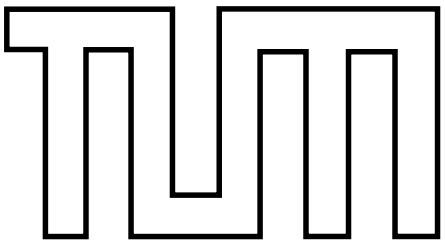
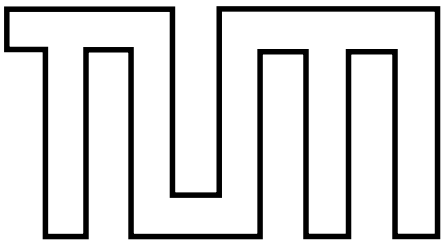


Fakultät für Informatik
der Technischen Universität München

Diplomarbeit in Informatik

Erweiterung des TeleTeachingTools um kooperative Whiteboardbenutzung und Instant Messaging

Thomas Döhring



Fakultät für Informatik
der Technischen Universität München

Diplomarbeit in Informatik

Erweiterung des TeleTeachingTools um kooperative Whiteboardbenutzung und Instant Messaging

Extension of the TeleTeachingTool by cooperative whiteboard utilisation and Instant Messaging

Bearbeiter:	Thomas Döhring
Aufgabensteller:	Prof. Dr. Helmut Seidl
Betreuer:	Michael Petter
Abgabedatum:	13.06.2008

Ich versichere, dass ich diese Diplomarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

_____

Datum, Unterschrift

Hörer von Vorträgen oder Vorlesungen sollen die Möglichkeit bekommen, dem Dozenten über elektronischem Wege Rückmeldungen geben zu können, um Fragen zu stellen oder Antworten geben zu können, da dies unter gewissen Umständen nicht mündlich möglich ist. Aufbauend auf der Präsentationsaufzeichnungssoftware TeleTeachingTool wird erläutert, wie dieses um ein Nachrichtensystem erweitert werden kann, so dass die Hörer Textnachrichten und durch eigene Annotationen erweiterte Folien und Whiteboards an den Dozenten senden können. Desweiteren wird auch eine Implementation des Nachrichtensystems erstellt.

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung in die Präsentationsaufzeichnung

Heutzutage ist es gang und gäbe, dass Vorlesungen bzw. Vorträge mit Hilfe eines Videoprojektors und eines Laptops gehalten werden. Auf dem Laptop hat der Dozent seine Folien, die er in seinem Vortrag zeigen möchte, vorbereitet, indem er zum Beispiel eine PowerPoint-Präsentation erstellt hat oder die Folien als PDF abgelegt hat. Zu Beginn der Präsentation startet er dann die PowerPoint-Präsentation oder öffnet die PDF-Datei in einem entsprechenden Anzeigeprogramm.

Da die Präsentation schon auf einem Rechner läuft, könnte dieser die Vorlesung auch gleich aufzeichnen. Stellt man dann die Aufzeichnung als Download bereit, würden Personen, die den Vortrag nicht hören konnten, die Möglichkeit erhalten, dies an einem Rechner nachzuholen. Aber auch Hörer des Vortrags können sich so Teile oder den ganzen Vortrag nochmals ansehen, um manche Erklärungen des Dozenten besser zu verstehen oder um ihre Notizen vervollständigen zu können. Neben den Präsentationsfolien an sich, muss auch die Stimme des Dozenten aufgezeichnet werden. Um um die Aufzeichnung abzurunden, wäre die Aufzeichnung eines Videobilds des Dozenten wünschenswert.

Damit der Dozent seine Erklärungen visuell unterstützen kann, bietet es sich an, dass der Dozent während der Vorlesung Markierungen auf seinen Präsentationsfolien oder sogar zusätzlichen Text oder Zeichnungen erstellen kann. Solche Annotationen unterstützen den Hörer bei der Aufnahme der Informationen aus dem Vortrag.

Drei Programme, die die Aufzeichnung einer Präsentation bieten, sind *Camtasia* [www-camtasia], *Lecturnity* [www-lecturnity] und das *TeleTeachingTool* [ttt-tum]. Sie bieten auch teils unterschiedliche Möglichkeiten der Annotation von Folien während des Vortrags. Camtasia und Lecturnity sind kommerzielle Produkte und kosten somit Geld,

hingegen kann das TeleTeachingTool kostenlos heruntergeladen werden. Die drei Programme weisen auch zum Teil unterschiedliche Ausrichtungen auf. Einen Überblick gibt zum Beispiel [Lämmle].

1.2 Rückmeldungen über Nachrichten(system)

Keine dieser Anwendungen bietet (bis dato) die Möglichkeit, dass Hörer während der Vorlesung auf elektronischem Wege über ein Nachrichtensystem Rückmeldungen an den Dozenten geben können. Rückmeldungen können Fragen zum Vortrag oder speziell zu einzelnen Folien sein oder auch Antworten auf Fragen, die der Dozent an die Hörer gestellt hat.

