorgestellt, umzusetzen zu können. Um die Java-Referenzarchitektur analog umsetzen, müssen folgende Spracheigenschaften von Flex unterstützt werden:

- Vererbung: Genereller Aufbau.
- Polymorphie: Genereller Aufbau.
  - Method-Overriding: Genereller Aufbau.
  - Method-Overloading: Genereller Aufbau. Wird z.B. in Abschnitt 4.8.17 genutzt, um auf verschiedene Arten Nachrichten zu schicken.
  - Definiton abstrakter Klassen: Genereller Aufbau. Wird z.B. in Abschnitt 4.8.15 verwendet, um die Klasse TextComponent definieren.
- Exception-Handling: Generelles Verfahren, um Fehler zur Laufzeit behandeln zu können.
- Datenkapselung anhand von Schnittstellen: Genereller Aufbau.
- Generics: Finden bei der Node-Klasse Anwendung, siehe Abschnitt 4.8.9.
- Reflexion: Findet beim Push-Mechanismus Anwendung, siehe Abschnitt 4.8.7.

5. Flex

In diesem Kapitel soll das Rich Internet Application Framework Flex, der Firma Adobe[1] vorgestellt werden. Dabei sollen die Themen des Frameworks behandelt werden, welche zur Schilderung der Machbarkeitsstudie in Kapitel 6 nötig sind. Zuerst soll einen Überblick der historischen Entwicklungen des Frameworks gegeben werden. Anschließend soll der allgemeine Aufbau von Flex anhand der beiden Sprachen MXML und ActionScript sowie die Unterscheidung des Flash Players und des Flex Frameworks geschildert werden. Das Grundlagenkapitel von Flex soll zum einen sprachspezifische Konzepte von ActionScript und zum anderen für das Framework spezifische Konzepte des Flex Frameworks wie Datenbindung und Remote-Kommunikation erläutern. Abschließend soll im Kapitel „Entwicklung mit Flex“ eine Vorstellung der Entwicklungsumgebung, sowie der Themen Debugging, Logging und Unit-Testing erfolgen, welche bei der Entwicklung mit Flex eine Rolle spielen.

Für ein tieferes Verständnis der vorgestellten und auch für die nicht behandelten Themen bezüglich des Flex Frameworks empfiehlt der Autor dieser Arbeit das Buch Kazoun C. (1. Auflage 2008).

5.1. Entwicklung von Flex

Version 1.0 wurde von Macromedia entwickelt und erschien im März 2004(Wikipedia (a)). Flex 1.0 wurde dabei unter dem Namen „Flex Presentation Server“ als ein Flash Server für Java positioniert, mit einem nicht unerheblichem Preis (pro CPU ca. 6000 EUR, siehe actionscript praxis.de). Das Kernelement war ein entwicklerfreundliches und leistungsfähiges Framework zur Entwicklung von Flash Anwendungen. Zur Definition von graphischen Komponenten wurde eine XML basierte Sprache verwendet. Für die Entwicklung wurde ein Plug-in für die ebenfalls von Macromedia erstellte Anwendung Dreamweaver namens Flex-Builder entwickelt.

Die Serverkomponente bestand aus einem Compiler und einer in Java implementierten Kommunikationskomponente. Der Compiler übersetzte den Code bei Bedarf in eine SWF-Datei, das Format einer Flash-Anwendung. Die SWF-Datei wurde dann auf dem Client ausgeführt.

[1] http://www.adobe.com/

Die Kommunikationskomponente ermöglichte es Flash, über ein optimiertes binäres Protokoll mit dem Java-Backend zu kommunizieren.

Version 1.5 brachte im Oktober 2004 (Wikipedia (a)) nur relativ kleine Änderungen mit sich, wie z.B. die Möglichkeit, Bibliotheken zu erstellen, um auf dasselbe Modul von mehreren Anwendungen aus zugreifen zu können (release Notes (a)). Im Dezemer 2005 wurde Macromedia von Adobe übernommen Wikipedia (b). Nach einer eingehenden Prüfung der von Macromedia entwickelten Technologie Flex durch Adobe entschied man sich zu einem Kurswechsel bezüglich der Positionierung am Markt, zudem sollte Flex nicht mehr nur als reine Server-Lösung eingesetzt werden können. Weiterhin sollte die Entwicklungsumgebung den Anforderungen einer Entwickler-IDE wie z.B. Visual Studio[2] oder Eclipse[3] genügen, da Dreamweavers Schwerpunkt auf der Entwicklung und Gestaltung von Webseiten lag und viele gängige Features wie z.B. contentAssist oder Refactoring nicht unterstützte.

Version 2.0 erschien im Juni 2006 (Wikipedia (a)). Der Kernpunkt von Flex2, das Flex-Framework, ist eine Bibliothek von ActionScript3-Objekten und stellt die Grundlage für den Aufbau von unter Flash laufenden Rich Internet Applications dar. Weiterhin besteht Flex nicht mehr zwingend aus einer Client- und einer Serverkomponente. Zeitgleich zu der Einführung von Actionscript 3 wurde der FlashPlayer 9 eingeführt, der es über eine virtuelle Maschine (ActionScript Virtual Machine 2 - AVM2) erlaubt, Actionscript 3-Code über einen Just-in-Time-Compiler in maschineneigenen Code umzuwandeln.

Als Entwicklungsumgebung wurde der neue Flex-Builder als ein Plugin der Open-Source IDE Eclipse realisiert. Er stellt eine umfassende Umgebung für die Bearbeitung eines Flex-Projektes dar. Weiterhin setzt Flex Builder nun auf eine bei Entwicklern bekannte Entwicklungsumgebung, die sich einer breiten Community erfreut, und profitiert durch deren Synergieeffekte.

Die Flex-Data-Services(FDS) stellen die Weiterentwicklung des ursprünglichen Flex-Servers dar. FDS wurde dabei um diverse Features erweitert, wie z.B. Client/Server-Messaging, Integration des Java Messaging Service[4], ein Rich-Data Model- und Datensynchronisationsframework, Data-Paging und Proxy-Services. Dabei stellt die bidirektionale Unterstützung von Messaging zwischen Client und Server eine Kernerweiterung dar.

Im Zuge der neuen Marktstrategie wurde das Flex-SDK samt Compiler und anderen Utilities frei zur Verfügung gestellt. Die AVM2 wurde zu OpenSource erklärt. Das Ziel war, einen höheren Akzeptanz- und Verbreitungsgrad zu erreichen. Der FDS ist dabei kommerziell in

[2] http://www.microsoft.com/germany/msdn/vstools/default.mspx
[3] http://www.eclipse.org/
[4] http://java.sun.com/products/jms/

einer freien Express-Edition unter der Einschränkung einsetzbar, dass Anwendungen nicht geclustert oder über mehrere CPUs verteilt laufen können. Der Flex Builder hingegen ist proprietär und kostenpflichtig, wobei ein Studentenversion zur unentgeltlichen Nutzung für Schüler und Studenten existiert.

Die Informationen des obigen Abschnittes wurden größten Teils aus (Kazoun C. (1. Auflage 2008), Geleitwort S.13 - S.17) entnommen.

Version 3.0 wurde im Februar 2008 (Wikipedia (a)) veröffentlicht. Die Kernpunkte liegen in der Unterstützung von Adobes hauseigener Laufzeitumgebung Air, sowie der Flex Ajax Bridge Bibliothek (FABridge). Air ermöglicht es, Rich Internet Applications lokal auf dem Rechner wie eine Desktopanwendung laufen zu lassen. Die FABridge ermöglicht es ohne zusätzlichen Aufwand, ActionScript-Klassen für JavaScript verfügbar zu machen. Weiterhin wurde auch Flex selbst samt Compiler, Debugger und weiteren Utilities als Open Source veröffentlicht (vgl. release Notes (b)).

Die serverseitige Komponente Flex Data Services wurde auf den Namen „Flex Livecycle Data Services ES“[5] umgetauft. Ein Teil der Flex Livecycle Data Services wurde unter dem Namen BlazeDS[6] als Open Source veröffentlicht.

Da die aktuelle Version von Flex erst nach Beginn der Bearbeitungszeit dieser Arbeit veröffentlich wurde, kann diese nicht angemessen berücksichtigt werden. Evaluiert wird die Vorgängerversion 2.0.

5.2. Allgemeiner Aufbau von Flex

Flex besteht aus den beiden Sprachen MXML und ActionsScript (Version 3.0). Dabei soll dieses Kapitel hinsichtlich des Aufbaus und Einsatzes der beiden Sprachen als Wissensbasis für Kapitel „Grundlagen von Flex“ dienen. Es sollen die grundlegenden Konzepte erklärt und das Zusammenspiel von MXML und ActionScript erläutert werden. Abschließend soll der Unterschied zwischen Flash Player und Flex Framework verdeutlich werden.

5.2.1. MXML

MXML ist eine deklarative Markup-Sprache und basiert auf XML. Laut Wikipedia (c) gibt Adobe keine Erklärung für die Bedeutung des „M“ in MXML. Es wird verwendet um die grafischen

[5] http://www.adobe.com/products/livecycle/dataservices/
[6] http://opensource.adobe.com/wiki/display/blazeds/

Komponenten einer Flex-Anwendung schneller, übersichtlicher und einfacher zu erstellen. In MXML werden Tags verwendet, um Komponenten wie Buttons und Texteingabefelder zu erstellen, deren Reaktion auf Ereignisse zu konfigurieren sowie Daten an Komponenten zu binden (vgl. Kazoun C. (1. Auflage 2008)S.49). Dabei ist es möglich, jede ActionScript-Klasse auch in MXML zu definieren; mehr dazu in Abschnitt 5.2.3.

Knotentypen

Alle MXML-Dokumente müssen einen Stammknoten besitzen. Dabei gibt es zwei verschiedene Typen von Stammknoten: Application-Knoten für Anwendungsdokumente und Komponenten-Knoten für Komponentendokumente. Komponentendokumente haben eine existierende Komponente als Stammknoten. Die Struktur der beiden Dokument-Typen ist ansonsten gleich. Eine Flex-Anwendung hat nur ein Anwendungsdokument, kann aber mehrere Komponentendokumente haben. Durch die Komponentendokumente kann man Teile der Anwendung zusammenfassen zu Gunsten der Pflegbarkeit der Anwendung.(vgl. Kazoun C. (1. Auflage 2008)(S. 50-51))

Namensraum

Ein Namensraum dient dazu, Actionscript-Klassen über eine Manifest-Datei auf einen Namensraum zu mappen. `mx:Button` verweist zum Beispiel auf eine Klasse namens `mx.controls.Button` aus dem Flex-Framework. Ein Namensraum dient also zur Abgrenzung von Klassen, um Namenskonflikte zu vermeiden und die Übersichtlichkeit von Paketen zu erhöhen. Der mx-Präfix ist eine Konvention für den Namensraum der Komponenten des FlexFrameworks (vgl. Kazoun C. (1. Auflage 2008) S. 52 - S.53).

Komponententypen

MXML-Komponenten können wie folgt kategorisiert werden:

visuelle Komponenten	nicht-visuelle Komponenten
Container	Datenkomponenten
UI-Elemente	Utilitykomponenten

Container Container können andere Komponenten aufnehmen, ein Beispiel hierfür ist das Application-Element. Container werden genutzt, um das Layout der Anwendung festzulegen. Das Flex-Framework bringt bereits Container für viele verschiedene Arten von Layout wie z.B. horizontales, vertikales oder Gitter-Layout mit.

UI-Elemente Dies sind die Komponenten, mit denen der Benutzer interagieren kann. Unter diese Kategorie fallen z.B. Buttons oder Texteingabefelder (vgl. Kazoun C. (1. Auflage 2008), S. 54 - S.55).

Datenkomponenten Dieser Komponenten-Typ wird zum einen für den Aufbau von Arrays oder Collections und zum anderen bei der Kommunikation mit entfernten Diensten verwendet (vgl. Kazoun C. (1. Auflage 2008), S. 57).

Utilitykomponenten Utilitykomponenten bieten Funktionalitäten für voneinander unabhängige Aufgaben (vgl. Kazoun C. (1. Auflage 2008) S. 57).

Interaktivität mit MXML

Grundsätzlich gibt es zwei verschiedene Arten, wie über MXML Interaktivität erzeugt werden kann, nämlich Ereignisse und Datenbindung:

Ereignisse Verschiedene Komponenten haben verschiedene Verhaltensweisen. Zum Beispiel können alle visuellen Komponenten sich selbst initialisieren oder ihre Größe anpassen. Ein Button kann z.B. auf einen Klick des Benutzers reagieren. Diese Reaktionen werden Ereignisse (Events) genannt. Über ein Ereignis ist es dabei einer Komponente möglich, andere Komponenten zu informieren, wenn Aktionen stattfinden. Den Vorgang, des Sendens einer Benachrichtigung, nennt man Ereignis-Dispatch. Es gibt zwei Möglichkeiten, um auf Ereignisse zu reagieren: Zum einen über ActionScript anhand von Listenern oder direkt in MXML. Die Verwendung von Listenern wird in Abschnitt 5.3.1 erläutert. Bei MXML ist es möglich, den Event-Handler direkt im Tag mit anzugeben. Der Name des Attributes entspricht dabei dem Namen des Ereignisses, im Falle von einem Button wäre dies zum Beispiel das click-Attribut. Der einem Ereignis-Attribut zugewiesene Wert wird dabei als ActionScript interpretiert (vgl. Kazoun C. (1. Auflage 2008) S. 57-58).

```
<mx:Button id="warnungButton" label="Warnung!" click="mx.controls.
   Alert.show('Warnung')"/>
```

Listing 5.1: Beim Klicken des Knopfes wird ein Warnfester geöffnet

Datenbindung Mit diesem Feature kann eine Komponente mit einer anderen Komponente oder einem ActionScript-Objekt verknüpft werden.

```
<mx:VBox>
    <mx:Text id="output" text="{input.text}"/>
    <mx:TextInput id="input"/>
</mx:VBox>
```

In dem obigen Beispiel wird das text-Attribut der Text-Komponente output mit dem text-Attribut der TextInput-Komponente input verknüpft. Somit werden alle Änderungen von input.text sofort auch an output.text weitergeleitet (vgl. Kazoun C. (1. Auflage 2008) S. 59).

5.2.2. ActionScript 3.0

ActionScript 3.0 richtet sich nach der ECMA-262-Spezifikation und unterstützt bereits Teile der noch offenen Version 4.0, nähere Informationen sind hier ECMA (a) zu finden. AS3 ist eine objektorientierte Sprache und unterstützt viele Features wie z.B. formalisierte Klassen, Interfaces, Pakete, Ausnahmebehandlung zur Laufzeit, Reflexion und mehr. AS wird dazu verwendet, um Datenmodelle und anspruchvollere clientseitige Geschäftslogik zu entwickeln. Da AS eine objektorientierte Sprache ist, kann man sie auch als eine Sammlung von Schnittstellen betrachten. Innerhalb AS lassen sich drei Ebenen von Schnittstellen abgrenzen:

1. Flash Player-APIs: In den Flash Player-APIs sind die elementaren Klassen wie z.B. String, Number, Date und Array verankert. Zusätzlich existieren noch für den Flash Player spezifische Klassen, wie z.B. DisplayObject, Video oder NetConnection. Die Schnittstellen des Flash Players können direkt in seiner Laufzeitumgebung ausgeführt werden.
2. Flex Framework-APIs: Diese Schnittstellen stellen das Flex Framework dar. Da das Framework selbst in AS geschrieben ist, nutzt es die „primitiven“ APIs des Flash Players. Man könnte es als die Schicht bezeichnen, die auf die Player-API aufsetzt. Das Flex Framework besteht aus Containern, Steuerelementen sowie den dazugehörigen Daten-, Manager- und Utility-Klassen.
3. Eigene APIs: Schnittstellen und Klassen, welche in eigenen Anwendungen zum Einsatz kommen. Diese können auf Flash Player- und auch auf Flex Framework-APIs zugreifen.

(vgl. Kazoun C. (1. Auflage 2008) S.61-62)

Verwendung von AS

ActionScript kann auf vier verschiedene Wege verwendet werden:

- Inlining in MXML-Tags
- geschachtelt in MXML-Tags
- in MXML-Scripten
- als ActionScript-Klassen

Inlining ActionScript kann direkt in einem MXML-Tag eingebunden werden. Dies ist zum Beispiel bei Inline-Event-Handler (siehe Abschnitt 5.1) oder -Datenbindungen (siehe Abschnitt 5.2.1) der Fall. Allerdings können Inline-Datenbindungen nur einen ActionScript-Ausdruck in den geschweiften Klammern enthalten. Bei Inline-Event-Handlern kann hingegen eine beliebige Anzahl von ActionScript-Ausdrücken verwendet werden. Das Beispiel aus Abschnitt 5.1 lässt sich wie folgt erweitern:

```
<mx:Button id="warnungButton" label="Warnung!" click="mx.controls.
   Alert.show('Warnung');warnungButton.label='noch mal!'"/>
```

Zuerst wird das Warnfester geöffnet und dann steht auf dem Knopf nicht mehr „Warnung!“ sondern „noch mal! “Üm besseren und einfach zu lesenden Code zu produzieren, wird jedoch meist davon abgesehen, mehrere ActionScript-Ausdrücke zu verwenden, und anstelle dessen eine Methode als Event-Handler referenziert.

Geschachtelt ActionScript kann man zusätzlich auch in MXML-Tags schachteln. Dies funktioniert genau wie die Schachtelung der meisten Eigenschaften. Dabei ist es nur möglich, die Werte für Event-Handler zu schachteln. Da AS-Code Sonderzeichen beinhalten könnte, muss der Code innerhalb eines CDATA-Blockes stehen, da diese ansonsten von dem MXML-Compiler interpretiert werden.

```
<mx:Button id="warnungButton" label="Warnung!">
    <mx:click>
        <![CDATA[
            mx.controls.Alert.show('Warnung');
            warnungButton.label='noch mal!';
        ]]>
    </mx:click>
</mx:Button>
```

Das obige Beispiel stellt nur eine andere, etwas übersichtlichere Schreibweise dar, es funktioniert genauso wie das Beispiel aus Abschnitt 5.2.2.