Die Möglichkeit von Rückmeldungen auf elektronischem Wege wird aber dann wichtig, wenn der normale Weg über Handmeldung und mündlichem Vortrag der Frage oder Antwort nicht oder nur eingeschränkt möglich ist. Dies ist beispielsweise der Fall, wenn die Vorlesung live in andere Räume oder gar - dank der globalen Vernetzung - irgendwohin z.B. in eine andere Universität übertragen wird. Das TeleTeachingTool bietet eine solche Live-Übertragung eines Vortrags an. Um für die Hörer in den entfernten Räumen einen Rückweg anzubieten, könnte man dort jeweils ein Mikrofon und vielleicht sogar eine Videokamera aufbauen, deren Signale man dann dem Dozenten anzeigt. Dies hat aber je nach Art und Weise der Audio- und Videoübertragung einen nicht unerheblichen technischen Aufwand und ggf. Kosten für die Geräte und Übertragungsvolumina zur Folge.

Aber nicht erst bei der Übertragung von Vorlesungen kann die mündliche Meldung schwierig sein, sondern auch schon bei einem normalen Vortrag vor einer großen Anzahl Hörern. Wegen des Geräuschpegel von hunderten oder gar über tausend Hörern, wie es auch im universitären Umfeld möglich ist, und der dann entsprechenden Größe des Vortragraums ist es unter Umständen schwierig bis unmöglich für den Dozenten, Hörer aus den hinteren Reihen akustisch zu verstehen.

Ein weiterer Punkt bei der elektronischen Rückmeldung ist die „Anonymität“ eines Hörers beim Stellen einer Frage gegenüber den anderen Hörern. Natürlich kann der Fragesteller je nach Art der Implementation der Nachrichtensystems durch seine IP, seine MAC-Adresse oder sein Zertifikat identifiziert werden. Diese Information bleibt aber zunächst dem Dozenten vorbehalten, sofern die Nachrichten der Hörer zunächst nur für den Dozenten und nicht für das breite Publikum sichtbar dargestellt werden. Dies kann die

Hemmschwelle senken, Fragen zu stellen, da die vermeintliche Blamage, eine „doofe“ Frage gestellt zu haben, ausbleibt. Dadurch bekommt der Dozent auch ein besseres Feedback, ob er manche Punkte seines Vortrages ausreichend erklärt hat.

Manchmal wäre es von Vorteil, wenn Hörer auf die Präsentationsfolien des Dozenten zeichnen könnten. Damit könnten sie die Stellen markieren, auf die sich ihre Frage bezieht. Damit ist es auch für den Dozenten leichter, den Kontext der Frage zu verstehen. Der Hörer könnte sich aber auch die Möglichkeit wünschen, seine Frage mit einer Zeichnung zu unterstützen. D.h., wenn die elektronische Rückmeldung auch die Bearbeitung von Folien des Dozenten und das Erstellen von Zeichnungen auf einem Whiteboard untersützt, könnte diese sogar Vorteile gegenüber dem mündlichen Stellen einer Frage haben.

Außerdem bietet die elektronische Übertragung von Fragen den Vorteil, dass diese mit in die Vorlesungsaufzeichnung übernommen werden können. Bei mündlichen Fragen müsste dazu entweder ein tragbares Mikrofon für Fragesteller verwendet werden, dessen Audiosignal mit in die Audio-Aufzeichnung des Dozenten gemischt wird oder getrennt aufgenommen wird. Oder der Dozent müsste die Frage wiederholen, was von diesem aber gerne vergessen wird.

1.3 TeleTeachingTool

Wie in Abschnitt 1.1 schon erwähnt, ist das TeleTeachingTool (kurz TTT) ein Programm zur Aufzeichnung von Vorträgen und Vorlesungen. Es wurde von Dr. Peter Ziewer zunächst an der Universität Trier entwickelt und wird derzeit an der TU München weiterentwickelt. [ttt-tum]

Aufgezeichnet werden können durch das TTT der Bildschirminhalt eines Rechners, die Stimme des Dozenten als Audiostrom und ein Videostrom. Zusätzlich bietet es für den Dozenten während der Vorlesung über verschiedene Zeichenfunktionen die Möglichkeit Bereiche hervorzuheben, Linien und Rechtecke zu zeichnen oder Freihandtexte zu schreiben. Diese Annotationen, die entweder direkt auf dem Bildschirminhalt oder auf Whiteboards erstellt werden, werden mit in die Aufzeichnung übernommen.

Das TeleTeachingTool ist in der Programmiersprache Java geschrieben und ist deswegen auf zahlreichen Betriebssystemen und Plattformen einsetzbar.