In einem Script-Tag ActionScript-Code kann innerhalb des Script-Elements stehen, muss aber wieder von einem CDATA-Block umschlossen werden.

```
<mx:Script>
    <![CDATA[
        private function beispiel():void{
            mx.controls.Alert.show('Warnung');
            warnungButton.label='noch mal!';
        }
    ]]>
</mx:Script>
<mx:Button id="warnungButton" label="Warnung!" click="beispiel()">
```

Listing 5.2: Funktioniert genauso wie Beispiel 5.2.2. Der auszuführende Code wurde nur in die Methode `beispiel` ausgelagert.

Der ActionScript-Code kann auch in eine andere Datei ausgelagert und dann über das Script-Element folgendermaßen eingebettet werden:

```
<mx:Script source="beispiel.as"/>
```

Klassen Der Code für ActionScript-Klassen wird in eigenen Dokumenten von den MXML- und Komponentendokumenten getrennt gehalten. Klassen stellen das wichtigste und leistungsfähigste Einsatzgebiet von ActionScript 3.0 dar.

5.2.3. Zusammenspiel von MXML und Actionscript

MXML ist sehr gut geeignet, um Benutzerschnittstellen zu entwickeln. Hingegen sollte man ActionScript verwenden, um Daten- und Geschäftsmodelle zu definieren. Aus den beiden vorherigen Abschnitten ist zu erkennen, das zwischen MXML und ActionScript eine Kopplung bestehen muss. MXML wird während der Kompilierung in ActionScript umgewandelt. Eine MXML-Komponente lässt sich also anhand des Begriffes einer ActionScript-Klasse verstehen. Das folgende MXML-Tag erzeugt einen Button, das MXML-Tag-Attribute **`label`** verhält sich analog zu der gleichnamigen Objekt-Eigenschaft, das Attribut `id` korrespondiert mit dem Namen der Objektinstanz:

```
<mx:Button id="myButton" label="Dies ist mein Button"/>
```

Das vorangehende MXML-Tag ist mit dem folgendem ActionScript-Code äquivalent:

```
var myButton:Button = new Button();
myButton.label = "Dies ist mein Button";
```

Ebenso wie MXML-Komponenten entsprechen auch MXML-Dokumente ActionScript-Klassen. Das Anwendungsdokument erweitert die Klasse mx.core.Application, Komponentendokumente erweitern die entsprechenden Komponentenklassen. Wird Code über ein MXML-Skript eingebunden, entspricht dies dem Platzieren des Codes im Rumpf der Klasse. Variablendefinitionen in MXML-Skripten werden als Eigenschaften und Funktionen als Methoden der Klasse betrachtet. Die Regeln für die Entwicklung von ActionScript-Klassen finden also auch bei der Entwicklung von MXML Anwendung. MXML vereinfacht also die Entwicklung von Klassen, die der Benutzerschnittstelle angehören, indem das Flex Framework viele wichtige Aufgaben wie die Initialisierung und die Anwendung von Layoutregeln übernimmt und so vor dem Entwickler versteckt (vgl. Kazoun C. (1. Auflage 2008) S.65 - S.67).

Meta-Tags

Über Meta-Tags werden dem Compiler zusätzliche Information über Komponenten oder Klassen geliefert. Diese Informationen weisen den Compiler z.B. an, alles Nötige vorzubereiten, um eine Klasse datenbindungsfähig zu machen. Das folgende Beispiel zeigt, wie ein Meta-Tag des Typs Bindable unter der Angabe von Parametern deklariert wird (vgl. Adobe (e)).

```
[Bindable(event="nameChanged")]
```

In Abschnitt „Datenbindung benutzerdefinierter Klassen“ (5.3.2) auf Seite 95 wird das Meta-Tag `[Bindable]` eingesetzt, um eine Klassen datenbindungsfähig zu machen.

5.2.4. Unterscheidung zwischen Flash Player und Framework

Der Flash Player ist die Laufzeitumgebung für Flash- und Flex-Anwendungen, dementsprechend können die beiden Anwendungstypen nur das tun, wofür der Flash Player ihnen die Dienste anbietet. Der Unterschied zwischen Flash und Flex-Anwendungen ist also nicht der Inhalt, sondern die Art, wie dieser erstellt wurde.

Bei Flash-Anwendungen steigt die Größe der .swf-Datei proportional zu dem entwickelten Code und den eingebunden Dateien (z.B. Bilder) an, da sie nur die Dienste des Flash Players verwenden. Da bei Flex-Anwendungen auf die Bibliotheken des Flex Frameworks zugegriffen wird, müssen diese natürlich auch geladen werden und erhöhen dementsprechend auch die Größe der .swf-Datei.

Die Dienste des Flash Players lassen sich anhand der Paketstruktur identifizieren, da sie mit einem führenden „flash“ gekennzeichnet sind. Die Klassen des Flex Frameworks sind

hingegen mit einem führenden „mx“ versehen (vgl. Kazoun C. (1. Auflage 2008) S.100 - S.101).

Im Anhang unter Kapitel A.2, wird beschrieben, wie der Lebenszyklus einer Flex Anwendung und das Laden einer Flex Anwendung in eine andere Flex Anwendung erfolgt.

5.3. Grundlagen von Flex

In diesem Kapitel sollen zuerst die fundamentalen Sprachkonzepte der Klassen, Variablen, Methoden, Arrays, Vererbung, Interfaces, Ereignisse, Fehlerbehandlung, XML und Reflexion von ActionScript 3.0 dargestellt werden. Anschließend sollen die Konzepte der Datenbindung und der Remote-Kommunikation des Flex Frameworks behandelt werden.

5.3.1. Grundlagen von ActionScript 3.0

Klassendeklaration

Eine Klasse muss eindeutig identifizierbar sein. Dies wird anhand von Paketen und dem daraus resultierenden vollständig qualifizierten Klassennamen erreicht. Das Konzept der Pakete dient zusätzlich der Strukturierung von Klassen. Über eine import-Anweisung kann die Klasse über ihre Kurzform, den Klassennamen, angesprochen werden und vereinfacht somit den Einsatz bei der Entwicklung. Gibt es zwei Klassen, welche über dieselbe Kurzform aufgerufen werden, müssen diese über den vollständig qualifizierten Klassennamen adressiert werden. Da dies als eine gängige Praxis auch in anderen objektorientierten Sprachen eingesetzt wird, soll hier nicht weiter ins Detail gegangen werden.

```
package beispiel
{
    import beispiel.NochEinBeispiel;

    public class Beispiel
    {
    }
}
```

Listing 5.3: Klassendeklaration in AS

Variablen und Eigenschaften

Eine Variable wird als ein benanntes Element definiert, indem man Daten speichern und abrufen kann. Bei der Variablendefinition muss nicht zwingend ein Datentyp angegeben werden, es gilt aber als empfehlenswert, dies immer zu tun, um die Vorteile von festen Typisierungen nutzen zu können. Eine Variable wird wie folgt deklariert:

```
var beispielVariable:Datentyp;
```

Man spricht von einer Variable, wenn diese innerhalb einer Methode definiert wurde und nur dort lokal gültig ist. Außerhalb von Methoden deklarierte Variablen heißen Eigenschaften, die mindestens innerhalb der Klasse verfügbar sind. Die Verfügbarkeit von Eigenschaften wird über Modifier geregelt. Es existieren die gewohnten Modifier public, private und protected. Zusätzlich existiert ein Modifier internal, welcher die Eigenschaft nur innerhalb des Paketes zugänglich macht. Die eben vorgestellten Geltungsbereich-Modifier können weiterhin mit dem static-Modifier kombiniert werden, um Eigenschaften direkt über die Klasse zugänglich zu machen und nicht über die einzelnen Instanzen dieser. Eine Eigenschaft wird genauso wie eine Variable definiert, nur wird sie im Body der Klasse positioniert. Eine Konstante ist einer Eigenschaft sehr ähnlich, nur dass sich der einmal festgelegte Wert nicht ändern kann. Anstelle des var-Schlüsselwortes wird const verwendet.

```
public class Beispiel
{
    public static const CURRENT_BEISPIEL = "
       FlexVariablesAndProperties";
}
```

Listing 5.4: Beispiel einer statischen Konstante

Eine weit verbreitete Konvention benennt die Namen von private und protected Eigenschaften mit einem führenden Unterstrich (_), um diese besser sichtbar von Variablen unterscheiden zu können. Per Konvention werden die Namen von Konstanten groß geschrieben. (vgl. Kazoun C. (1. Auflage 2008) S.67 - S.73)

Methoden

ActionScript erlaubt keine Überladung (Overloading). Es können also nicht Methoden mit denselben Namen und unterschiedlichen Parametern existieren. Der so genannte Restparameter erlaubt es allerdings, null oder mehr zusätzliche Parameter eines unbekannten Typs an eine Methode weiterzugeben. Auf die Werte des Restparameters kann innerhalb einer Methode über ein Array zugegriffen werden. Nach Konvention heißt der Restparameter rest.

Soll bei einem Methodenparameter der Datentyp nicht definiert werden, wird der Wildcard-parameter (*) eingesetzt.

Es ist zudem möglich, Parameter mit Default-Werten zu belegen. Beim Methodenaufruf müssen dabei mindestens die Parameter angegeben werden, welche keine Default-Parameter sind. Werden die Default-Parameter nicht angegeben, werden entsprechend ihre Default-Werte verwendet. Ein Methode könnte z.B. wie `doSomething(need:String,optional:String = "notSet"):void` aussehen (vgl. Adobe (b)).

Beim Überschreiben einer Methode einer Superklasse durch die Subklasse wird das Schlüsselwort override in der Methodendeklaration verwendet. Die Methodensignatur der überschriebenen Methode muss dabei dieselbe bleiben wie die der Superklasse (vgl. Kazoun C. (1. Auflage 2008) S.82 - S.83).

Bei Methoden gelten dieselben Modifier wie bei Eigenschaften. Wird kein Modifier angegeben, wird die Methode als internal gehandelt. Der Konstruktor einer Klasse muss den Namen der Klassen widerspiegeln, als public deklariert werden und darf keinen Wert zurückgeben. Anders als bei Java gehören bei ActionScript der Methodenname, die Parametertypen, der Rückgabetyp und der Zugriffs-Modifier zu der Methoden-Signatur.

Eine Methode kann von ihrer Objektinstanz getrennt werden, dies nennt man eine Bound Method. Eine Bound Method verweist weiterhin intern auf die Instanz, von der sie getrennt wurde. Sie reagiert genauso wie eine Methode, die direkt über die Objektinstanz angesprochen wird. Das dazugehörige Beispiel ist im Abschnitt Reflexion unter 5.3.1 zu finden (vgl. Adobe (b)).

Über den eben genannten Mechanismus ist es möglich, Methoden z.B. als Parameter zu übergeben, dies wird im Abschnitt Ereignisse (5.3.1) auf Seite 86 genutzt, um eine Methode als Handler für ein bestimmtes Ereignis zu deklarieren. Im Abschnitt Reflexion (5.3.1) auf Seite 88 wird der dynamische Aufruf über das Funktions-Objekt erläutert.

Weiterhin können in ActionScript implizite Getter- und Setter-Methoden über die Schlüsselwörter „get“ und „set“ definiert werden. Diese Methoden werden auch Accessors genannt. In dem folgenden Beispiel wird eine private Eigenschaft „_aString“ definiert. Auf diese Eigenschaft kann über die Accessors `public function get aString():String` und „public function set aString(string:String)“ zugegriffen werden. Über die Nutzung von Accessors ist es möglich, zusätzliche Funktionalität bei einem Zugriff auf eine Eigenschaft auszuführen.

```
public class Beispiel
{
    private var _aString:String;

    public function Beispiel(){}
```

```
    public function get aString():String{
        return _aString;
    }
    public function set aString(string:String){
        _aString=string;
```

Das folgende Beispiel zeigt, wie die impliziten Getter- und Setter-Methoden aufgerufen werden.

```
var myBsp:Beispiel = new Beispiel();
myBsp.aString = "Dies ist ein Test";    //Setter-Aufruf, "Dies ist
    ein Test" als Parameter
trace(myBsp.aString)                    //Getter-Aufruf
```

Die Methode trace gibt den Parameter während des Debug-Modus auf der Konsole aus, dies wird in Abschnitt 5.4.2 näher beschrieben (vgl. Kazoun C. (1. Auflage 2008) S.73 - S.77).

Arrays

Arrays werden in ActionScript als nicht streng typisierte Objekte behandelt. Es ist möglich, jede Art von Daten innerhalb desselben Arrays unterzubringen. Der Zugriff ist zum einen über das Methodenprotokoll des Array-Objektes möglich und zum anderen mit Hilfe der literalen Notation.

In ActionScript existieren keine Dictionaries oder ähnliche Typen. Um ein assoziatives Array zu erstellen, wird der Object-Typ verwendet, nicht der Array-Typ. Der Object-Typ ist im Gegensatz zu den meisten Klassen dynamisch und den Instanzen können beliebige Eigenschaften zugewiesen werden. Generell gilt es als besser, Datenmodellklassen zu entwickeln, doch es treten Fälle auf, in denen die Verwendung der Object-Instanz als assoziatives Array nützlich ist.

```
var beispielArray:Object = new Object();
beispielArray["erstes Beispiel"] = "Dies ist das erste Beispiel";
beispielArray["zweites Beispiel"] = "Dies ist das zweite Beispiel"
    ;
```

Das vorangehende Beispiel zeigt die Bildung eines assoziatives Arrays mithilfe des Object-Typs (vgl. Kazoun C. (1. Auflage 2008) S.80 - S.81).

Vererbung

Über das Schlüsselwort extends kann eine Subklasse von einer Superklasse erben. In ActionScript kann eine Subklasse jeweils nur von einer Superklasse erben. Die Subklasse kann dabei nur wie üblich auf die public und protected Eigenschaften sowie Methoden zugreifen. Zusätzlich können Klassen, die in demselben Pakete liegen, auf alle internal Eigenschaften und Methoden zugreifen. ActionScript unterstützt das Überschreiben von Methoden einer Superklasse, mehr dazu siehe Abschnitt 5.3.1, Methoden (vgl. Kazoun C. (1. Auflage 2008) S.82 - S.83).

Interfaces

Interfaces ermöglichen es, die Schnittstellen von der Implementierung zu trennen. Interfaces unterscheiden sich in dem Sinne von Klassen, dass sie keine Eigenschaften deklarieren können, nur public-Schnittstellen sowie die Methodensignatur definieren und nicht die Implementierung. Innerhalb eines Interfaces ist daher keine Angabe eines Modifier erlaubt, da alle Methoden zwingend den public-Modifier unterstützen müssen. Per Konvention fangen die Interfacenamen mit einem großen „I“ an. Klassen können über das Schlüsselwort implements ein Interface implementieren (vgl. Kazoun C. (1. Auflage 2008) S.83 - S.84).

Ereignisse

Ereignisse sind Reaktionen auf Benutzerinteraktionen, auf die Zeit oder auf asynchrones Messaging. Dabei nutzen fast alle ActionScript-Ereignisse unabhängig von der Ursache das gleiche Ereignis-Modell. Innerhalb von ActionScript werden Ereignisse verarbeitet, indem so genannte Listener für die Ereignisse registriert werden. Dabei ist der Listener eine Methode, welche bei einem bestimmten Ereignis benachrichtigt wird.

Eine Klasse, die Ereignisse auslösen kann, muss entweder die Klasse `flash.events.EventDispatcher` erweitern oder das Interface `flash.events.IEventDispatcher` implementieren. Über das Interface oder die Superklasse besitzt das Objekt die public-Methode `addEventListener(type:String, listener:Function, useCapture:Boolean = false, priority:int = 0, useWeakReference:Boolean = false):void`. Die ersten beiden Parameter sind Pflicht und repräsentieren zum einen den Event-Namen und zum anderen die Funktion, die bei der Auslösung des Ereignisses aufgerufen werden soll.

Die Listener-Funktion erwartet einen Parameter des Typs `mx.events.Event` oder einer entsprechenden Subklasse. Der Event-Parameter enthält Informationen über das Ereignis, eine Referenz auf das Objekt, welches das Ereignis ausgelöst hat (die `target`-Eigenschaft)

und eine Referenz auf das Objekt, welches das Ereignis zuletzt weitergeleitet hat (die `currentTarget`-Eigenschaft). Oft beziehen sich die `target`- und die `currentTarget` -Eigenschaft auf dasselbe Objekt. Die Registrierung eines EventListener kann über die Methode `removeEventListener(type:String, listener:Function, useCapture :Boolean = false):void` wieder aufgehoben werden. Da der Flash Player keine Garbage Collection eines Objektes durchführt, solange es noch referenziert wird, ist es wichtig, die EventListener zu entfernen, sobald diese nicht mehr nötig sind (vgl. Kazoun C. (1. Auflage 2008) S.84 - S.86).

Fehlerbehandlung

Actionscript 3.0 unterstützt die Fehlerbehandlung zur Laufzeit. Dabei werden zwei Arten von Laufzeitfehlern unterschieden, synchrone und asynchrone.

Synchrone Fehler treten direkt bei einer bestimmten Ausführung aus. Um synchrone Fehler zu behandeln wird try/catch/finally verwendet. Alle Fehler sind dabei entweder vom Typ `flash.errors.Error` oder eine Subklasse davon.

Asynchrone Fehler treten bei ActionScript deutlich häufiger auf als synchrone. Sie sind Reaktionen auf Netzwerkoperationen. Ein asynchroner Fehler wird wie ein Ereignis abgehandelt, es muss ein EventListener für den Fehlerfall registriert werden.

XML

Innerhalb von ActionScript kann man XML mit zwei verschiedenen Methoden bearbeiten. Zum einen mit einer veralteten XMLDocument-Klasse, sowie der neuen XML-Klasse, welche den „ECMAScript for XML“Standard (E4X) (Die Spezifikation ist unter ECMA (b) zu finden) unterstützt.

XML-Objekte erzeugen Ein XML-Objekt kann zum einen über XML-Literale und zum anderen über den XML-Konstruktor erzeugt werden. XML-Literale werden verwendet, wenn XML-Daten direkt im Code definiert werden sollen.

```
var test:XML =
<beispiele>
    <beispiel>
        <titel>XML in Flex</titel>
        <codeExamples>
```

```
            <code>CodeExample 1_1</code>
            <code>CodeExample 1_2</code>
        </codeExamples>
    </beispiel>
    <beispiel>
        <titel>Reflexion in Flex</titel>
        <codeExamples>
            <code>CodeExample 2_1</code>
            <code>CodeExample 2_2</code>
        </codeExamples>
    </beispiel>
</beispiele>;
```

Sollen die Daten nicht direkt im Code angegeben werden, müssen die Daten als String geladen und danach an den XML-Konstruktor übergeben werden. Dabei werden alle String-Inhalte als XML-Daten interpretiert, auch Leerzeichen, Zeilenumbrüche, Tabulatoren etc..

XML lesen Um XML-Daten aus einem XML-Objekt zu lesen, kann auf die Daten entweder über den E4X-Syntax[7] zugegriffen oder das Document Object Model (DOM) verwendet werden. Dabei können beide Techniken kombiniert werden. DOM betrachtet XML-Daten als eine hierarchische Datenstruktur, bestehend aus Eltern- und Kindknoten. Dabei liegt das Interesse primär auf der Struktur und nicht auf dem Inhalt der Daten. Die XML-Klasse stellt diverse Methoden zur Verfügung wie z.B. `children():XMLList`, `length():int`, `parent():*` oder `attributes():XMLList` über die ein Zugriff auf die DOM-Strukturinformationen möglich ist.

Wenn die Struktur der Daten bereits bekannt ist, erweist es sich als deutlich effektiver, mit dem E4X-Syntax zu arbeiten.

```
trace(test.beispiel[0].codeExamples.code[0]);
```

Das vorangegangene Beispiel zeigt, wie auf den Inhalt des ersten code-Knoten innerhalb der codeExamples des ersten Beispieles aufgerufen wird. E4X bringt einen beträchtlichen Satz von Anweisungen mit sich, um XML-Daten zu filtern und zu manipulieren, auf die hier aber nicht näher eingegangen wird.

Reflexion

Es ist möglich, über die Methode `getQualifiedClassName()` den zugehörigen vollständig qualifizierten Klassennamen einer Instanz zu ermitteln. Über die Methode

[7] http://www.ecma-international.org/publications/standards/Ecma-357.htm

`getQualifiedSuperclassName()` kann entsprechend der Name der Superklasse bestimmt werden. Anhand des vollständig qualifizierten Klassennamens erhält man über die Methode `getDefinitionByName()` eine Referenz auf die Klasse. Anhand der Klassenreferenz kann ein neue Instanz der entsprechenden Klasse erzeugt werden.

```
var bsp:Beispiel = new Beispiel;
var className:String = getQualifiedClassName(bsp);
var classReference:Class = Class(getDefinitionByName(className));
var newInstance:Object = new classReference();
```

Listing 5.5: Erzeugen einer neuen Instanz per Reflexion

Über die Methode `describeType()` erhält man für das untersuchte Objekt ein XML-Objekt zurück, welches Informationen über Klasse, Superklasse, Klasseneinstellungen, Interfaces, Konstruktorsignaturen, Methodensignaturen der public Methoden und öffentliche Eigenschaften enthält.

```
<type name="beispiel::Beispiel" base="Object" isDynamic="false"
   isFinal="false" isStatic="false">
  <extendsClass type="Object"/>
  <accessor name="aString" access="readwrite" type="String"
     declaredBy="beispiel::Beispiel"/>
  <variable name="testString" type="String"/>
  <method name="aTest" declaredBy="beispiel::Beispiel" returnType=
     "void">
    <parameter index="1" type="String" optional="false"/>
  </method>
</type>
```

Listing 5.6: XML-Description der Klasse Beispiel

Über die Description können alle relevanten Informationen gewonnen werden, um dynamische Systeme aufzubauen. (vgl. Kazoun C. (1. Auflage 2008) S.93 - S.96)

Es ist möglich, eine Methode dynamisch für ein Objekt aufzurufen. Da Methoden auch nur Objekte darstellen, können sie von der Objektinstanz getrennt werden. Ist der Name einer Methode erst zur Laufzeit verfügbar, kann die Methode über die literalte Notation anhand von eckigen Klammern ([Methodenname]), extrahiert werden Adobe (a). Die extrahierte Methode ist dabei vom Typ Function. Über die Methode `call(thisArg:*, ... args):*` kann die Methode aufgerufen werden (siehe Adobe (d)).

```
public class ReflectionClass
{
    private var _counter:int;

    public function ReflectionClass()
```

```
    {
        _counter = 0;
    }

    public function get counter():int{
        return _counter;
    }

    public function countUp(number:int):void{
        _counter = _counter+number;
    }
}
```

Listing 5.7: Die Definiton der Klasse, dessen Zustand verändert werden soll. Über die Methode `countUp(number:int)` wird der counter um die entsprechende Anzahl hochgezählt.

```
var anObj:ReflectionClass = new ReflectionClass();
var f:Function = anObj["countUp"];
f.call(null,[1]);
trace(anObj.counter);
```

Listing 5.8: Dynamischer Methodenaufruf eines Objektes

Das vorangegange Beispiel zeigt, wie eine Methode über einen String von der Objektinstanz getrennt wird. Die Ausführung wird durch das Aufrufen der call()-Methode erreicht. Bei der Ausgabe von „trace(anObj.counter)“ wird auf der Konsole der Wert 1 ausgegeben.

5.3.2. Mit Daten arbeiten

Datenmodelle

Innerhalb von Flex-Anwendungen gibt es verschiedene Arten mit Daten zu arbeiten, von MXML, XML bis hin zu ActionScript-Varianten. Jede hat seine Vor- und Nachteile. Datenmodelle können verwendet werden, um z.B. die Antwort eines Remote Procedure Calls zu erfassen, die Benutzereingabe eines Formulars zu speichern oder als Hilfsmittel zum Füllen von Formulareingabeelementen nutzen.

Model-Tag Über das Tag <mx:Model> kann ein Objekt erstellt werden, welches eine Datenstruktur repräsentiert. Dabei kann die Datenstruktur entweder über das src-Attribut geladen oder selbst direkt im Editor erstellt werden. Es muss genau einen Wurzelknoten geben, das Datenmodell wird über die id angesprochen.

```
<mx:Model id="beispielData">
    <beispiel>
        <text>
            <title></title>
            <content></content>
        </text>
        <codeExample></codeExample>
    </beispiel>
</mx:Model>
```

Das vorangehende Beispiel zeigt exemplarisch, wie ein Datenmodell in MXML erstellt werden kann. Wird die Datenstruktur über das src-Attribut geladen, muss die Datei nicht mit deployed werden, da der Inhalt während der Kompilierung und nicht erst zur Laufzeit mitgeladen wird. Zusätzlich ist es möglich, ein Datenmodell zu initialisieren. Das folgende Beispiel zeigt den Inhalt eines XML-Dokumentes namens staaten.xml.

```
<states>
    <state>Hamburg</state>
    <state>Bremen</state>
    <state>NRW</state>
    <state>Bayern</state>
</states>
```

Nun kann ein Modell über diese Daten gefüllt werden, indem das src-Attribut verwendet wird.

```
<mx:Model id="staatenModell" source="staaten.xml"/>
```

Dies macht eine MXML-Datei deutlich lesbarer. Oft wird Datenbindung genutzt, um auf die Datenmodelle zuzugreifen; dieses Verfahren wird im Abschnitt „Datenbindung“ dargestellt. Es soll zunächst das <mx:Model>-Tag und dessen Umsetzung in ActionScript näher betrachtet werden. Das `<mx:Model>`-Tag besitzt keine äquivalente ActionScript-Klasse, sondern arbeitet eher wie ein dynamischer Konstruktur für die Klasse `ObjectProxy`. Der `ObjectProxy` ist als ein Wrapper für eine Objekt-Instanz zu verstehen und ermöglicht die Datenbindung für Objekt-Instanzen. Die `id` gilt dabei als Synonym für den Wurzelknoten `<beispiel>`, über sie wird das Modell bei einem Aufruf angesprochen. Die Kindknoten des Wurzelknotens sind Eigenschaften des `ObjectProxy`-Objekts und Kindknoten der Kindknoten werden zu Eigenschaften der Eigenschaften. Wenn eine Modellstruktur aus zwei oder mehr Geschwisterknoten desselben Namens besteht (wie z.B. bei staaten.xml), werden sie in ein Array konvertiert (vgl. Kazoun C. (1. Auflage 2008) S.285 - S.290).

XML Über das Tag `<mx:XML>` kann mit einem XML basierten Datenmodell gearbeitet werden. Da die Struktur des XML-Datenmodells der Struktur eines Model-Datenmodells sehr

ähnlich ist, wird hier nicht näher auf diese eingegangen. Das Laden einer Datei ist auf dieselbe Art möglich. Zum Bearbeiten und Lesen der Daten kann zum einen das veraltete XML-Object oder der E4X-Syntax verwendet werden (siehe Abschnitt 5.3.1) (vgl. Kazoun C. (1. Auflage 2008) S.290 - S.292).

ActionScript-Klassen Die beiden vorangehenden Methoden, um Datenmodelle aufzusetzen, eignen sich gut, um statische Daten einfach darzustellen. ActionScript-Klassen sollte man verwenden, wenn es um komplexe, dynamische Daten geht oder die Daten noch mit zusätzlicher Funktionalität verknüpft sind. Die Verwendung von ActionScript-Kassen bietet folgende Vorteile:

Starke Typisierung Bei der Model- und XML-Variante können keine Datentypen festgelegt werden, bei ActionScript-Klassen ist dies möglich.

Datenkonsistenz Es ist in ActionScript möglich, Werteüberprüfungen durchzuführen und entsprechend zu entscheiden, ob ein Wert verwendet, verworfen oder sogar einen Fehler auslösen soll, mit `<mx:Model>` oder `<mx:XML>` ist dies nicht möglich.

Geschäftslogik Bei einer Wertzuweisung soll es möglicherweise zu weiteren Operationen kommen, die durch die Verwendung einer ActionScript-Klasse ausgeführt werden können.

Entwurfsmuster

Eine ActionScript-Klasse kann sehr einfach erstellt werden, über die Accessors sollten die verschiedenen Eigenschaften zugänglich gemacht werden.

```
package beispiel
{
    public class DataBeispiel
    {
        private var _codeExample:String;
        private var _title:String;
        private var _content:String;

        public function set codeExample(arg:String):void{
            _codeExample = arg;
        }

        public function get codeExample():String{
            return _codeExample;
        }

        public function set title(arg:String):void{
```

```
            _title = arg;
        }

        public function get title():String{
            return _title;
        }

        public function set content(arg:String):void{
            _content = arg;
        }

        public function get content():String{
            return _content;
        }

        public function DataBeispiel()
        {

        }
    }
}
```

In MXML kann dann die ActionScript-Klasse folgendermaßen instanziiert werden:

```
<beispiel:DataBeispiel id="data" codeExample="var str:String;"
   title="anExample" content="blabla"/>
```

In Actionscript muss über den Konstruktor ein neue Instanz erstellt und anschließend müssen die verschiedenen Eigenschaften einzeln gefüllt werden (vgl. Kazoun C. (1. Auflage 2008) S.292-295).

Datenbindung

Um Datenbindungen zu verwenden, gibt es drei Möglichkeiten: die Verwendung von geschweiften Klammern {}, das <mx:Binding>-Tag und die Verwendung der BindingUtils-Klasse.

Geschweifte Klammern Über die geschweiften Klammern innerhalb eines Tag-Attributes können die Daten mit einer Ziel-Daten-Quelle verbunden werden. Intern wird eine Art Listener, der so genannte ChangeWatcher erstellt, um bei Änderungen der Ziel-Quelle die lauschende Eigenschaft ebenfalls zu ändern. Auf den ChangeWatcher wird in Abschnitt BindingUtils näher eingegangen. Da die geschweiften Klammern generell jeden beliebigen

ActionScript-Syntax ausführen, können auch XML-Datenmodelle über E4X-Ausdrücke angesprochen werden.

```
<mx:Text id="output" text="{input.text}"/>
<mx:TextInput id="input"/>
```

Das vorangegangene Beispiel erläutert die Verwendung der geschweiften Klammern zur Datenbindung. Das Textfeld enthält jeweils den Text, der in das Texteingabefeld geschrieben wird (vgl. Kazoun C. (1. Auflage 2008) S. 295 - S.299).

<mx:Binding> Das Tag erzeugt dasselbe Resultat wie die Verwendung der geschweiften Klammern. Allerdings werden die Komponenten sauberer voneinander getrennt. Das folgende Beispiel stellt das Äquivalent zu der Verwendung von geschweiften Klammern dar (vgl. Kazoun C. (1. Auflage 2008) S.299 - S.302).

```
<mx:Binding destination="output.text" source="input.text"/>
<mx:Text id="output"/>
<mx:TextInput id="input"/>
```

BindingUtils Soll zur Laufzeit die Datenbindung dynamisch verändert oder erstellt werden, muss die Klasse BindingUtils verwendet werden. Sie stellt die statische Methode `bindProperty(site:Object, prop:String, host:Object, chain:Object , commitOnly:Boolean = false):ChangeWatcher` zur Verfügung über die eine Datenbindung zwischen zwei Eigenschaften erstellt werden kann. Das folgende Beispiel zeigt, wie über die Klasse `BindingUtils` eine zu `<mx:Binding>` und den geschweiften Klammern äquivalente Datenbindung erreicht werden kann.

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" layout="
   absolute" initialize="initHandler(event)">
<mx:Script>
    <![CDATA[
        import mx.binding.utils.BindingUtils;
        private function initHandler(event:Event):void{
            BindingUtils.bindProperty(output,"text",input,"text");
        }
    ]]>
</mx:Script>
<mx:VBox>
    <mx:Text id="output"/>
    <mx:TextInput id="input"/>
</mx:VBox>
</mx:Application>
```

Wie schon aus der Methodensignatur von `bindProperty()` zu erkennen ist, wird ein Objekt der Klasse `ChangeWatcher` zurückgegeben, welches die Datenbindung repräsentiert. Soll nun die Datenbindung zwischen zwei Eigenschaften beendet, eine neue Quelle angeben oder der derzeitige aktuelle Wert der Quelle gelesen werden kann dies über das `ChangeWatcher`-Objekte geregelt werden (vgl. Kazoun C. (1. Auflage 2008) S.302 - S.306).

Datenbindung benutzerdefinierter Klassen Die Datenbindung muss für eigene Klassen erst über das Meta-Tag `[Bindable]` aktiviert werden. `[Bindable]` kann dabei alle öffentlichen Eigenschaften sowie Accessors einer Klasse, eine Eigenschaft, ein Accessor-Paar oder auch nur einen einzelnen Accessor datenbindungsfähig machen. Existiert nur ein Accessor, z.B. eine Getter-Methode, kann diese entsprechend nur als Quelle fungieren. Über das `[Bindable]`-Tag wird der Compiler angewiesen die nötigen Konfigurationen vorzunehmen, um die gewünschten Elemente datenbindungsfähig zu machen. Wird eine mit einem `[Bindable]`-Tag versehene ActionScript-Klasse aus einem MXML-Dokument heraus instanziiert, muss bei der Variablendeklaration ebenso das `[Bindable]`-Tag verwendet werden. Kann ein Objekt aus irgendeinem Grund die Datenbindung nicht selbst unterstützen, sollte ein Proxy-Objekt als Wrapper verwendet werden, um die Dienste der Datenbindung für diese Objekt zu ermöglichen (vgl. Kazoun C. (1. Auflage 2008) S.314) (vgl. Kazoun C. (1. Auflage 2008) S.306 - S.308).

Datenbindung anpassen Eine Datenbindung wird intern über das Ereignis-Modell von ActionScript realisiert (siehe Abschnitt 5.3.1). Bei der Verwendung des `[Bindable]`-Tags werden Änderungen anhand des propertyChange-Ereignisses mitgeteilt. Da es Fälle gibt, in denen es sich als ineffizient erweist, jede Änderung einer Eigenschaft einer Klasse mitzuteilen, ist es möglich, benutzerdefinierte Ereignisse auszulösen. Der Name des Ereignisses kann über das `[Bindable]`-Tag hinzugefügt werden:

```
[Bindable(event="customEvent")]
```

Nun muss über `dispatchEvent(new Event("customEvent"))` das Ereignis manuell ausgelöst werden. Zusätzlich muss das [`Bindable`]-Tag vor jeder Eigenschaft oder jedem Accessor eingefügt werden (vgl. Kazoun C. (1. Auflage 2008) S. 308 - S.309).

5.3.3. Remote-Kommunikation

Innerhalb von Flex können zwei für diese Arbeit relevante Remote-Datenkommunikationsverfahren unterschieden werden:

1. HTTP-Request-Response-Kommunikation
2. Echtzeitkommunikation

HTTP-Request-Response-Kommunikation

Die Request-Response-Verfahren lassen sich in drei verschiedene Verfahren unterteilen: Die Verwendung von einfachen HTTP-Diensten, Web-Services und Flash Remoting. Alle Verfahren funktionieren dabei grundsätzlich gleich, sie stellen eine Anfrage und erhalten darauf eine Antwort. Das am besten zu nutzende Verfahren hängt dabei von der Art des Dienstes ab, den ein Server zur Verfügung stellt (vgl. Kazoun C. (1. Auflage 2008) S.420).

Einfache HTTP-Dienste Ein einfacher HTTP-Dienst wird dazu genutzt, um z.B. Text oder XML-Ressourcen einzulesen, aber auch, um Skripte auf Seiten auszuführen die eine bestimmte serverseitige Funktionalität auslösen oder bestimmte Daten zurückliefern. Um HTTP-Dienste aufzurufen, kann die `HTTPService`-Klasse des Flex Frameworks oder die Flash Player-Klasse `flash.net.URLLoader` genutzt werden (vgl. Kazoun C. (1. Auflage 2008) S.420). Da die Verwendung der Klasse `flash.net.URLLoader` deutlich aufwändiger ist als die der `HTTPService`-Klasse und keine weiteren Vorteile bringt, wird sie hier nicht weiter behandelt.

HTTPService Die Klasse `HTTPService` kann wie gewohnt per MXML oder ActionScript verwendet werden. In MXML wird über das `<mx:HTTPService>`-Tag eine `HTTPService`-Instanz erstellt. Die `send()`-Methode der `HTTPService`-Instanz stellt die Anfrage. Sollen Parameter bei der Anfrage übergeben werden, werden diese als Eigenschaften eines `Object`-Objektes definiert (Die Verwendung der `Object`-Klasse wird in Abschnitt 5.3.1 erläutert) und der `request`-Eigenschaft des `HTTPService`-Objektes zugeordnet. Der Name und der Wert der Eigenschaft stellen dabei den Namen und den Wert des Parameters dar. Das Format, in dem die Parameter übermittelt werden, kann über die Eigenschaft `contentType` bestimmt werden. Es stehen die Optionen `application/x-www-form-urlencoded`, um die Daten in dem URL-encoded-Format zu übermitteln und `application/xml`, damit sie als reines XML gesendet werden, zur Verfügung. Über die Eigenschaft `method` wird die Transportmethode festgelegt. `GET`, `POST`, `HEAD`, `OPTIONS`, `PUT`, `TRACE` oder `DELETE` sind möglich.

Über das result-Attribut kann ein Handler für das Antwortereignis definiert werden. Das Ergebnis der Anfrage wird in der Eigenschaft lastResult des HTTPService-Objektes gespeichert und ist vom Typ Object. Falls es anhand der Daten möglich ist, werden sie auf dieselbe

Art wie die Daten eines Model-Tags deserialisiert, ansonsten wird die Antwort als Text interpretiert. Über die Eigenschaft resultFormat des HTTPServices-Objektes können folgende Antwortformate gewählt werden:

text Die Antwortdaten werden nicht geparst, sondern als Text behandelt

flashvars Es wird angenommen, dass die Antwortdaten im URL-encoded-Format vorliegen, sie werden als Objekt mit Eigenschaften geparst, welche mit den Name/Werte-Paaren übereinstimmen.

array Die Daten werden als XML interpretiert und in Objects geparst. Sollten sich die Antwortdaten nicht in ein Array parsen lassen, werden sie in einem untergebracht, das Ergebnis ist also immer ein Array.

xml Die Daten werden als XML interpretiert, wobei die ActionScript-Klasse XMLNode verwendet wird.

e4x genauso wie xml, nur wird die ActionScript-Klasse XML verwendet, welche den E4X-Standard unterstützt.

(vgl. Kazoun C. (1. Auflage 2008) S.421 - S.429)

Webservices Das Flex Framework stellt eine Unterstützung für SOAP-Webservices zur Verfügung, der Flash Player unterstützt diesen Kommunikationsdienst nativ nicht. Die Webservice-Komponente kann wie üblich über MXML oder ActionScript genutzt werden. Für einen Webservice muss der Standort seiner Web Services Description Language(WSDL)-Definition festgelegt werden. Die WSDL beschreibt die Funktionen, Daten, Datentypen und Kommunikationsprotokolle des Webservices. In der ActionScript-Variante kann nun einfach die WSDL geladen und die Operationen angesprochen werden. In MXML müssen die Operationen als Tags selbst definiert werden, damit der Entwickler auf sie zugreifen kann. Das Ergebnis kann über einen EventHandler oder per Datenbindung über die lastResult-Eigenschaft (siehe Abschnitt HTTPService) genutzt werden. Wenn Parameter für eine Methode angegeben werden sollen, kann dies in MXML direkt im Operations-Tag geschehen, oder auch wie in Actionscript als ein ganz normaler Methodenaufruf realisiert werden.

Ein Vorteil bei der Nutzung von Webservices ist, dass in der WSDL der Rückgabetyp für ein Ergebnis definiert werden kann. Flex konvertiert dabei die Ergebnisse automatisch in den passenden ActionScript-Typ (vgl. Kazoun C. (1. Auflage 2008) S.434 - S.443).

Flash Remoting Flash Remoting ist eine weitere Technologie, welche auf der Request-Response-Kommunikation anhand von HTTP basiert. Die Unterschiede zu den allgemeinen Webservices sind folgende:

- Datenserialisierung und -deserialisierung werden bei Anfragen und Ergebnissen automatisch durchgeführt.
- Nicht nur standard- sondern auch benutzerdefinierte Datentypen können automatisch konvertiert werden.
- als Übertragungsformat wird nicht XML sondern, ein binäres, proprietäres Format namens Action Message Format (AMF) verwendet.
- AMF wird vom Flash Player nativ unterstützt, dadurch wird es automatisch und deutlich schneller verarbeitet
- ein Gateway auf dem Server empfängt die AMF-Nachrichten und delegiert sie an den entsprechenden Dienst weiter. Der eigentliche Dienst des Servers kann dabei eine Standardklasse sein und muss, wenn überhaupt, nur wenig angepasst werden. Ein enge Kopplung wird dadurch vermieden.

FlashRemoting wird vom Flash Player selbst direkt unterstützt, es müssen bei der Verwendung keine speziellen Bibliotheken eingebunden werden. Der Server muss dagegen das Gateway bereitstellen, welches die Verarbeitung von AMF-Paketen über HTTP organisiert.

Flash Remoting kann im Gegensatz zu den Webservices nur in ActionScript implementiert werden. Ansonsten unterscheidet sich der Aufruf des Flash Remoting Gateways nur minimal von der eines Webservices und wird daher nicht näher behandelt (vgl. Kazoun C. (1. Auflage 2008) S.443 - S.446).

Echtzeitkommunikation

Über den Flash Player werden zwei Typen von persistenten Socket-Verbindungen nativ unterstützt. Bei der dritten Variante handelt es sich um ein proprietäres Protokoll von Adobe. Alle vorgestellten Verfahren können TCP als Übertragungsprotokoll verwenden.

XML-Sockets Über XML-Sockets kann eine persistente-Socketverbindung zwischen einem Client und einem Server hergestellt werden. Ein XML-Socket erlaubt nur den Austausch von XML-Daten. XML-Sockets werden im Bereich von ActionScript oft dazu verwendet, um Chat oder Multiplayer-Anwendungen zu realisieren, da der Server von sich aus Daten an Client schicken kann, ohne auf einen clientseitigen Aufruf warten zu müssen. (vgl. Adobe (c))

Binärsockets Binärsockets unterscheiden sich von XML-Sockets in dem Sinne, dass sie nicht XML-Pakete austauschen müssen. Die Informationen können binär übertragen werden. Dies ermöglicht es, verschiedene Dienste, z.B. Mail-Server über POP3, SMTP oder IMAP, anzusprechen (vgl. Adobe (c)).

RTMP Das sogenannte Real Time Messaging Protocol(RTMP) wird zum einen für Medien und zum anderen für die Echtzeit-Datenkommunikation genutzt. Der Schwerpunkt liegt allerdings auf Audio- und Videostreaming. Bei RTMP handelt es sich um ein proprietäres Protokoll und wird bisher ausschließlich von Adobe-Servern wie z.B. dem Flash Media Server bereitgestellt (vgl. Kazoun C. (1. Auflage 2008) S.446).

Es existiert zwar ein Open Source Flash Server namens Red5 (Red5), dieser befindet sich allerdings noch im Beta-Stadium. Da RTMP an Adobes eigene kommerzielle Server gebunden ist und über den Flash Player auch nicht nativ unterstützt wird, ist es für die allgemeine Entwicklung nicht sinnvoll.

5.4. Entwicklung mit Flex

Da der Nutzen eines Frameworks nicht allein von den dessen Konzepten sowie den Eigenschaften der genutzten Programmiersprachen abhängt, sollen in diesem Kapitel die Entwicklungsumgebung und die Möglichkeiten im Bereich Debugging, Logging und Unit-Testing beschrieben werden.

5.4.1. Entwicklungsumgebung

Die einzige zurzeit verfügbare Entwicklungsumgebung mit einer vollständigen Unterstützung für Flex ist Adobes Flex Builder[8]. Der Flex Builder ist ein Plug-in für die bekannte Entwicklungsumgebung Eclipse[9]. Im folgenden Abschnitt werden die verschiedenen Fenster und Ansichten vorgestellt. Anschließend wird beschrieben, welche Features der Flex Builder für einen Entwickler bietet, der die Funktionen einer Java-Eclipse-Version gewohnt ist.

Das Aussehen des Flex Builders spiegelt das Aussehen der Java-Eclipse-Version wider. Für das Erstellen von Elementen, z.B. einer Klasse, stehen wie gewohnt Wizards zur Verfügung, die den Entwickler unterstützen. Im Falle der Klasse bietet der Flex Builder ein Fenster an, über welches der Entwickler alle relevanten Einstellungen vornehmen kann (siehe Abbildung 5.1).

[8] http://www.adobe.com/products/flex/features/flex_builder/
[9] http://www.eclipse.org/

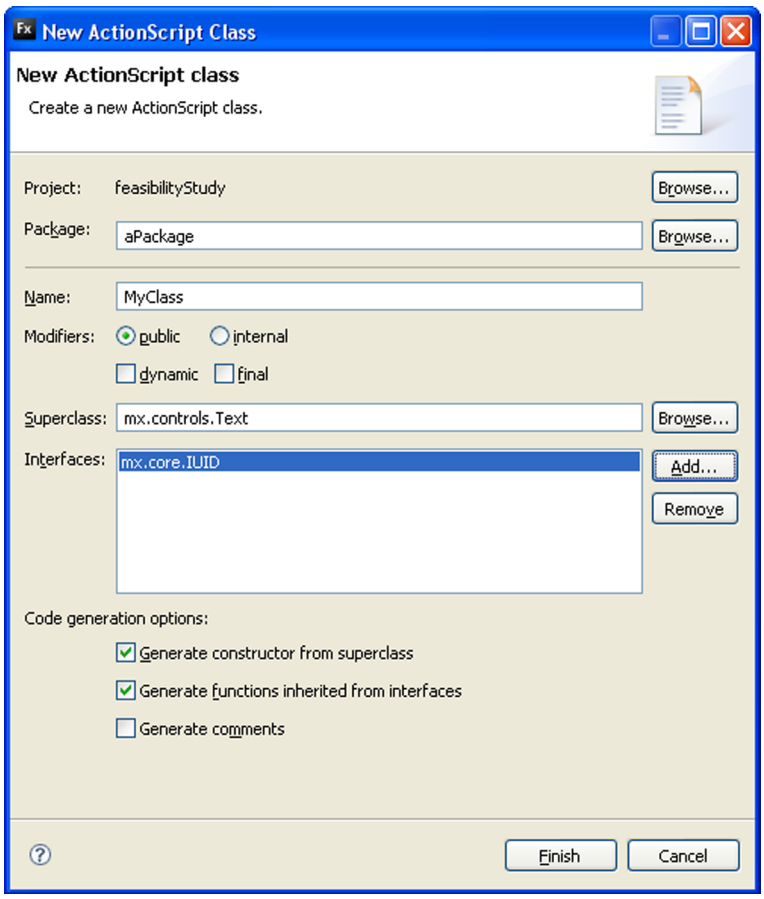

Abbildung 5.1.: Flex Builder: Klassenerstellung

Das Console- und das Problemfenster werden in Flex Builder analog verwendet. Das gewohnte Fenster „Outline“ kann innerhalb der Codeansicht zwischen einer MXML und einer Klassensicht wechseln, der Aufbau und das Aussehen ist vergleichbar mit der Java-Version. Abbildung 5.2 verdeutlicht, wie ähnlich das Aussehen der beiden Editor-Versionen ist.

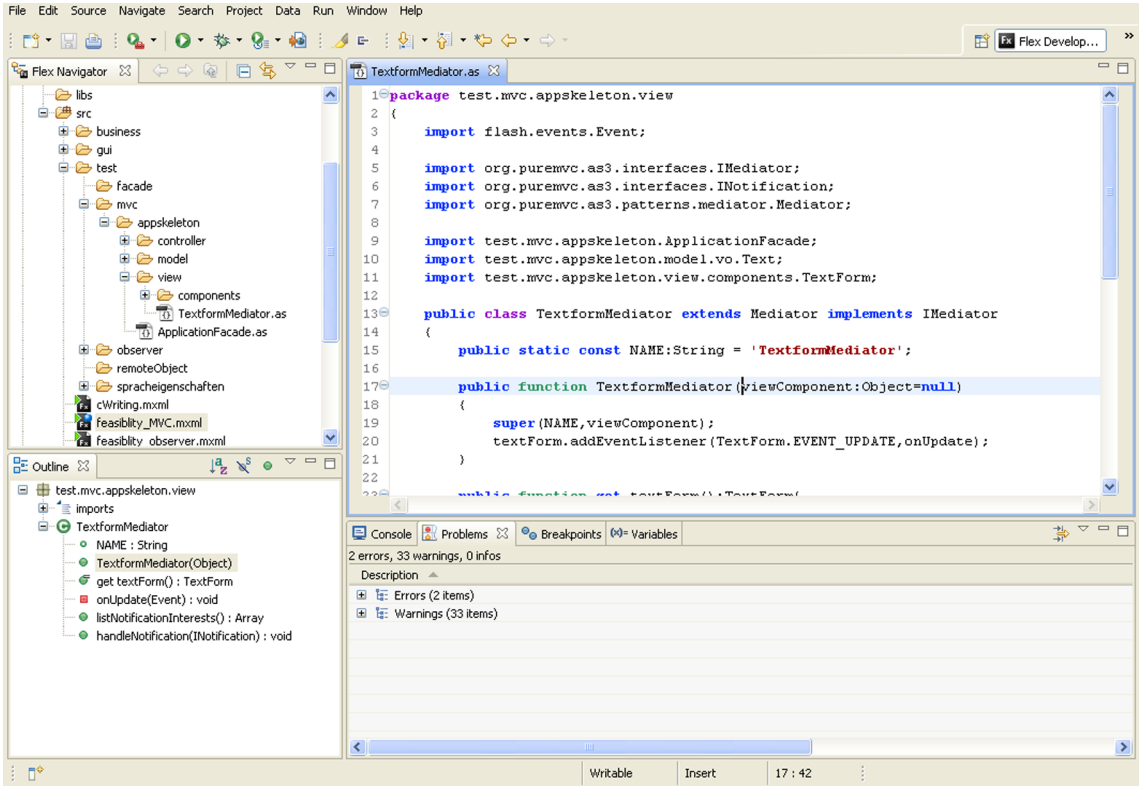

Abbildung 5.2.: Flex Builder: Codeview

Für die Designansicht gibt es verschiedene Flex spezifische Fenster, welche den Entwicklern dabei unterstützen, schnell ein grafisches Layout zu entwerfen und anzupassen:

Components-Fenster zeigt Komponenten an, welche der Entwickler per Drag’n’Drop direkt auf die Oberfläche ziehen kann

Flex-Properties-Fenster zeigt hauptsächlich Layout- sowie die meist verwendeten Property-Eigenschaften einer ausgewählten Komponente an

States-Fenster zeigt die Anwendungszustände und deren Struktur an

Einem Java-Entwickler stehen bei der Verwendung von Eclipse eine Palette von Unterstützungen zur Seite, die wichtigsten sollen hier kurz aufgezählt werden:

- contentAssist, mit Doku-Fenster

- Refactoring, z.B. Umbenennen oder Verschieben von Klassen
- Code-Generation, z.B. Getter- und Settermethoden
- Surround With, z.B. try/catch-Block

Der Flex Builder weist dabei nur die Unterstützung des contentAssists auf, wobei allerdings kein Fenster mit einer entsprechenden Dokumentation erscheint und die Unterstützung des Refactoring auch nur in dem Falle, dass eine Klasse oder Methode umbenannt werden soll.

Abschließend betrachtet, stellt der Flex Builder eine solide Grundlage für die Entwicklung von Flex dar. Es gibt allerdings auch viele Stellen, bei denen noch nicht der gewohnte Grad an Unterstützung bereitgestellt wird.

5.4.2. Debugging

Der Flash Debug Player stellt für Entwickler die z.B. aus dem Javabereich gewohnten Funktionen zur Verfügung. Dabei ist es unter anderem möglich, Unterbrechungspunkte zu setzen, Code schrittweise auszuführen oder Variablen zur Laufzeit zu ändern. Der Debug Player ist dabei in zwei Versionen verfügbar, als Browser-Plugin und als Standalone-Variante. Er stellt verschiedene Funktionalitäten bereit, welche Zugriff auf das Laufzeitverhalten einer laufenden Anwendung ermöglichen, und wird z.B. von dem Flex Builder-Debugger oder dem Kommandozeilen-Debugger verwendet. Die Ansicht des Flex Builder-Debuggers richtet sich dabei an das entsprechende Java-Pendant der Eclipse-Umgebung.

Startet man eine Flex-Anwendung, werden Laufzeitfehler dabei nur angezeigt, wenn ein Player mit Debugger installiert ist. Bei einem Fehler erscheint ein neues Fenster mit entsprechenden Informationen. Ist kein Player mit Debugger vorhanden, wird der Fehler nicht angezeigt und der Player arbeitet stumm weiter.

Da es im Bereich von grafikintensiven Anwendungen leicht zu Engpässen bei der Zeichenperformanz kommen kann und deren Bestimmung sich als schwierig erweist, ist es möglich, über eine Option des Flash Debug Players die Bereiche hervorzuheben, welche neu gezeichnet worden sind. Die Option kann über das Kontextmenü des Flash Debug Players ausgewählt werden (vgl. Kazoun C. (1. Auflage 2008). S.450-454)

5.4.3. Logging

Der Flash Player unterstützt einfaches Logging anhand der globalen Methode `trace()`. Sie kann von überall aufgerufen werden und nimmt als Parameter eine beliebige Anzahl von

Strings an. Der Einsatz der trace-Methode erfordert den Flash Debug Player, welcher auch über den Flex Builder-Debugger genutzt wird. Die Loggingnachrichten werden dann innerhalb des Debug-Modus des Flex Builders in der Konsole ausgegeben. Der Flash Debug Player ermöglicht es auch anderen Benutzern die trace-Mitteilungen in eine Datei umzuleiten, dies könnte dabei helfen, einen Fehler einfacher zu isolieren.

Neben der trace-Methode wird in Flex ein Logging-Framework zur Verfügung gestellt. Anhand des Frameworks ist es möglich, voreingestellte oder auch eigene Ziele für Logs zu definieren und zusätzliche Informationen wie z.B. Kategorie oder unterschiedliche Stufen für eine Lognachricht festzulegen. Ein Ziel könnte dabei z.B. die bereits vorgestellt trace-Methode oder die Ausgabe in eine Datei sein.

Für die Komponenten WebService, HTTPService und RemoteObject wird das Logging-Framework bereits intern verwendet, um bei einem Fehler nützliche Informationen wie z.B. Daten die gesendet oder empfangen wurden, bereitzustellen (vgl. Kazoun C. (1. Auflage 2008) S.464 - S.469).

5.4.4. Unit-Testing