Das TTT verwendet VNC (*Virtual Network Computing*), welches die Darstellung des Bildschirminhalts und die Fernbedienung von entfernten Rechnern bietet. Damit kann

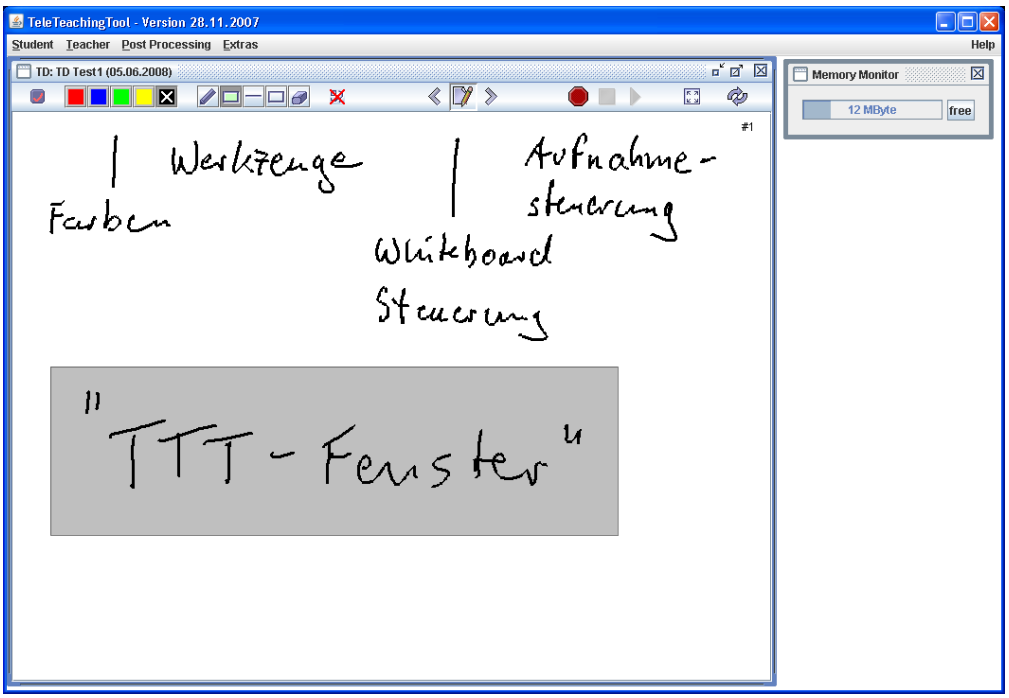


Abbildung 1.1: TeleTeachingTool mit einer gestarteten Dozentensitzung und einem bearbeiteten Whiteboard

der Dozent mit seiner gewohnten Computerarbeitsumgebung den Vortrag halten, indem er auf seinem eigenen Rechner einen VNC-Server installiert und startet und sich bei Beginn des Vortrags vom TTT-Präsentationsrechner aus auf seinem Rechner einloggt. Er kann dann in seiner gewohnten Umgebung die Vorlesung mit dem von ihm präferierten Programm halten. Für eine genauere Erklärung über den technischen Aufbau des TTT und die Verwendung von VNC sei auf die Publikationen des Entwicklers Dr. Ziewer verwiesen, die auch auf seiner Homepage zur Verfügung stehen [www-ziewer]

Die so aufgenommen Vorträge können entweder direkt mit dem TTT wiedergegeben werden oder in unterschiedliche Formate wie PDF, SWF (Flash) und HTML exportiert werden.

Noch ein kurzer Überblick über die Benutzeroberfläche, die das TTT dem Dozenten während der Aufzeichnung eines Vortrags bietet. Bild 1.1 zeigt diese. Innerhalb des

Hauptfensters befindet sich ein Fenster, dass den Bildschirm des per VNC verbundenen Rechners oder das derzeit bearbeitete Whiteboard zeigt. Auf dieses Fenster wird in dieser Arbeit über den Begriff „TTT-Fenster“ referenziert. Im oberen Teil dieses Fensters finden sich die verschiedenen Werkzeuge, mit denen der Dozent diesen Bildschirminhalt (meist eine Folie) annotieren kann. Außerdem kann er in einen Whiteboard-Modus schalten, bei dem er dann über hundert Whiteboards zur Erstellung von Zeichnungen zur Verfügung hat. Zwischen den Whiteboards kann er über die Vor- und Zurückbuttons umschalten.

1.4 Outline

In dieser Arbeit sollen die Möglichkeiten erörtert werden, wie das TeleTeachingTool um einen Rückweg erweitert werden kann, mit dem Hörer Fragen stellen und Antworten geben können. Neben dem Versand von Textnachrichten sollen diese auch die Möglichkeit erhalten, Folien des Dozenten mit Annotationen zu erweitern oder auch selbst neue Whiteboards zu erstellen.

Im nächsten Kapitel 2 werde ich anhand von Anwendungsfällen die Anforderungen erarbeiten, um die im Zuge dieser Arbeit das TTT erweitert werden soll. Daraufhin nenne ich in Kapitel 3 mögliche Umsetzungen für diese Anforderungen. Aus diesen werden ich dann in Kapitel 4 eine sinnvolle auswählen und deren Implementationen beschreiben.

Zum Ende hin werde ich in Kapitel 5 ermitteln, in wie weit die Implementation alle Anforderungen umsetzt. Abschließend gebe ich noch einen kurzen was erreicht wurde und werde Möglichkeiten zur Verbesserung und Erweiterung nennen.