Bei der Entwicklung von Software gilt es weiterhin als sehr schwer, die Betriebssicherheit zu garantieren und diese in den Entwicklungsprozess mit einzubeziehen. Das sogenannte Test-driven Development (TDD) verfolgt den Ansatz, zuerst automatisierte Unit-Tests und anschließend den eigentlichen Code zu schreiben. Die beiden Phasen wechseln sich dabei schnell und häufig ab. Der Entwickler soll sich auf die Schnittstellen und das Verhalten einer Anwendung konzentrieren, wobei die Anwendung nach und nach wächst (vgl. Janzen (2005)).

Da bisher viele Projekte im traditionellen Flashbereich eher kleiner ausgefallen sind, hat sich das Unit-Testing nicht besonders durchgesetzt. Mit der Einführung von Flex ergeben sich nun andere Anforderungen an die Überprüfbarkeit der Funktionalität einer Anwendung, da Flex eher in Enterprise-Umgebungen angesiedelt wird. Es ist fraglich, ob der Ansatz des Test-driven Development vollständig verfolgt werden muss, allerdings gilt der Nachweis der Betriebssicherheit einer Komponente anhand von Unit-Tests als ein Qualitätsmerkmal. Dieser Nachweis ist besonders nützlich, wenn eine Anwendung über einen längeren Zeitraum Veränderungen erfährt, da über die bisher geschriebenen Unit-Tests die gewünschte ursprüngliche Funktionalität überprüft werden kann.

Die FlexUnit[10] orientiert sich an der weit verbreiteten JUnit[11], ein Unit-Testing Framework für

[10] http://code.google.com/p/as3flexunitlib/
[11] http://www.junit.org/

Java, und erfüllt analog die Anforderungen an Unit-Testing für Flex. FlexUnit wird von Adobe selbst empfohlen und unterstützt (Adob).

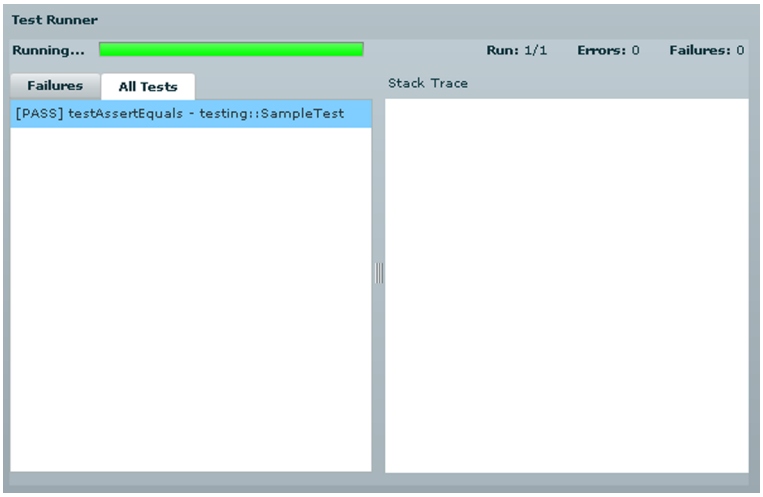

Abbildung 5.3.: FlexUnit

6. Machbarkeitsstudie

Die Machbarkeitstests sind analog zu den Anforderungen aus Abschnitt 4.10, am Ende des Design-Kapitels aufgebaut. Zuerst werden die Architektur, dann die Entwurfsmuster und danach die Sprachanforderungen untersucht. Abschließend erfolgt ein bewertendes Fazit, welches die Gesamtheit der Machbarkeitsstudie betrachtet.

6.1. Architekturanforderungen

6.1.1. Schichten

Es ist innerhalb Flex möglich, eine Trennung von Client und Server vorzunehmen. Die Clientschichtung kann dabei wie im Design beschrieben (siehe 4.8.1) vorgenommen werden. Die verschiedenen ActionScript-Klassen können Komponenten, welche zu bestimmten Schichten gehören, zugeordnet werden. Verdeutlicht wird dies bei dem Machbarkeitstest des MVC-Architekturmusters.

6.1.2. MVC

Um die Anforderungen des Model-View-Controller-Musters zu überprüfen, ist es sinnvoll, sich zuerst einen Überblick über die zur Verfügung stehende MVC-Frameworks für Flex machen und ein Framework für den Machbarkeitstest auszuwählen. Abschließend soll die Tauglichkeit in einem Fazit bewertet werden.

Im Bereich der MVC-Frameworks für Flex, gibt es zwei Konkurrenten, PureMVC[1] und Cairngorm[2]. Beide Frameworks sind zur Zeit viel diskutiert. Cairngorm wurde von Adobe Labs entwickelt, stellt das mit Abstand vebreiteste MVC-Framework dar und ist Open Source.
PureMVC wurde von Futurescale produziert und läuft unter der Creative Commons Attribution Lizenz 3.0[3].

[1] http://www.puremvc.org/
[2] http://labs.adobe.com/wiki/index.php/Cairngorm
[3] http://creativecommons.org/licenses/by/3.0/

Bevor auf die Unterschiede eingegangen wird, sollten zuerst die Gemeinsamkeiten vorgestellt werden. Beide schreiben Normen vor, welche es ermöglichen, komplexe Anwendungen schneller zu erstellen und zudem leichter wart- und erweiterbar zu gestalten. Beide Frameworks ermöglichen es, verschiedene Komponenten wiederzuverwenden. Sie geben eine vollständige Kontrolle über den Ereignisablauf und integrieren ein eigenes unabhängiges Ereignisverfahren. Die Tabelle 6.1 stellt die Kernunterschiede der beiden Frameworks dar (vgl. Robson).

	PureMVC	Cairngorm
+	Vollständige Trennung graphischer Komponenten von Model und Controller, daher höher Wiederverwendungsgrad	auf Flex zugeschnitten und arbeitet mit Flex-nativen Funktionalitäten wie z.B. Databinding
+	flexibles Model, da über Commands und Macrocommands Arbeitsabläufe erstellt werden können und Teile von Arbeitsabläufen leicht wiederverwendbar sind	meist verbreitete Framework, wird von Adobe gestützt
-	PureMVC bindet diverse Features wie z.B. Datenbindung nicht in sein Konzept mit ein	Der ModelLocator kann in großen Anwendungen unhandlich werden. Er erlaubt es einer grafischen Komponente, ihr zugehöriges Modell zu finden
-		große Masse von Elementen erschweren die Übersichtlichkeit

Tabelle 6.1.: Unterschiede von PureMVC und Cairngorm,Quelle: (vgl. Robson)
Ein Vorteil wird durch „+“ und ein Nachteil anhand von „-“ dargestellt

In einem, innerhalb der Flex Community, stark beachteten Vortrag (Bayes L.) von Luke Bayes und Ali Mills, der sich mit der Evaluation von Flex Application-Frameworks beschäftigt, wird der interne Aufbau der beiden Frameworks näher betrachtet.

Als ein sehr großer Nachteil von Cairngorm, wird die Art und Weise angegeben, wie Singletons (siehe Gamma u. a. (2001)) verwendet werden. Die Singletons werden direkt, innerhalb der Applikation, vom Entwickler verwendet, um die gewünschte Komponente anzusprechen. Durch den Einsatz von Singletons werden Module stark aneinander gekoppelt. Zudem machen sie das Testen von Anwendungen sehr schwer, da über Singletons meistens eine Art von Anwendungszustand festgehalten wird. Durch lose gekoppelte Module können Tests unabhängig von deren Reihenfolge durchgeführt werden, bei der Verwendung eines Singletons könnte es dabei zu unterschiedlichen Verhaltensweisen führen. Zusätzlich müssen beim Testen für verschiedene Module oft Mocks implementiert werden. Wird das Konzept der Single-

tons sehr zentral eingesetzt wie in Cairngorm, muss für das Testen kleiner Module zwangsweise bereits ein großer Teil der Application über Mocks verfügbar gemacht werden.

PureMVC kann hingegen auf eine sehr saubere Implementierung zurückgreifen, da intern strikt nur auf Interfaces gearbeitet wird. Sollte zu einem Zeitpunkt ein Fehler auftreten oder ein anderes Verhalten gewünscht sein, ist es leicht, eine eigene Implementierung zu integrieren. Innerhalb von PureMVC kommen deutlich weniger Singletons zum Einsatz, zudem wird die Verwendung der Singletons über eine Fassade vor dem Entwickler versteckt. Singletons werden nie direkt angesprochen. Gegenwärtig ist bei PureMVC eine Multicore-Version des Frameworks in Entwicklung, welche es erlaubt, auch mehrere PureMVC-Module gleichzeitig einzusetzen.

Da PureMVC eine deutlich gewissenhaftere Trennung von Verantwortlichkeiten propagiert und zusätzlich eine vorteilhaftere interne Struktur vorweisen kann, wird es zur Durchführung des Tests ausgewählt.

PureMVC

Die aus dem MVC-Muster bekannten Rollen, Model, View und Controller, sind nicht mit den in PureMVC verwendeten Klassennamen gleichzusetzen. Folgende Definitonen entsprechen den ursprünglichen Rollen (siehe Websector):

Model Proxies

View Mediatoren und ihre ViewComponents

Commands Controller

Die Klassen `Facade`, `View`, `Model` und `Controller`, sind im PureMVC-Framework als Singletons realisiert. `Model` hält Referenzen auf die Proxies, `View` hält Referenzen auf die Mediators und `Controller` besitzt die gemappten Command-Klassen, welche keinen Zustand besitzen und nur instanziiert werden, falls dies notwendig ist. Das Objekt `Facade` initialisiert die Singletons `Model`, `View` und `Controller`. Weiterhin versteckt die Klasse `Facade` die Dienste der drei Objekte. Abbildung 6.1 verdeutlicht die interne Struktur von PurMVC.

Bei PureMVC wird nicht auf die ActionScript eigene Implementierung von Events zurückgegriffen, sondern ein eigener äquivalenter Mechanismus namens Notifications eingesetzt. Dadurch ist das PureMVC-Modell unabhängig, von der Ereignisimplementierung des Flash Players (siehe Dehats , Kommentar von Cliff Hall).

Einen ausführlichen Überblick, anhand von UML-Diagrammen, ist unter PureMVC (b) zu finden.

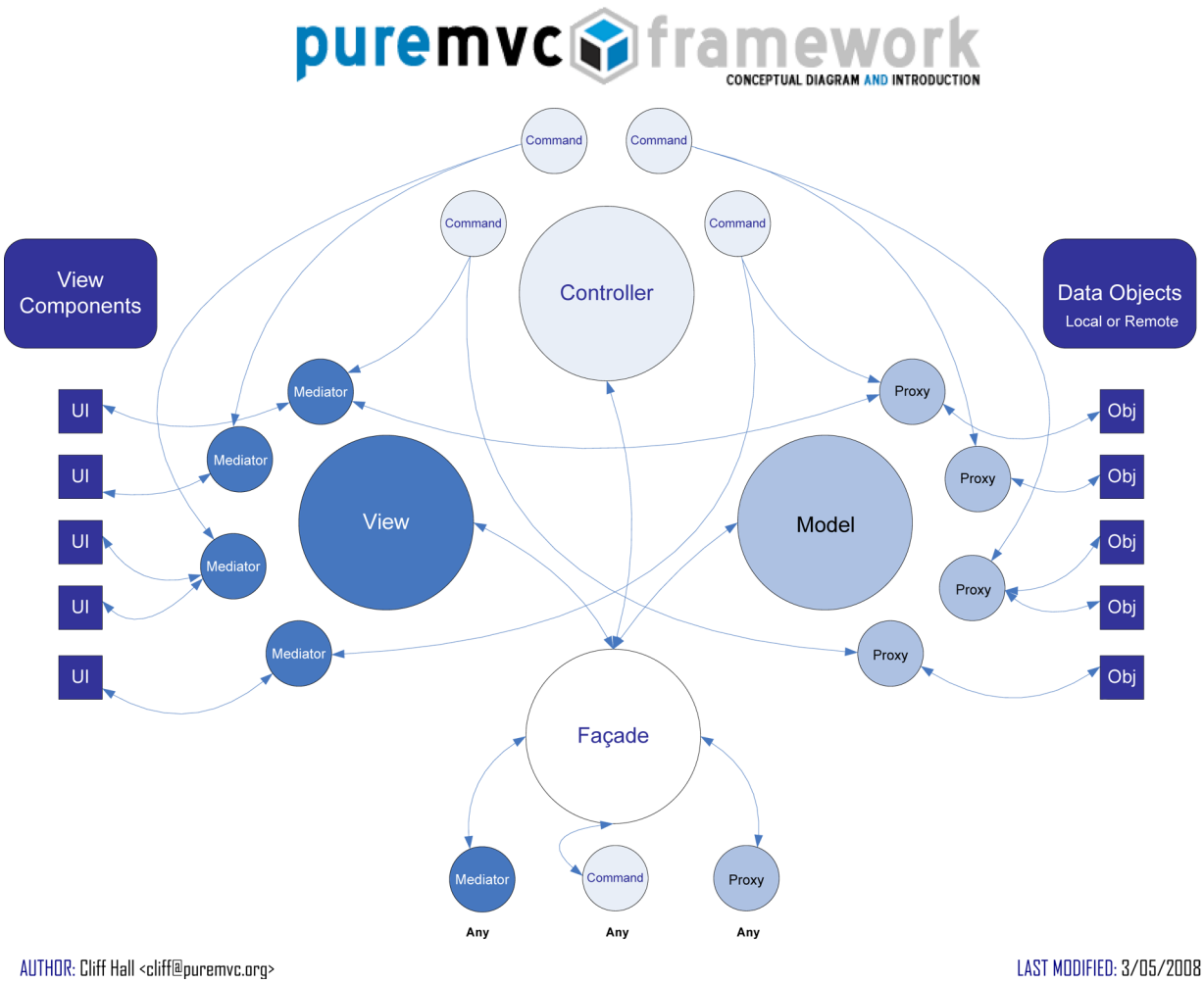

Abbildung 6.1.: Konzeptioneller Aufbau, (Quelle: PureMVC (a))

Beispiel anhand PureMVC

Das Beispiel soll die Umsetzung des MVC-Architekturmusters in Flex anhand des PureMVC-Frameworks untersuchen. Zuerst wird die Funktionsweise der Beispielanwendung erläutert und anschließend werden die internen Abläufe dargestellt.

In einem Fenster ist ein Textfeld, ein leeres Feld und ein Updateknopf zu sehen. Wenn der Benutzer Zeichen in das Textfeld eingibt und auf den Updateknopf drückt, wird die Eingabe in das leere Feld übertragen und der Inhalt des Textfeldes gelöscht.

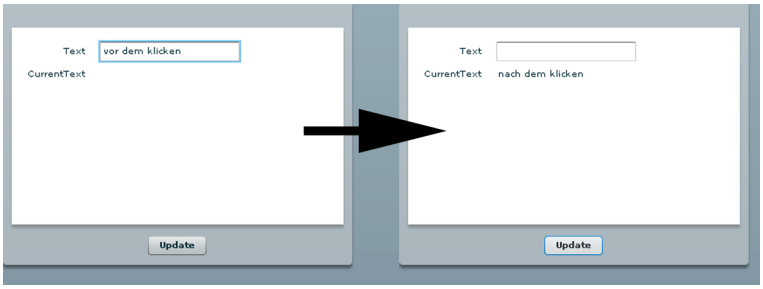

Abbildung 6.2.: MVC-Beispiel

Folgende Voraussetzungen sind für das Beispiel gegeben:

- Die Eingabe des Textfeldes ist über Databinding an ein `ValueObject` des Typs `Text` gebunden, dieses repräsentiert den Zustand des `TextForms`.
- Der `TextFormMediator` ist als `EventListener` des `TextForms`, für das Ereignis `EVENT_UPDATE` eingetragen.
- Der `TextFormMediator` wartet auf `Notifications` des Typs `UPDATE_TEXTVIEW` .
- Das `UpdateTextCommand` wurde während der Initialisierung der Anwendungsfassade für das `UPDATE_TEXT`-Ereignis registriert.
- Der `TextProxy` hält den aktuellen Zustand des `ValueObjects Text`. Hinweis: Dieses Objekt ist nicht mit dem `ValueObject` des Textfeldes identisch!

Wenn nun der Updateknopf gedrückt wird, ist der Ablauf wie folgt:

1. Drücken des Knopfes löst ein Ereignis des Typs `EVENT_UPDATE` aus.
2. `TextFormMediator` hat sich als Listener für dieses Ereignis registriert und schickt selbst eine `Notification` des Typs `UPDATE_TEXT`. Das `ValueObject editedText` des Textfeldes wird dabei als Argument mitgeschickt.

3. `UpdateTextCommand` aktualisiert die `ValueObject` Eigenschaft des `TextProxy`, durch das `ValueObject editedText`.
4. Der `TextProxy` schickt wegen der Änderung eine `Notification` des Typs `UPDATE_TEXTVIEW` mit dem aktualisierten `ValueObject` heraus.
5. Der `TextFormMediator` erhält die `Notification` und ersetzt das `ValueObject` des `TextForms`.

Fazit

Die Umsetzung des MVC-Architekturmusters ist in Flex mithilfe des PureMVC-Frameworks sehr klar und einfach strukturiert. MXML kann dabei gut zur Gestaltung der einzelnen GUI-Komponenten verwendet werden, die Trennung der Funktionalität einer GUI-Komponente in einen Mediator ist sinnvoll, da die Komponente dadurch wiederverwendbar wird. Ein Entwickler findet die gewohnten Muster und Verantwortlichkeiten des MVC-Musters wieder und kann sich entsprechend schnell auf die eigentliche Anwendung konzentrieren.

6.1.3. Broker

In Flex gibt es keinen äquivalenten Mechanismus zu einem Object-Broker, der auf gleiche Weise die Kommunikation gewährleistet. Nun stellt sich die Frage, welche der aus Abschnitt (5.3.3) vorgestellten Kommunikationsarten auf welche Weise verwendet werden kann, um ein möglichst ähnliches Verhalten zu erreichen. Zusätzlich wird ein alternativer Kommunikationsansatz aus dem Designkapitel aufgegriffen. Abschließend erfolgt ein bewertendes Fazit.

Der Bereich der Remotekommunikation lässt sich grob in HTTP-Request-Response-Kommunikation und Echtzeitkommunikation aufteilen. Die Kommunikationsdienste des HTTP-Bereiches erfordern immer eine vorangehende Anfrage an den Server, dieses Verhalten ist nicht erwünscht(siehe Abschnitt 4.4.2 auf Seite 46).

Innerhalb der Echtzeikommunikation stehen nun drei Alternativen zur Verfügung: XML-Sockets, Binärsockets und RTMP (siehe Abschnitt 5.3.3, S. 98). Bei diesen drei Varianten ist es möglich, auch vom Server aus ohne eine vorherige Anfrage des Clients Nachrichten an den Client zu schicken. Setzt man nun auf Binärsockets, ist es nötig, eine Art von Standard festzulegen, ein Protokoll, welches den Ablauf und den Inhalt der Kommunikation beschreibt. Dies ist eine zusätzliche Anforderung an die Entwicklung einer Anwendung, die bei der Verwendung eines Broker nicht anfällt.

Bei der Nutzung von XML-Sockets muss eine zusätzliche Schicht eingeführt werden, ein Adapter, welcher die Anfragen in ein XML-Format umwandelt und zudem die Ergebnisse aus XML wieder zurück in die anwendungsspezifischen Datentypen konvertiert. Dies wird ebenso von dem Broker ohne zusätzlichen Aufwand seitens des Entwicklers bereitgestellt.

Bevor über die Verwendung von RTMP ein Urteil getroffen werden kann, müsste zuerst eine aussagekräftige Evaluation erfolgen. Es würde den Rahmen dieser Arbeit sprengen, die verfügbaren Server und deren Tauglichkeit zu überprüfen. Die Schwerpunkte liegen bei RTMP bei der Übertragung von Video und Audio. Um RTMP zu nutzen, stehen zwei Server zur Verfügung (siehe 5.3.3, S. 99). Red5 wurde noch nicht veröffentlicht, daher ist es fraglich, ob dieser Server für die Entwicklung einer Anwendung eingesetzt werden sollte. Der Flash Media Server ist zum einen kostenpflichtig und zum anderen würde die Anwendung an ein proprietäres Produkt gebunden werden. Es ist ungewiss, ob RTMP, wenn es eine so enge Bindung an eine bestimmte kommerzielle Servergruppe zur Folge hat, entsprechenden Mehrwert bietet.

Eine Alternative zu der Verwendung von Sockets stellt der Einsatz eines Messaging-Dienstes dar. Das Prinzip des Messagings wurde bereits kurz in Abschnitt Message Passing (4.4.3) auf Seite 46 erläutert. Messaging scheint auf den ersten Blick, für das Verteilen von Änderungen während einer laufenden Anwendung sinnvoll zu sein. Dabei bleibt es fraglich, wie einmalige Abläufe, wie die Anmeldung oder das initiale Laden der verschiedenen Stammdaten, mit Messaging realisiert werden können. Möglich wäre es z.B. einen punkt-zu-punkt-Kanal zu öffnen, der nur von einem Client an den Server reicht. Ist dies nicht möglich, muss eine der auf HTTP-basierten Request-Response-Varianten genutzt werden.

Beispiel für den Einsatz von Messaging

Der BlazeDS[4], ein Open Source Server, ermöglicht die Nutzung des Java Messaging Services (JMS). An dieser Stelle soll nicht auf die einzelnen Technologien eingegangen werden, sondern anhand eines kurzen Beispiels der Einsatz verdeutlicht werden. Eine genauere Betrachtung ist nicht Gegenstand dieser Arbeit.

Messaging: Config

Zuerst muss für den BlazeDS der Kanal konfiguriert werden, der für die Nachrichtenübertragung genutzt werden soll. Dies wird in der `messaging-config.xml`, im Verzeichnis WEBAPP_ROOT/WEB-INF/flex vorgenommen.

[4] http://opensource.adobe.com/wiki/display/blazeds/