2 Grundlagen

In diesem Kapitel werden die Anforderungen ermittelt, die die Erweiterung des TeleTeachingTools aufweisen soll. Dazu wird zunächst das allgemeine Szenario beschrieben und danach an möglichen Anwendungsfällen der Rahmen für zu implementierende Funktionen abgesteckt. Abschließend werden in 2.4 die gefunden Anforderungen nochmals übersichtlich zusammengefasst.

2.1 Allgemein

Der Ausgangspunkt ist ein mit dem TeleTeachingTool gehaltener Vortrag. Damit der Dozent bequem direkt am Bildschirm Freihandtext eingeben kann, sollte der Bildschirm eine GraphicTablet-Funktion mit entsprechendem Eingabestift aufweisen. Dies kann entweder über einen Tablet-PC oder durch einen an einem normalen Rechner oder Laptop angeschlossenen Bildschirm mit entsprechender Funktion realisiert sein.

Da der Dozent normalerweise vor dem Bildschirm steht, wird dieser – außer ggf. zum Schreiben von Text – seine Hand nicht auf den Bildschirm ablegen. Deshalb sollten Mouse-Over Anzeigen, d.h. die Anzeige von Informationen, wenn der Mauszeiger eine bestimmte Zeitspanne lang still über einem Bereich gehalten wird, vermieden werden. Der Dozent müsste dafür seine Hand auf dem Bildschirm abstützen und dann den Stift für einige Zeit wenige Millimeter über den Bildschirm an der selben Position halten. Dies ist aber aus dem Stehen und während eines Vortrags keine Handlung, die flüssig von der Hand geht.

Aus dem selben Grund sollte die Bedienoberfläche für das Nachrichtensystem auf Doppelklicks verzichten. Der Bereich, in dem zwei aufeinanderfolgende Klicks liegen müssen, um als Doppelklick erkannt zu werden, ist zwar meistens einstellbar, nichtsdestotrotz hat die Beschränkung auf einfache Klicks eine ergonomerische Bedienung mit dem Eingabestift zur Folge. Ebenso sollte auf Rechtsklicks verzichtet werden. Die meisten Eingabestifte weisen zwar einen Knopf auf, mit dem dieser ausgeführt werden kann, dazu muss der Benutzer aber wieder dem Stift über dem Bildschirm schweben.

Außerdem sollten die Eingabeknöpfe bzw. „Klickbereiche“ eine ausreichende Größe aufweisen und nicht zu klein sein. Erstens sind sie dann leichter zu treffen und zweitens ist leichter erkennbar, welche Funktion sie ausführen. Auch angezeigter Text, etwa die empfangenen Nachrichten der Hörer, sollte groß genug dargestellt werden, damit der Dozent sich auch ein wenig vom Bildschirm entfernen kann und trotzdem noch eingehende Nachrichten im Blick hat.

Damit die Hörer Nachrichten an den Dozenten schicken können, benötigen sie natürlich ein (mobiles) elektronisches Gerät, auf dem dann der TTT-Nachrichten-Client ausgeführt werden kann. Eine Vorraussetzung, die die Geräte erfüllen müssen, dass sie einen drahtlosen Netwerkanschluß (kurz WLan) aufweisen, damit sie darüber eine Verbindung zum Rechner des Dozenten herstellen können. Dies können Laptops sein, die heutzutage fast immer WLan integriert haben, aber auch SmartPhones oder PDAs mit WLAN sein, wie z.B. Geräte mit dem Windows Mobile Betriebssystem.

Der unter Hörern häufiger anzutreffende Gerätetyp ist das Laptop, so dass in dieser Arbeit der Nachrichten-Client für die Verwendung auf Laptops entwickelt wird. Nichtsdestotrotz soll eine spätere Erweiterung der einsetzbaren Geräte um SmartPhones und PDAs vorgesehen sein, so dass dies u.a. bei der Auswahl des Kommunikationsprotokolls zu berücksichtigen ist.

2.2 Anwendungsfälle

2.2.1 Senden einer Textnachricht

Dieser Anwendungsfall (siehe Bild 2.1 auf Seite 14) stellt den einfachen Fall dar, dass ein Hörer eine Frage oder Antwort als Textnachricht an den Dozenten schickt. Als erstes muss der Dozent das Messaging aktivieren über einen Menüpunkt im TeleTeachingTool aktivieren. Dabei soll eine Bedienoberfläche eingeblendet und der Nachrichten-Server gestartet werden. Daraufhin können sich Hörer über den Client mit dem Server verbinden.