```
<destination id="feasibility">
</destination>
```

Nun ist der Kanal innerhalb von Flex verfügbar.

Messaging: Konsument

Über die MXML-Komponente Consumer kann der Nachrichtenkanal angesprochen werden. Um eine Nachricht zu verarbeiten, wird diese an die `messageHandler()`-Methode übergeben, welche den Text der Nachricht über ein Textfeld ausgibt. Anhand der zwei Knöpfe wird geregelt, ob auf auf die Nachrichten des Kanal reagiert werden soll.

```
<mx:Consumer id="consumer" destination="feasibility" message="
   messageHandler(event.message)"/>
<mx:Script>
    <![CDATA[
        import mx.messaging.messages.IMessage;
        private function messageHandler(message:IMessage):void{
            serverClientAnswer.text= "This message was received by
                the Server: "+message.body;
        }

        private function initHandler(event:Event){
            consumer.subscribe();
        }
    ]]>
</mx:Script>
    <mx:VBox height="100%" width="100%">
        <mx:Text id="serverClientAnswer" text="no Message received
            so far"/>
        <mx:Button label="Subscribe" click="consumer.subscribe()"
           enabled="{!consumer.subscribed}"/>
        <mx:Button label="Unsubscribe" click="consumer.unsubscribe
           ()" enabled="{consumer.subscribed}"/>
    </mx:VBox>
```

Messaging: Produzent zu Konsument

Der Produzent wird als ein eigener Thread realisiert, der ständig Nachrichten schickt. Es wird in diesem Beispiel nur die Kernmethode `triggerMessage()` aufgezeigt. Sie verschickt automatisch generierte Nachrichten, die aus einem Text und der aktuellen Zeit zusammengesetzt sind. Über die Methode `setDestination()` wird der Kanal festgelegt, an den die

Nachricht geschickt werden soll. Die Methode `routeMessageToService()` der Klasse `flex.messaging.MessageBroker` ist für das Verschicken der Nachricht zuständig.

```
private void triggerMessage(String msgContent){
    System.out.println("sendMessage.jsp triggered:"+msgContent);
    MessageBroker msgBroker = MessageBroker.getMessageBroker(null)
       ;
    String clientID = UUIDUtils.createUUID();

    AsyncMessage msg = new AsyncMessage();
    msg.setDestination("feasibility");
        msg.setClientId(clientID);
        msg.setMessageId(UUIDUtils.createUUID());
        msg.setTimestamp(System.currentTimeMillis());
        msg.setBody(new String(msgContent));
        msgBroker.routeMessageToService(msg, null);
}
```

Der Thread wird über eine jsp-Datei gestartet und angehalten.

```
<%@page import="test.MessageSender"%>
<%
    try {
        MessageSender sender = new MessageSender();
        sender.start();
        out.println("MessageThread started");
    } catch (Exception e) {
        out.println("A problem occured while starting the feed: "+
           e.getMessage());
    }
%>
```

Fazit

Da es keinen Objekt-Broker in Flex gibt, hat dies starke Auswirkungen auf das gesamte Design der Anwendung. Die Anwendung kann nicht entsprechend der Anforderungen, auch nicht durch die Verwendung einer Hilfskonstruktion, zu einem ähnlichen Verhalten gebracht werden. Das Beispiel suggeriert zwar, dass der Einsatz von Messaging eine praktikable Alternative darstellt, doch muss dies erst näher untersucht werden, um eine eindeutige Antwort geben zu können.

6.2. Entwurfsmusteranforderungen

6.2.1. Fassade

ActionScript bietet die Möglichkeit, Interfaces zu nutzen, um Fassaden zu erstellen, siehe Abschnitt Interfaces (5.3.1 S. 86). Wenn eine Klasse ein bestimmtes Interface implementiert und ein Objekt nur auf dessen Interface arbeitet, bleibt die Komplexität vor dem aufrufenden Objekt verborgen. Das Entwurfsmuster ist somit implementierbar.

6.2.2. Beobachter

In Flex wird das Beobachter-Muster, durch das in Abschnitt 5.3.1 auf Seite 86 vorgestellte Ereignis-Modell realisiert. Die Klasse, die Ereignisse auslösen kann, muss entweder eine Subklasse von `flash.events.EventDispatcher` sein oder das Interface `flash.events.IEventDispatcher` implementieren. Der Beobachter muss eine Funktion definieren, welche einen Parameter des Typs `flash.events.Event` oder eine Subklasse davon akzeptiert. Die Namen in dem Beispiel sind dabei analog gehalten zu den Namen der UML-Abbildung 4.8 des Beobachtermusters auf Seite 50.

Beispiel

Die `Application`-Klasse initalisiert das Subject, sowie den Beobachter. Wird der Knopf der Application-Klasse gedrückt, wird zu dem `subjectZustand` eins hinzugezählt. Der Beobachter wird verständigt und es öffnet sich ein Fenster, indem der neue Zustand ausgegeben wird.

```
<mx:Script>
    <![CDATA[

        private var subject:KonkretesSubjekt
        private var beobachter:KonkreterBeobachter;

        private function initializeHandler(event:Event):void{
            subject = new KonkretesSubjekt();
            beobachter = new KonkreterBeobachter(subject);
        }
    ]]>
</mx:Script>
<mx:Button click="subject.subjectZustand=subject.
   subjectZustand+1" label="Hier klicken um hochzuzählen"/>
```

KonkretesSubjekt besitzt eine Variable, _subjectZustand des Typs **int**. Über die get- und set-Accessor-Methode subjectZustand, kann auf die Variable zugegriffen werden. Auf diesem Wege erhöht der Knopf den Wert von _subjectZustand. Wird die Variable neu gesetzt, wird ein Event ausgelöst.

```
public class KonkretesSubjekt extends EventDispatcher
{
    public static const CHANGED:String = "changed";
    private var _subjectZustand:int = 0;

    public function KonkretesSubjekt(target:IEventDispatcher=
       null)
    {
        super(target);
    }

    public function get subjectZustand():int{
        return _subjectZustand;
    }

    public function set subjectZustand(aNum:int):void{
        _subjectZustand = aNum;
        dispatchEvent(new Event(KonkretesSubjekt.CHANGED));
    }
}
```

Der Konstruktor des KonkreterBeobachter verlangt ein Parameter des Typs KonkretesSubject, bei welchem er sich als Beobachter anmeldet und auf ein bestimmtes Event wartet. Wenn das Event eintritt, wird ein Fenster geöffnet, welches den neuen Zustand anzeigt.

```
public class KonkreterBeobachter
{
    private var _subject:KonkretesSubjekt;
    public function KonkreterBeobachter(subject:
       KonkretesSubjekt)
    {
        _subject = subject;
        _subject.addEventListener(KonkretesSubjekt.CHANGED,
           aktualisiere);
    }

    public function aktualisiere(event:Event){
        Alert.show("neue Zustand:"+_subject.subjectZustand);
    }
```

```
}
```

Fazit

Der Machbarkeitstest zeigt, dass das Beobachter-Muster ist in Flex umsetzbar ist.

6.3. Sprachanforderungen

Die Anforderungen an die Spracheigenschaften aus Abschnitt 4.10 werden gesammelt dargestellt und nach deren Unterstützung durch Flex gegliedert. Bei Anforderungen, die nicht unterstützt werden können, soll, wenn möglich, ein alternativer Lösungsweg dargestellt werden. Eine Bewertung erfolgt im Fazit am Ende der Anforderungsdarstellung. Falls eine Anforderung nicht unterstützt wird und es auch keine sonstige Lösung für diese Anforderung gibt, wird entsprechend dargestellt, welche Auswirkungen sich daraus ergeben.

Der Unterstützungsgrad wird wie folgt unterteilt:

unterstützt Kann in Flex analog zu Java umgesetzt werden.

bedingt unterstützt Kann in Flex mit Einschränkungen umgesetzt werden.

nicht unterstützt Kann in Flex nicht umgesetzt werden.

6.3.1. Auflistung der Spracheigenschaft nach Unterstützungsgrad

- Vererbung: **unterstützt**
- Polymorphie: **bedingt unterstützt**
  - Method-Overriding: **unterstützt**
  - Method-Overloading: **nicht unterstützt**
  - Definiton abstrakter Klassen: **nicht unterstützt**
- Exception-Handling: **unterstützt**
- Datenkapselung anhand von Schnittstellen: **unterstützt**
- Generics: **nicht unterstützt**
- Reflexion: **unterstützt**

6.3.2. Unterstützte Spracheigenschaften

Vererbung

Das Konzept der Vererbung ist im Abschnitt Vererbung (5.3.1) auf Seite 86 des Flex Kapitels beschrieben und wird unterstützt.

Method-Overriding

Das Überschreiben einer Methode wird im Abschnitt Methoden (5.3.1) auf Seite 83 im Kapitel Flex erläutert.

Exception-Handling

ActionScript stellt den auch in Java üblichen Mechanismus des try/catch/finally für die Fehlerbehandlung zur Laufzeit zur Verfügung. Da entfernte Methodenaufrufe oft asynchron ablaufen, wird für die Behandlung von asynchronen Fehlern ein ereignisbasiertes Modell verwendet. Die Fehlerbehandlung wird ausführlich in Abschnitt Fehlerbehandlung (5.3.1) auf Seite 87 des Flexkapitels behandelt.

Datenkapselung anhand von Schnittstellen

Das Konzept der Datenkapselung wird in ActionScript über Interfaces unterstützt, siehe Abschnitt Interfaces (5.3.1) Seite 86 im Kapitel Flex. Die Interfaces in Java unterscheiden sich dabei von den Interfaces in ActionScript dadurch, dass sie auch Variablen in Interfaces definiert werden können. Dieser Unterschied ist minimal und somit zu vernachlässigen.

Reflexion

Das Erzeugen von Objekt-Instanzen anhand eines Klassennamens und der dynamische Aufruf einer Methode für ein Objekt wird in Abschnitt Reflexion (5.3.1 S. 88) erläutert und von Flex unterstützt.

6.3.3. Nicht unterstützte Spracheigenschaften

Method-Overloading

Es ist mit ActionScript nicht möglich, mehrere Methoden desselben Namens mit unterschiedlichen Parametern zu definieren. Es gibt drei verschiedene Möglichkeiten, ein ähnliches Verhalten zu erzeugen; diese werden im Abschnitt Methoden (5.3.1 auf Seite 83) erläutert.

Der Rest-Parameter(...) Über den Rest-Parameter ist es möglich, eine Methode mit einer beliebigen Anzahl von Parametern anzusprechen. Abhängig von der Anzahl der Parameter kann entschieden werden, welche Methode aufgerufen wird. Die varargs-Notation in Java verhält sich analog zum Rest-Parameter.

```
public function doSomething(...args):void
{
    switch (args.length) {
        case 0: doSomething0(); break;
        case 1: doSomething1(args[0]); break;
        case 2: doSomething2(args[0], args[1]); break;
    }
}
```

Der Wildcard-Parameter(*) Der Wildcard-Parameter erlaubt die Eingabe jeglicher Wertetypen. Über den is-Operator, kann der Typ festgestellt und dann entsprechend verzweigt werden.

```
public function doSomething(obj:*) : void
{
    if (obj is int)
       doIntMethod(obj);
    else if (obj is String)
        doStringMethod(obj);
    else
        //another method
}
```

Der Default-Parameter Es ist möglich, für Parameter Default-Werte anzugeben. Für diese Parameter, muss bei einem Aufruf der Methode kein Wert mitgegeben werden, es wird anstelle der Default-Wert verwendet.