Allerdings muss dem Client mitgeteilt werden, wo der Server zu finden ist. Eine Möglichkeit wäre dies automatisiert durchzuführen, indem der Server in regelmäßigen Abständen Advertisement-Broadcasts sendet um seine Existenz preiszugeben. Oder der Client könnte eine Umfragenachricht als Broadcast senden, auf die der Server dann antwortet. Dies funktioniert aber nur so lange, wie sich der Client im selben Subnetz wie der Server

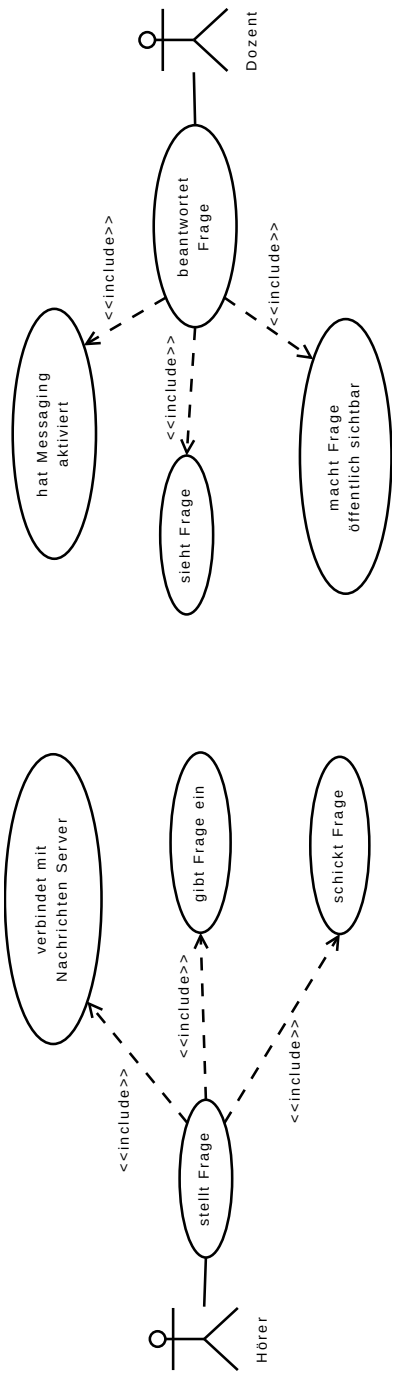


Abbildung 2.1: Anwendungsfalldiagramm: Senden einer Textnachricht

befindet. Dies könnte aber z.B. nicht der Fall sein, wenn eine Vorlesung mit Hilfe des TTT an eine andere Universität übertragen wird.

Die einfachste Variante ist die Bekanntgabe des Rechnernamens (sofern der Rechner einen solchen besitzt und darüber erreichbar ist) oder der IP-Adresse des Rechners durch den Dozenten zu Beginn des Vortrags. Die Hörer müssten dann diese zum Verbinden im Client eingeben. In Abwandlung davon könnte die IP auch im durch die Hörer sichtbaren Bereich der Videoprojektion eingeblendet werden. Sie wäre dann ständig sichtbar und wäre dann für die Unaufmerksamen und Vergesslichen unter den Hörern auch während des Vortrags verfügbar.

Ein weiterer Punkt ist, ob die Hörer einen Namen und/oder ein Passwort zum Verbinden eingeben müssen, oder ob eine andere Art der Authentifikation nötig ist. An der TU München werden z.B. an die Informatik-Studenten elektronische Zertifikate ausgegeben, die dazu verwendet werden können. Da aber das Nachrichtensystem kaum Mißbrauchsmöglichkeiten bietet, kann auf eine Authentifikation verzichtet werden. Dies hat auch den Vorteil, dass es für Dozenten einfacher ist, das Nachrichtensystem einzusetzen, als wenn diese oder deren Assistenten sich mit unter Umständen komplizierten Authentifikationssystemen auseinandersetzen müssen.

Nachdem der Hörer sich über den Client verbunden hat, gibt er den Text, den er schicken möchte ein, und schickt diesen an den Server des Dozenten. Somit muss im Client eine Texteingabemöglichkeit und ein Sende-Knopf bestehen.

Der Dozent bekommt dann den Text angezeigt. Die Anzeige der eingetroffen Nachrichten erfolgt am besten als Liste. Um die Anzahl der angezeigten Nachrichten zu erhöhen, soll zunächst nur ein Teil der Nachricht dargestellt werden. Um den vollen Text zu sehen muss der Dozent dann eine weitere Aktion ausführen. Eine MouseOver-Darstellung des kompletten Inhalts fällt als Möglichkeit wegen der Stiftbedienung weg. Aber die Darstellung des gesamten Textes nach einem einfachen Klick auf den bereits dargestellten Teil wäre ein gangbarer Weg.

Eine weitere Frage ist die Darstellung der Nachrichten für die Hörer. Wenn diese alle eingehenden Nachrichten sofort sehen würden, könnten diese vom Vortrag abgelenkt werden. Und um außerdem „Witzbolde“ auszuschließen, die nicht vortragsrelevante Texte schicken könnten, sollen die eingegangenen Nachrichten nur für den Dozenten sichtbar sein. Damit die Hörer jedoch einen Bezugspunkt haben, wenn der Dozent eine Frage über das Nachrichtensystem beantwortet, soll der Dozent eine Nachricht für das gesamte Publikum sichtbar darstellen können. Zusätzlich sollte der dargestellte Text mit in

die Aufzeichnung einfließen um auch während des Abspielens einer Aufzeichnung der Antwort des Dozenten folgen zu können.

2.2.2 Hörer als Störer

Hier geht es um Hörer, die den Dozenten stören wollen, indem sie unter Umständen zahlreiche Nachrichten mit sinnlosem, nicht zum Vortrag passenden Inhalt senden. (s. Diagramm 2.2)

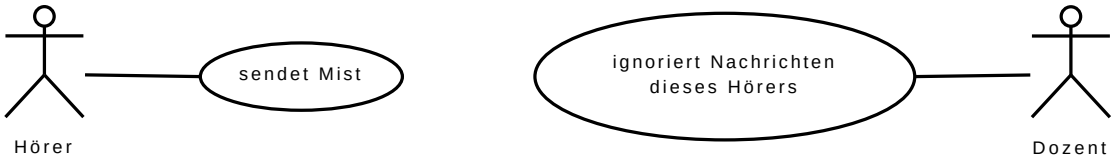


Abbildung 2.2: Anwendungsfalldiagramm: Hörer als Störer

Um nicht den Überblick über die anderen, sinnvollen Nachrichten anderer Hörer zu verlieren und um den Dozenten nicht durch ständiges Löschen der einzelnen Nachrichten des Störers von seinem Vortrag abzulenken, benötigt man eine durch den Dozenten schnell aufrufbare Funktion um Nachrichten des Störers zu ignorieren und die bisherigen, schon erhaltenen Nachrichten zu löschen.

Zu überlegen ist, ob der Störer auch mitbekommen soll, dass er ignoriert wird. So könnte nach dem Ignorieren durch den Dozenten bei weiteren Versuchen des Störers, eine Nachricht zu senden, bei diesem eine Meldung erscheinen, dass keine Nachrichten von ihm mehr dargestellt werden. Da eine solche Rückmeldung aber auch als Ansporn gesehen werden kann, solange zu nerven, bis man den „Ignoriert-Status“ erreicht, ist ein stilles Ignorieren der Nachrichten besser.

Wie identifiziert den Absender einer Nachricht, um die unerwünschten ausfiltern zu können? Ein einfacher Weg geht über die IP, die die Netzwerkschnittstelle des Geräts des Störers hat. Es wird einfach auf dem Dozenten-Server eine Liste mit den IPs der bisher ignorierten User geführt. Der Störer kann dies natürlich umgehen, indem er sich bei DHCP-verwalteten Netzen eine neue IP zuweisen läßt oder er selbst eine neue IP-Adresse vergibt. Außerdem wäre es möglich, dass vielleicht ein anderer Hörer dieselbe IP-Adresse durch DHCP zugewiesen bekommt, wenn der Störer seine vor Ende des Vortrages abgibt. Ersteres ist mit Mehraufwand für den Störer verbunden und letzteres ist recht

unwahrscheinlich, so dass sie kaum Einschränkungen für eine IP-basierte Blockierung von Nachrichten darstellen.

Eine andere Möglichkeit ist die Generierung einer Identifikationsnummer bei Installation des Nachrichten-Clients. Diese könnte zufallsbasiert sein und oder oder Eigenschaften des Gerätes miteinbeziehen. Der Störer kann aber herausfinden, wo diese Nummer peristent gespeichert wird, und durch Löschen eine Neugenerierung der Nummer auslösen. Sofern die Nummer teilweise zufallsbasiert ist, bekommt er eine neue und hätte damit die Blockierung umgangen.

Falls sich die Hörer (wie in 2.2.1 als Möglichkeit genannt) bei der Verbindung mit dem Server, beispielsweise über ein Zertifikat, eindeutig authenfizieren müssen, kann dies natürlich zur Identifizierung benutzt werden.

2.2.3 Antwortmöglichkeit und Sichtbarkeit von Nachrichten

Wenn der Benutzer eine Frage an den Dozenten sendet, soll diese auch bei den anderen Hörern mit verbundenem Client dort angezeigt werden? Diese Frage ist eng mit der ähnlichen Frage aus Abschnitt 2.2.1 verbunden, ob die Nachrichten in der Projektion sichtbar sein sollen. Der Vorteil wäre, dass die anderen Hörer dann sehen könnten, dass eine Frage, die sie vielleicht auch selbst stellen wollten, schon gestellt wurde. Damit würde man dem Dozenten Fragen mit dem gleichen oder ähnlichem Inhalt ersparen.

Ein Nachteil ist aber die „Mißbrauchsmöglichkeit“ als Kommunikationsmittel unter den Hörern. Diese könnten sich darüber unterhalten oder „Späßchen machen“ und würden sich selbst und andere damit vom Vortrag ablenken. Da vermutlich die Anzahl der ankommenden gleichlautenden Fragen sehr gering ist und diese Funktion von Hörern wie eben beschrieben verwendet werden würde, wird in dieser Arbeit auf die Implementierung dieser Funktion verzichtet.