```
public function doSomething(obj:String,action:String = "default")
    : void
{
     if(action=="default")
         doDefault(obj);
     else
         doOther(obj,action);
}
```

Fazit Bei der Verwendung des Rest- und des Wildcard-Parameters wird die strikte Typisierung umgangen. Zusätzlich ist über die Methodensignatur nicht mehr unbedingt zu erkennen, wofür die Methode eigentlich gedacht ist, und die möglichen Parameter müssen entweder direkt dem Code oder der Dokumentation entnommen werden. Sollen Methoden mit mehr als einem Parameter überladen werden, ist es wichtig, von welchem Typ die Parameter sind und es müssen die beiden Varianten kombiniert werden.

Eventuell treten Fehler bei der Verwendung des Rest- und des Wildcard-Parameters erst zur Laufzeit auf, daher sollten sie nur eingesetzt werden, wenn dies wirklich zwingend notwendig ist. Es sollte daher bei Method-Overloading auf eine Namenskonvention gesetzt werden. Ein typischer Fall von Overloading, ist z.B. das Hinzufügen von Objekten verschiedener Typen: `addElement(element:Square)` und `addElement(element:Round)`. Hier sollten die Methodennamen sinnvoll gewählt werden z.B. `addElementSquare(element:Square)` und `addElementRound(element:Round)`.

Über die Nutzung von Default-Parametern kann eine andere Grundlage für die Ausführung einer Methode erreicht werden. Dies bietet aber nur ein wenig mehr Flexibilität. Wenn eine Methode mehrere Default-Parameter verwendet und nur der Wert für den letzte Default-Parameter angegeben werden soll, müssen für alle vorherigen die Default-Werte fest eingetragen werden. Ändern sich die default Werte im Laufe einer Entwicklung, kann es daher sein, dass eventuelle Fehler nicht direkt erkannt werden. Dies könnte verhindert werden, wenn die Default-Parameter über static-Variablen definiert werden. Über den Flex-Builder wird die Methodensignatur samt der Default-Werte angezeigt.

Da die strikte Typisierung bei der Verwendung von Default-Parametern erhalten bleibt, ist die Nutzung der Default-Parameter eher als die des Rest- und des Wildcard-Parameters zu vertreten.

Abstrakte Klassen

Es ist in ActionScript nicht möglich, abstrakte Klassen zu definieren. Der allgemeine Weg, um eine abstrakte Klasse in ActionScript zu simulieren, ist eine normale Klasse zu verwenden,

welche die entsprechenden Methoden implementiert. Dabei werfen alle abstrakten Methoden einen IllegalOperationError und geben Null zurück, falls sie etwas zurückgeben müssen.