Ein Punkt, der sich zum Thema „Instant Messaging“ stellt, ist, ob der Dozent sich über die Nachrichten mit einzelnen Hörern unterhalten oder ob er zumindest über Textnachrichten Antworten auf Fragen an alle Hörer senden können soll. Ersteres würde den Vortragsfluss stören, wenn der Dozent ohne für die anderen Hörer erkenntlichen Zweck in seinen Rechner Text eintippt. Zweiteres ist an sich schon nicht sinnvoll, da der Dozent die Antwort auf normalem Wege mündlich dem Publikum geben kann und damit alle Hörer anspricht und nicht nur diejenigen, die ein Gerät mit verbundenem Client vor sich haben.

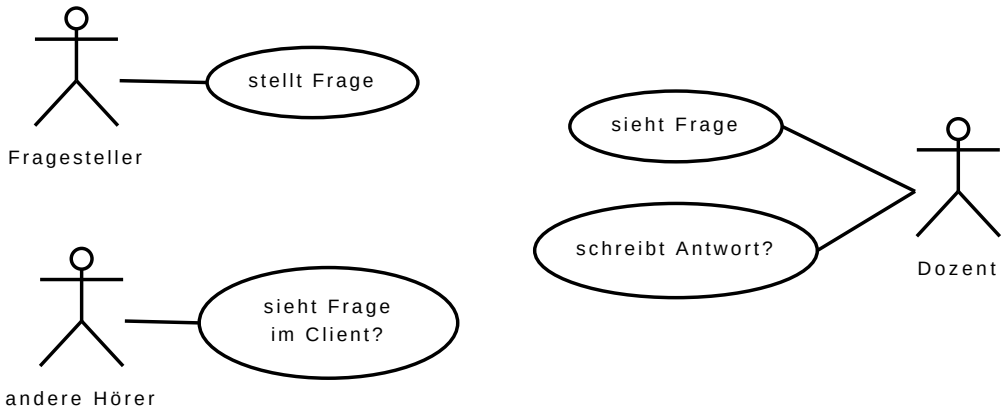


Abbildung 2.3: Anwendungsfalldiagramm: Antwortmöglichkeit und Sichtbarkeit von Nachrichten

Damit wird die Funktion, dass der Dozent Nachrichten versenden kann, nicht in die Anforderungsliste aufgenommen.

2.2.4 Abstimmungen

Eine Funktion, die man auf der Grundlage eines Nachrichtensystems aufbauen kann, ist die Durchführung von Abstimmungen. Natürlich kann man diese auch ohne elektronische Unterstützung durch einfache Handmeldung durchführen. Man bekommt damit aber nur einen ungefähren Überblick über die Stimmenverteilung, wohingegen die elektronischen Stimmen exakt ausgezählt werden können.

Um die Abstimmungen verwalten zu können, muss dem Dozenten eine entsprechende Benutzerschnittstelle zur Verfügung gestellt werden. Beim Erstellen muss er die Frage und die Antwortmöglichkeiten angeben. Der Dozent soll aber möglichst wenig von seinem Vortrag abgehalten werden, so dass das Erstellen einer Abstimmung schnell von statten gehen muss oder ihm die Möglichkeit gegeben werden muss, die Abstimmungen vor Beginn des Vortrags zu erstellen zu können.

Die Hörer sollen natürlich erst an zu Beginn erstellten Abstimmungen teilnehmen können, wenn der Dozent an der passenden Stelle in seinem Vortrag angekommen ist. Außerdem soll der Dozent eine Abstimmung auch wieder beenden können. Somit muss man