```
public class AbstractElement
{
    public function AbstractElement()
    {}

    public function getId():String{
        throw new IllegalOperationError("This is an abstract Class
            and the Method public function getId():String must be
           overridden");
        return null;
    }
}
```

Fazit Da abstrakte Klassen nicht direkt von der Sprache unterstützt werden, können etwaige Fehler auch nicht durch den Compiler und die Entwicklungsumgebung sichtbar gemacht werden. Ein Fehler könnte z.B. darin bestehen, dass eine Subklasse nicht alle abstrakten Methoden überschreibt. Da sie als Subklasse trotzdem die Methoden anbietet, treten erst zur Laufzeit Fehler zu Tage. Abstrakte Klassen sollten daher nur, wenn es wirklich nötig ist, eingesetzt werden. Zudem sollte bei der Verwendung abstrakter Klassen besonders auf deren UnitTests geachtet werden. Gerade, wenn mehr als nur ein Entwickler ein Projekt betreut, kann es hier durch Kommunikationsmangel einfacher zu unerwarteten Laufzeitfehlern kommen.

Generics

Für das Konzept der Generics in Java gibt es innerhalb von Flex kein äquivalentes Konstrukt. In ActionScript ist es möglich, jeden Objekt-Typ in einem Array oder einem assoziativem Array (die dynamische Objekt-Klasse) zu speichern. Dies wird in Abschnitt Arrays (5.3.1) auf Seite 85 im Kapitel Flex erläutert. Wenn nun ein Objekt aus einem Array entnommen wird, muss es entsprechend konvertiert werden, um den entsprechenden Typ zu garantieren.

Fazit Anhand von Generics kann die Entwicklung deutlich flexibler und auch typsicherer gestaltet werden. Da das Konzept der Generics innerhalb des Designs nicht für die Umsetzung zentraler Punkte genutzt wird, ist es zu vernachlässigen, dass Flex dieses Feature nicht unterstützt.

6.3.4. Bedingt unterstützte Spracheigenschaften

In diesen Bereich fällt die Polymorphie, da ein Teil der Voraussetzungen erfüllt und ein anderer Teil nicht erfüllt werden kann. Im Fazit wird erläutert, ob Polymorphie wie üblich einsetzbar ist.

Polymorphie

Die Polymorphie wird nur bedingt unterstützt, da Method-Overloading und die Definition von abstrakten Klassen nur anhand von Hilfskonstruktionen unterstützt werden.

Fazit Abstrakte Klassen sind zwar per Definiton nicht verfügbar, trotzdem kann ihr Verhalten in gewissem Maße simuliert werden. Im Bereich des Method-Overloading sollte auf Namenskonventionen gesetzt werden, um von der Sicherheit der strikten Typisierung zu profitieren. Polymorphie ist also durchaus umsetzbar. Allerdings sollte ein Entwickler darauf achten, komplexe Strukturen eher serverseitig abzubilden, falls dies möglich ist.

6.4. Bewertendes Fazit

In diesem Abschnitt werden die Anforderungen und deren Umsetzbarkeit als Ganzes bewertet und erläutert.

Das Design und die daraus resultierenden Anforderungen sind nicht mit Flex umsetzbar. Dies ist hauptsächlich darauf zurückzuführen, dass für Flex kein Object-Broker vorhanden ist. Es war nicht möglich, das Fehlen des Object-Brokers über einen anderen Mechanismus zu kompensieren. Im Machbarkeits-Abschnitt des Brokers (6.1.3), wurden dazu zwei verschiedene Alternativen benannt, die Verwendung von XML-Sockets oder Messaging. Unabhängig davon, welche Alternative eingesetzt wird, muss das Design überarbeitet werden. Das Konzept des Messaging bietet sich für eine Designgrundlage der Vision eher an als die XML-Socket-Variante. Dies ist mit dem Umstand zu erklären, dass der Grundgedanke der Synchronisierung von Elementen ein Kernpunkt der Anwendung darstellt. Das Prinzip des Messaging selbst stammt aus dem Bereich der parallelen Verarbeitung, wobei die Synchronisierung von Objekten dort ebenso eine Schlüsselrolle spielt. Aus der Verwendung des Messaging als grundlegender Kommunikationsmechanismus entspringt ein eigener Zweig von Architekturmustern. In Dustdar S. (2003) werden zwei Architekturen für die Verteilung und Synchronisierung von Informationen vorgestellt, die Push-Architektur und das Event-Based-System (EBS). Aktuelle Entwicklungen aus der Forschung im Bereich der EBS werden bei

event-based.org[5] dargestellt. Beide Quellen sollen hier nur als Anhaltspunkte für eine nähere Betrachtung des Themas dienen. Die Anwendung eines dieser Architekturmuster auf das Design stellt eine einschneidende Veränderung dar und zieht eine grundlegende Überarbeitung nach sich. Die Überarbeitung würde den Rahmen dieser Arbeit überschreiten.

Betrachtet man die weiteren Architektur- und Entwurfsmuster des Anforderungskataloges, lassen sich diese einwandfrei umsetzen. Gerade die Konzepte des MVC- und des Beobachter-Musters konnten durch den in Flex grundlegenden Einsatz von Ereignissen gut unterstützt werden.

Die Sprachanforderungen konnten von Flex erfüllt werden. Es ist allerdings unverkennbar, dass ActionScript sich aus dem Bereich der Skripting-Sprachen zu einer objektorientierten Sprachen entwickelt hat. Diese Herkunft wird unter anderem dadurch erkennbar, dass es nicht zwingend notwendig ist, Datentypen festzulegen oder es neben der objektorientierten Zugriffsweise auf Methoden eine literale Zugriffsweise gibt (siehe Beispiel „Dynamischer Methodenaufruf eines Objektes“ auf Seite 90). Um komplexe objektorientierte Strukturen auch in ActionScript vollständig zu unterstützten, sollten die Konzepte des Method-Overloadings und der abstrakten Klassen in die Sprache aufgenommen werden. Anwendungen, welche stark auf diese beiden Konzepte aufbauen, sollten diese Vorzugsweise serverseitig realisieren.

Flex ist grundsätzlich für die Entwicklung komplexer kollaborativer Client-Server-Anwendungen zu gebrauchen, doch basiert diese Entwicklung auf einer anderen Designphilosophie, als die der traditionellen Objektorientierung. Über die Nutzung von MXML können schnell Prototypen entwickelt werden, da Flex dem Entwickler z.B. über Datenbindung die Verknüpfung von grafischen Komponenten und deren Model abnimmt. Der Prototyp kann dann inkrementell erweitert und die Zuständigkeiten sauberer getrennt werden. Die Bestimmung der Eigenheiten dieser Philosophie sind nicht Gegenstand dieser Arbeit, sollten aber bei einem Architekturentwurf und dem Prozess der Umsetzung berücksichtigt werden.

[5] http://event-based.org/

7. Schlussfolgerung

Zum Abschluss dieser Arbeit sollen in diesem Kapitel eine Zusammenfassung der bisherigen Ausführungen und ein Ausblick auf zukünftige Entwicklungen gegeben.

7.1. Zusammenfassung

Im Rahmen dieser Arbeit befasste sich der Autor mit der Evaluation des Flex Frameworks, mit dessen Hilfe Rich Internet Applications erstellt werden können.

Im Einleitungskapitel 1 wurde die Art und Weise der Evaluation sowie die kollaborative Anwendung selbst näher bestimmt. Es ergab sich folgende Fragestellung:

> Kann anhand von Flex 2.0 eine Rich Internet Application entwickelt werden, wobei diese den Anforderungen einer kollaborativen Desktopanwendung genügt, welche auf Basis von gängigen Architektur- und Entwurfsmustern erstellt wurde?

In Kapitel 2 wurden die für das Verständnis der Arbeit nötigen Grundlagen dargestellt. Es wurden die Konzepte von Rich Internet Applications geschildert, indem zuerst auf traditionelle Webanwendungen, dann auf den Begriff des Web 2.0 und abschließend auf Rich Internet Applications selbst eingegangen wurde. Anschließend erfolgte eine kurze Übersicht von Rich Internet Application Frameworks. Es wurden zwei Ajax-basierte Frameworks, Google Web Toolkit und Echo 2, sowie ein Plug-in-basiertes Framework, Microsoft Silverlight 1.0 vorgestellt. Das Grundlagenkapitel schließt mit der Darstellung des Begriffes des Computer Supported Cooperative Work ab.

In Kapitel 3 wurde zuerst die Vision der Referenzanwendung beschrieben, daraus wurden die Anforderungen abgeleitet. Am Ende des Analysekapitels erfolgte eine Abgrenzung zwischen den Anforderungen, welche die Basisfunktionalität der Anwendung bereitstellen und Anforderungen, welche eine zusätzliches, aber nicht für den Minimalentwurf notwendiges Verhalten bereitstellen.

Im Designkapitel (4) wurden die angewendeten Architektur- und Entwurfsmuster sowie das endgültige Design selbst vorgestellt. Einleitend erfolgte ein Überblick der verwendeten Architekturmuster. Anschließend wurde das Schichtenmuster erläutert und im folgenden Abschnitt auf die möglichen Teilungen der Schichten in Bezug auf Client-Server-Anwendungen eingegangen. Dabei wurde herausgestellt, wo die Anwendung selbst zu positionieren ist. Danach wurde die Kommunikation von Client-Server-Anwendungen, die Rolle des Broker-Architekturmusters und die Kommunikation anhand von Webservices und Message Passing dargelegt. Nun erfolgte eine grobe Darstellung und Erläuterung der Serverkomponenten. Anschließend wurde auf die Clientkomponenten sowie die Dokumentenstruktur eingegangen. Um das Zusammenspiel der Komponenten zu verdeutlichen, wurden im folgenden Abschnitt ausgewählte funktionale Anforderungen anhand von Sequenzdiagrammen beschrieben. Abschließend wurden aus dem Design die Anforderungen an Flex zusammengefasst und kategorisiert.

Das Flex Framework, bestehend aus MXML- und ActionScript-Komponenten, wurde in Kapitel 5 vorgestellt. Bei der Beschreibung des Frameworks wurde sich auf die für die Anforderungen relevanten Bereiche beschränkt. Diese umfassen zum einen allgemeine Spracheigenschaften und zum anderen Themen wie Reflexion, Ereignisverarbeitung, Datenbindung und Remote-Kommunikation. Der letzte Abschnitt des Kapitels beschäftigte sich mit der Art und Weise, wie Anwendungen mit Flex entwickelt werden können und geht auf die Entwicklungsumbegung, das Debugging, Logging und Unit-Testing ein.

In der Machbarkeitsstudie (6) wurden die Anforderungen des Designskapitels wieder aufgegriffen und bezüglich der Umsetzbarkeit durch Flex geprüft. Das Broker-Muster konnte nicht entsprechend der Anforderungen umgesetzt werden, alternative Kommunikationsmechanismen wurden diskutiert, bewertet und in einem Machbarkeitstest geprüft. Dem MVC- sowie dem Beobachter-Muster wurde anhand eines Machbarkeitstests die Umsetztbarkeit bescheinigt. Der Umsetzbarkeitsgrad der Spracheigenschaften wurde anhand des Flex-Kapitels diskutiert und belegt; es konnten keine grundlegenden Komplikationen festgestellt werden. Abschließend erfolgte ein bewertendes Fazit.

Die Fragestellung aus dem Einleitungskapitel (siehe oben), konnte nicht mit „Ja“ beantwortet werden. Dies ist auf das Architekturmuster des Objekt-Brokers zurückzuführen, welches durch Flex nicht unterstützt wird. Das Muster garantiert innerhalb von Desktopanwendungen Ortstransparenz. Allerdings wurde eine weiterer Kommunikationsmechanismus als Alternative vorgeschlagen, Messaging Passing, wobei die Tauglichkeit von Flex erneut untersucht werden müsste. Dies würde eine weitere Evaluation erfordern, die nicht Gegenstand dieser Arbeit ist.

7.2. Ausblick

Rich Internet Applications stellen eine Renaissance der Rechenzentren dar. Die Software läuft zentral und wird über ein Netzwerk auf den Client übertragen. Der Client kann dabei sehr leicht ausfallen, eine Mindestanforderung ist ein gängiger Browser und eventuell ein zusätzliches Plug-in. Als ein Beispiel kann hier der JackPC[1] der Firma Chip PC Technologies angeführt werden. Der JackPC ist ein kostengünstiger Rechner, welcher sich in die Wand oder ins Mobiliar einbauen lässt. Über den JackPC kann der Benutzer per Web Browser auf die Webanwendungen des Rechenzentrums zugreifen. Abbildung 7.1 zeigt einen JackPC.

Abbildung 7.1.: JackPC

Bei der Verwendung eines Clients wie dem JackPC, besteht eine dauerhafte Netzwerkverbindung. In diesem Szenario würden Rich Internet Applications für das Erledigen der alltäglichen Arbeit innerhalb des Unternehmens genutzt werden.

Greift man das Beispiel des Außendienstmitarbeiters aus dem Einleitungskapitel (1.1) wieder auf, gibt es hier zwei alternative Herangehensweisen, um Rich Internet Applications zu nutzen. Entweder wird davon ausgegangen, dass der Mitarbeiter ständig online ist, oder es muss ein Mechanismus zur Synchronisation der Daten bereitgestellt werden, welche während der Zeit, in der keine Verbindung möglich war, manipuliert worden sind. Obwohl die Synchronisation von Rich Internet Applications immer mehr von den verschiedenen Frameworks unterstützt wird, z.B. anhand von Google Gears[2] oder Adobe Air[3], würde dies das

[1] http://www.chippc.com/thin-clients/jack-pc/
[2] http://gears.google.com/
[3] http://www.adobe.com/de/products/air/

reine Konzept der Rich Internet Applications aushöhlen. Es ist zu erwarten, dass auch im mobilen Bereich, immer flächendeckendere und leistungsstärkere Technologien entwickelt werden und Rich Internet Applications daher besser unterstützt werden können.

Literaturverzeichnis

[Adob] ADOB: *Unit Testing and Test Coverage with Flex*. – URL http://weblogs.macromedia.com/swebster/archives/2007/11/unit_testing_an.html

[Adobe a] ADOBE: *ActionScript 3.0 Livedocs: Basic Function Concepts*. – URL http://livedocs.adobe.com/flex/3/html/03_Language_and_Syntax_18.html#119202

[Adobe b] ADOBE: *ActionScript 3.0 Livedocs: Methods*. – URL http://livedocs.adobe.com/flex/3/html/04_OO_Programming_07.html#129545

[Adobe c] ADOBE: *ActionScript 3.0 Livedocs: Socket connections*. – URL http://livedocs.adobe.com/flex/3/html/17_Networking_and_communications_5.html

[Adobe d] ADOBE: *ActionScript 3.0 Reference API: Function-Class*. – URL http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/Function.html

[Adobe e] ADOBE: *Flex Livedocs: About metadata tags*. – URL http://livedocs.adobe.com/flex/201/html/metadata_141_03.html

[Bayes L.] BAYES L., Mills A.: *Evaluation Application Frameworks that work with Flex*. – URL http://www.asserttrue.com/articles/2007/10/17/silvafug-application-frameworks-presentation

[Borghoff 1998] BORGHOFF, Schlichter: *Rechnergestütze Gruppenarbeit*. Springer Verlag, 1998

[Buschmann u. a. 2001] BUSCHMANN ; MEUNIER ; ROHNERT ; SOMMERLAD ; STAL: *Pattern-Oriented Software Architecture A System of Patterns*. John Wiley & Sons Ltd., 2001

[Carfagno 2007] CARFAGNO: *Evaluation des Google Web Toolkits durch Entwicklung einer ajaxbasierten Mind-Mapping-Anwendung*, HAW-Hamburg, Diplomarbeit, 2007

[Dehats] DEHATS: *Dehats.com / From Cairngorm to PureMVC - a quick comparison / Comment by Cliff Hall*. – URL http://www.dehats.com/drupal/?q=node/26#comment-121

[Dustdar S. 2003] DUSTDAR S., Hauswirth M.: *Software-Architekturen für Verteilte Systeme*. Springer-Verlag Berlin Heidelberg 2003, 2003

[ECMA a] ECMA: *ECMA-Script*. – URL http://www.ecmascript.org/

[ECMA b] ECMA: *ECMAScript for XML*. – URL http://www.ecma-international.org/publications/standards/Ecma-357.htm

[Ellis u. a. 1991] ELLIS, Clarence A. ; GIBBS, Simon J. ; REIN, Gail: Groupware: some issues and experiences. In: *Commun. ACM* 34 (1991), Nr. 1, S. 39–58. – ISSN 0001-0782

[Gamma u. a. 2001] GAMMA ; HELM ; JOHNSON ; VLISSIDES: *Entwurfsmuster Elemente wiederverwendbarer objektorientierter Software*. Addison Wesley Verlag, 2001

[Garrett 2005] GARRETT: Ajax: A New Approach to Web Applications. (2005)

[Google] GOOGLE: *GWT Overview*

[Grudin 1994] GRUDIN: *CSCW: History and Focus. IEEE Computer*. 1994. – 19–26 S

[Janzen 2005] JANZEN, David S.: Software architecture improvement through test-driven development. In: *OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA : ACM, 2005, S. 240–241. – ISBN 1-59593-193-7

[Johansen 1988] JOHANSEN, R.: *Groupware: Computer Support for Business Teams*. Free Press, 1988

[Kabalkin 2008] KABALKIN: *Migration einer Rich Client Applikation auf Code on Demand Technologie*, HAW-Hamburg, Diplomarbeit, 2008

[Kazoun C. 1. Auflage 2008] KAZOUN C., Lott J.: *Programmieren mit Flex 2*. O'Reilly Verlag, 1. Auflage 2008

[Lesser u. a. 2005] LESSER ; GUILIZZONI ; LOTT. ; REINHARD ; WATKINS ; EPSTEIN, Bruce (Hrsg.): *Programming Flash Communication Server*. O'Reilly Media, Inc., 2005

[Loosley] LOOSLEY, Chris: Rich Internet Applications: Design,Measurement and Management Challenges. In: *Keynote System*

[Microsoft] MICROSOFT: *MSDN: Silverlight Overview*. 2008. – URL http://msdn.microsoft.com/en-us/library/bb404708(VS.95).aspx

[Microsoft 2008] MICROSOFT: *MSDN: Silverlight 1.0 Architecture*. 2008. – URL http://msdn.microsoft.com/en-us/library/bb693293.aspx

[Mund 2006] MUND: *Berechtigungsstrukturen in kollaborativen Umgebungen*, HAW-Hamburg, Diplomarbeit, 2006

[Neary und Cappello 2002] NEARY, Michael O. ; CAPPELLO, Peter: Advanced eager scheduling for Java-based adaptively parallel computing. (2002), S. 56–65. ISBN 1-58113-599-8

[NextApp 2007] NEXTAPP: *Echo2 Technical Overview*. 2007. – URL http://echo.nextapp.com/site/echo2/doc/tov

[Nezlek 2007] NEZLEK, Jason Farrell & George S.: Rich Internet Applications The Next Stage of Application Development. In: *Proceedings of the ITI 2007 29th Int. Conf. on Information Technology Interfaces* (2007)

[release Notes a] NOTES, Flex 1. release: *Adobe: Flex 1.5 release Notes*. – URL http://www.adobe.com/support/documentation/en/flex/1_5/releasenotes.html#whatsnew

[release Notes b] NOTES, Flex 3. release: *Adobe: Flex 3.0 release Notes*. – URL http://www.adobe.com/support/documentation/en/flex/3/releasenotes_flex3_sdk.html#whatsnew

[O'Reilly 2005] O'REILLY, Tim: What Is Web 2.0 Design Patterns and Business Models for the Next Generation of Software. In: *O'Reilly* (2005)

[actionscript praxis.de] PRAXIS.DE actionscript: *Presseinformation von Macromedia zum Thema Flex*. – URL http://actionscript-praxis.de/forum/viewtopic.php?t=713

[PureMVC a] PUREMVC: *Official PureMVC-Site*. – URL http://puremvc.org/

[PureMVC b] PUREMVC: *PureMVC-Überblick anhand UML-Diagramme*. – URL http://puremvc.org/component/option,com_wrapper/Itemid,35/

[Putzke-Röming u. a. 1998] PUTZKE-RÖMING, W. ; RADETZKI, M. ; NEBEL, W.: A flexible message passing mechanism for objective VHDL. (1998), S. 242–249. ISBN 0-8186-8359-7

[Red5] RED5: *Open Source Flash Server: Red5*. – URL http://osflash.org/red5

[Revout 2008] REVOUT, Alexandra: *Ein System für das kollaborative Bearbeiten von Dokumenten in mobilen Umgebungen*, HAW-Hamburg, Diplomarbeit, 2008

[Robson] ROBSON, Jim: *PureMVC vs. Cairngorm*. – URL http://robsondesign.com/blog/index.php/2008/04/10/puremvc-vs-cairngorm/

[Siedersleben 2003] SIEDERSLEBEN, J. ; J., Siedersleben (Hrsg.): *Softwaretechnik Praxiswissen für Softwareingenieure*. Carl Hanser Verlag München Wien, 2003

[Sommerville 2004] SOMMERVILLE: *Software Engineering 7*. Addison-Wesley Publishers Limited Pearson Education Limited, 2004

[Starke 2005] STARKE, Gernot: *Effektive Software-Architekturen Ein praktischer Leitfade 2. Aufage*. Carl Hanser Verlag München Wien, 2005

[Tanenbaum 2007] TANENBAUM, Marteen v.: *Distributed Systems Principles and Paradigms*. Pearson Education, Inc, 2007

[Teufel S. 1995] TEUFEL S., Sauter C.: *Computerunterstützte Gruppenarbeit*. Addison-Weasly, 1995

[Websector] WEBSECTOR: *Websector, 10 tips for working with PureMVC*. – URL http://www.websector.de/blog/2007/12/25/10-tips-for-working-with-puremvc/

[Wikipedia a] WIKIPEDIA: *Wikipedia: Adobe Flex*. – URL http://de.wikipedia.org/wiki/Adobe_Flex

[Wikipedia b] WIKIPEDIA: *Wikipedia: Macromedia*. – URL http://de.wikipedia.org/wiki/Macromedia

[Wikipedia c] WIKIPEDIA: *Wikipedia: MXML*. – URL http://en.wikipedia.org/wiki/MXML

A. Anhang

A.1. Anforderungen

A.1.1. Textabschnitt erstellen (F:T:A1)

B	öffnet das Kontextmenü des Strukturelements und wählt „Textabschnitt hinzufügen“ aus
S	Textabschnitt-Erstellen-Dialog erscheint
B	wählt ein Synonym aus, unter dem der Abschnitt in der Übersicht angezeigt werden soll und klickt auf „Erstellen“
S	fügt den Textabschnitt hinzu

A.1.2. Textabschnitt bearbeiten (F:T:A2)

Abschnitt „Element synchron bearbeiten“ (3.2.12) stellt den generellen Ablauf des synchronen Bearbeitens dar und muss zuerst durchlaufen werden. Danach kann der Benutzer die synchron editierbaren Elemente des Textabschnitts, den Inhalt, in einem Editor bearbeiten. Der Editor erlaubt es, den Text zu formatieren.

A.1.3. Textabschnitt verschieben (F:T:A3)

Das Verschieben eines Textabschnittes läuft analog zu dem Verschiebevorgang eines Strukturelements ab (siehe 3.2.8). Ein Textabschnitt kann dabei einem Strukturelement oder dem Schriftdokument zugeordnet werden.

A.1.4. Textabschnitt sperren (F:T:A4)

Ein Benutzer kann einen Textabschnitt sperren. Diese Sperrung bewirkt, dass kein anderer Benutzer den Textabschnitt ändern kann. Der Benutzer kann einen Grund für die Sperre und einen voraussichtlichen Zeitpunkt eintragen, an dem der Bereich voraussichtlich wieder frei verfügbar sein wird. Es wird eine Nachricht an alle Personen geschickt, welche das Dokument bearbeiten können. Es können weiterhin Kommentare für den entsprechenden Abschnitt erstellt werden.

A.1.5. Textabschnitt kommentieren (F:T:A5)

1	B	klickt auf das Kommentar-Symbol neben dem zu kommentierenden Textabschnitt
2	S	es erscheint die Kommentarübersicht, falls noch kein Kommentar existiert, folgt direkt Schritt4
3	B	es ist möglich, einen komplett neuen Kommentar zu erstellen oder einen Kommentar zu einem vorherigen Kommentar zu schreiben
4	S	das Übersichtsfenster wird abgeblendet und es erscheint das Kommentar-Erstellen-Fenster.
5	B	fügt seinen Kommentar in das entsprechende Feld ein und klickt auf „Kommentar einfügen“

A.1.6. Textabschnitt löschen (F:T:A6)

Ein Textabschnitt kann nur gelöscht werden, wenn es nicht gesperrt oder in Bearbeitung ist.

B	öffnet das Kontextmenü des Textabschnitt und wählt „löschen“ aus
S	überprüft, ob der Textabschnitt gelöscht werden darf, falls nicht, wird eine Fehlermeldung angezeigt
B	sieht die Übersicht des Schriftdokuments

A.2. Zusätzliche Informationen: Grundlagen des Flex Frameworks

A.2.1. Lebenszyklus einer Anwendung

Es ist wichtig den Lebenszyklus einer Flex-Anwendung zu verstehen um Features wie z.B eigene Preloader konfigurieren zu können, Laden und wieder Freigeben von Klassenbibliotheken oder Assets während der Laufzeit durchzuführen und generell optimierten Code zu entwickeln.

Das Flex Framework baut auf Flash-Anwendungen auf. Die Wurzel einer Flex-Anwendung ist der SystemManager und nicht die Application-Klasse. Der SystemManager ist eine Subklasse von flash.display.MovieClip. Ein MovieClip ist ein DisplayObject, welches Frames unterstützt. Hier wird die Verwandtschaft zum traditionellen Flash und dessen Herangehensweise besonders deutlich, da dort in der Regel auf einer Art Zeitleiste gearbeitet wird. Das .swf-Format ist ein progressives Downloadformat und somit kann der Flash Player bereits auf die übertragenen Inhalte der Frames zugreifen, obwohl noch nicht die ganze Datei heruntergeladen wurde. Der Systemmanager besitzt zwei Frames, in dem ersten ist der Preloader und in dem zweiten, die Anwendung inklusive dem Verwendeten Teil des Flex Frameworks enthalten. Der Preloader zeigt eine Fortschrittanzeige an, während die Anwendung im zweiten Frame geladen wird. Er ist sehr klein und enthält nur einen sehr geringen Teil des Flex Frameworks, wodurch der Frame sehr schnell heruntergeladen und ausgeführt werden kann. Wenn die Anwendung geladen ist, erzeugt der Preloader ein complete-Ereignis und die SystemManager-Instanz geht zum zweiten Frame über und erzeugt für die Anwendung eine Instanz der Application-Klasse. Die SystemManager-Klasse besitzt eine application-Eigenschaft, welche bei der Initialisierung des SystemManagers auf null gesetzt wird. Wenn nun das application-Objekt in Frame 2 erzeugt und der application-Eigenschaft zugewiesen wurde, wird die Anwendungsinstanz über eine eigene Startup-Prozedur initialisiert. Die internen Lebenszyklus-Ereignisse sind:

- preinitialize: Die Anwendung wurde instanziiert, aber noch keine Kindkomponenten hinzugefügt
- initialize: Die Anwendung hat Kindkomponenten erzeugt, diese aber noch nicht angelegt
- creationComplete: Die Anwendung wurde vollständig instanziiert, alle Komponenten angelegt.

Diese Ereignisse werden an SystemManager geschickt, welcher wiederum den Preloader über den Initialisierungsschritt benachrichtigt (vgl. Kazoun C. (1. Auflage 2008) S.107). Ist

die Startup-Prozedur abgeschlossen, wird der SystemManager benachrichtigt welcher wiederrum ein applicationCompleteEreignis anstößt um zu melden, dass die Anwendung startbereit ist. Der Preloader benachrichtigt den SystemManager, wenn er entfernt werden kann (vgl. Kazoun C. (1. Auflage 2008) S.107).

Der SystemManager besitzt eine Eigenschaft topLevelSystemManager. topLevelSystemManager hält eine Referenz auf die Wurzel-SystemManager-Instanz, welche den Ursprung bildet für alles, was zurzeit auf dem Flash Player läuft. Bei einer Flex-Anwendung, welche als die Hauptanwendung in den Flash Player geladen wurde verweist die Eigenschaft auf sich selbst. Eine Flex-Anwendung kann allerdings in eine andere Flex-Anwendung geladen werden. Ist dies der Fall, so zeigt der topLevelSystemManager auf das SystemManager-Objekt der übergeordneten Flex-Anwendung.

Alle Subklassen von UIComponents, also auch Application, besitzen eine systemManager-Eigenschaft und können so auf den SystemManager zugreifen. Dabei wird dies hauptsächlich zum Einsatz kommen, um auf Ereignisse zu warten die von einem DisplayObject in der Anwendung erzeugt worden sind. Das SystemManager-Objekt ist dabei die letzt Möglichkeit, Ereignisse zu verarbeiten bevor sie verloren gehen. (vgl. Kazoun C. (1. Auflage 2008) S.97 - S.99)

A.2.2. Eine Flex-Anwendung in eine andere Flex-Anwendung laden

Das eigentlichen Laden einer Flex-Anwendung kann einfach über eine SWFLoader-Instanz innerhalb eines MXML-Tags geschehen, wobei die source-Eigenschaft auf die .swf-Datei der zu ladenden Anwendung zeigt. Nachdem der SWFLoader die Anwendung geladen und initialisiert hat, wird das init-Ereignis ausgelöst. Nun kann der SystemManager der geladenen Anwendung angesprochen werden und ein Event-Listener für das applicationComplete-Ereignis registriert werden. Der Event-Listener kann bei auftreten des Ereignisses das Application-Object einfach referenzieren und die public-Methoden der geladenen Anwendungen ansprechen. (vgl. Kazoun C. (1. Auflage 2008) S.102 - S.104)

Anwendungsdomains

Das Konzept der Anwendungsdomains ist sehr wichtig, wenn .swf-Dateien in eine Flex-Anwendung geladen werden. Eine Anwendungsdomain stellt dabei eine Partition dar, in welcher eine Anwendung im Flash Player läuft. Wird nun eine .swf-Datei in eine Flex-Anwendung geladen, kann es passieren, dass sich vollständig qualifizierten Klassennamen eventuell gleichen. Um dieses Problem zu lösen werden Anwendungsdomains in drei verschiedenen Varianten eingesetzt:

1. Die .swf-Datei läuft in einer Anwendungsdomain, welche ein Kind einer existierenden Anwendungsdomain ist. Wenn eine Anwendung innerhalb einer Kinddomain läuft, erbt sie alle Klassen der Elterndomain. Dabei werden alle Klassen, die in der Kinddomain doppelt existieren (bei einer Flex-Anwendung also alle Klassen des Frameworks) zugunsten der gleichen Klassen der Elterndomain verworfen. Daraus ergeben sich folgende Vorteile:
   - weniger Speicherbedarf, da doppelte Klassen ausgeschlossen werden.
   - Dieselben Singletons sind für Eltern- und Kindanwendungen zugänglich und können gemeinsam verwendet werden.
   - Es ist möglich doppelte Klassen während der Kompilierung der Kind-.swf-Datei auszuschließen und somit die Dateigröße zu reduzieren
2. Die .swf-Datei läuft in einer neu erstellten, eigenen Anwendungsdomain. Überschneiden sich die vollständig qualifizierten Klassennamen, doch es werden unterschiedliche Implementationen verwendet, kann es sein, dass die Anwendung innerhalb einer Kinddomain nicht wie erwartet funktioniert. Daher muss die Anwendung in einer eigenen Anwendungsdomain zugewiesen, dies sollte aber nur in Erwägung gezogen werden, falls es wirklich notwendig ist.
3. Die .swf-Datei läuft in einer bereits vorhandenen Anwendungsdomain. Diesen Verhalten wird von Runtime Shared Librarys verwendet. Eine Runtime Shared Library bietet die Möglichkeit, Assets und Bibliotheken in mehreren .swf-Dateien zu nutzen, ohne dass die Inhalte für jede .swf-Datei geladen werden müssen und die Dateien somit kleiner ausfallen (Runtime Shared Librarys vgl. Kazoun C. (1. Auflage 2008) S.39 - S.40).

(vgl. Kazoun C. (1. Auflage 2008) S.104 - S.107)

A.3. CD

Auf der CD befindet sich:

- diese Arbeit als PDF-Format
- ein Flex Builder Projekt mit den Machbarkeitstests für das MVC-, das Beobachter, und das Broker-Muster.
- ein Flex Builder Projekt mit den verschiedenen FlexUnit-Tests für die Sprachanforderungen. Die Bibliotheken für die FlexUnit sind enthalten.

- Die UML-Diagramme des Systementwurfs dieser Arbeit wurden mit dem UML Plug-in[1] für NetBeans entworfen und liegen in Form eines UML Projektes vor.

[1] http://uml.netbeans.org/

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 20. Juni 2008	
Ort, Datum	Unterschrift