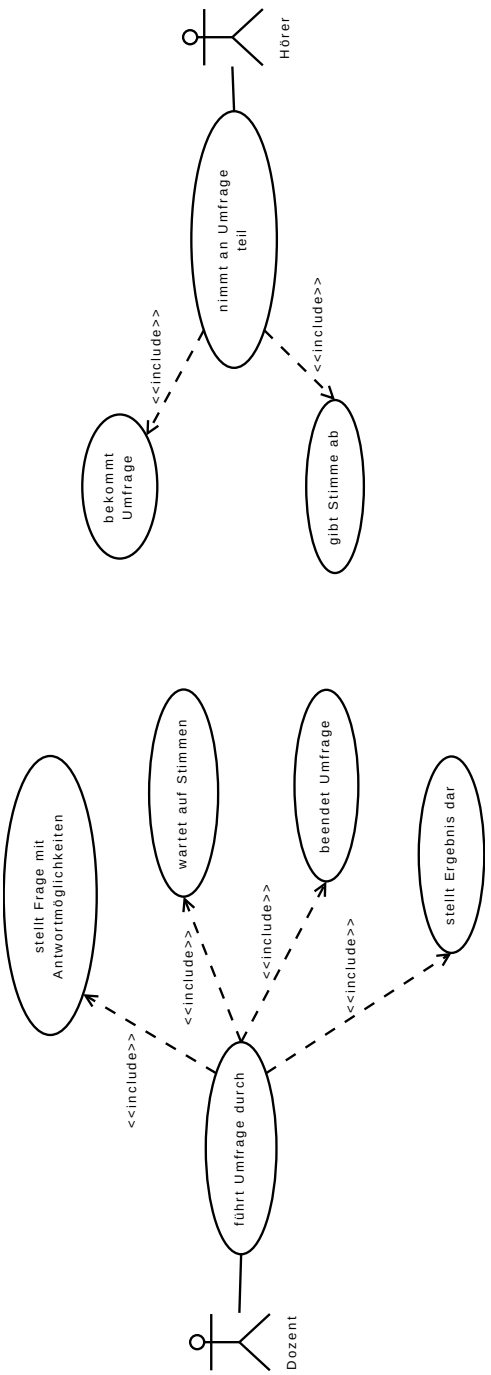


Abbildung 2.4: Anwendungsfalldiagramm Abstimmungen

ihm eine Funktion zum Freigeben und wieder Schließen der Abstimmungen zur Verfügung stellen.

Damit die Hörer auch ihre Stimme abgeben können, muss der Client von den aktuellen, freigegebenen Abstimmungen erfahren können und diese dann dem Hörer darstellen. Entweder der Server teilt den Clients mit, dass eine oder mehrere Abstimmungen freigegeben wurden. Dazu müssen die Hörer aber ihre Clients starten und zum Server verbinden bevor der Dozent eine Umfrage freigibt, damit sie die Nachricht des Servers mitbekommen. Auf diesem Weg würden aber Hörer von der Abstimmung ausgeschlossen, die zwar ein passendes Gerät für den Nachrichten-Client mit im Vortrag dabei haben, dieses aber ausgeschaltet oder den Client zu Beginn der Abstimmung nicht gestartet hatten.

Oder der Client holt sich eine aktuelle Liste der Abstimmungen mittels Nachfrage beim Server. Dies kann in festen zeitlichen Abständen automatisch im Hintergrund geschehen, so dass der Hörer ständig die aktuellen Abstimmungen sieht, wenn er verbunden ist. Oder die Liste wird erst abgefragt, wenn der Hörer sich dazu entscheidet, bei einer Abstimmung teilnehmen zu wollen. Erstere Variante würde je nach Abstand zwischen den automatischen Abfragenachrichten und Anzahl der Hörer mit verbundenem Client zu einer großen Anzahl an Abfragen führen. Da jedoch die Gesamtanzahl der durchgeführten Abstimmungen innerhalb eines Vortrages gering sein dürfte, bietet sich die zuletzt genannte Variante an.

Selbstverständlich soll jeder Hörer in jeder Abstimmung nur einmal eine Stimme abgeben können. Hierbei hat man dieselben Identifikationsmöglichkeiten wie beim Anwendungsfall in Abschnitt 2.2.2. Deshalb sollte man für das Ignorieren von Usern und für die Abstimmungen nur eine Identifikationslösung, die dann von beiden Teilen verwendet wird, auswählen und implementieren. Ein(e) Hörer/in sollte im Client auch nur Abstimmungen angezeigt bekommen, an denen er/sie noch nicht teilgenommen hat.

Damit der Dozent einen Überblick hat, wieviele Stimmen schon abgegeben wurden und wie der Trend der Abstimmung ist, sollte er die aktuelle Stimmenverteilung sehen können. Nach dem Ende einer Abstimmung soll der Dozent den Hörern die Ergebnisse darstellen können. Außerdem sollten die Ergebnisse mit in die Aufzeichnung des Vortrages gelangen.

2.2.5 Nachrichten mit Bezug zu einer Folie oder Whiteboard-Zeichnung

Die in Abschnitt 2.2.1 behandelte Textnachricht ist zwar eine einfache und schnelle Möglichkeit, dem Dozenten eine Frage zu stellen, kann aber nicht immer ausreichend sein. Wenn beispielsweise ein Hörer den Dozenten auf einen Fehler auf einer Folie aufmerksam machen möchte, ist es schwierig nur mittels Text zu erklären, wo der Fehler auf der Folie ist. Auch per Handmeldung ist das Deuten auf die Fehlerstelle schwierig sofern der Hörer nicht zufällig einen Laserpointer zur Hand hat.

Es wäre einfacher, wenn der Hörer die Fehlerstelle auf der Folie oder der Whiteboard-Zeichnung des Dozenten markieren könnte. Der Hörer soll somit die Möglichkeit bekommen, mit Hilfe des Clients die Folie oder das Whiteboard des Dozenten zu bearbeiten und dann zusammen mit seinen Anmerkungen wieder zurücksenden zu können.

Selbstverständlich beschränkt sich der Einsatz einer solchen Funktionalität nicht auf das Hinweisen von Fehlerstellen. Auch Fragen oder allgemein Anmerkungen, die Hörer haben, sind für den Dozenten besser verständlich wenn der Hörer die Stelle markiert, auf die er Bezug nimmt, oder er sogar eine kleine Zeichnung anfertigt.

Im Beispiel wurden nur Folien und die Whiteboard Funktionalität genannt. Das TeleTeachingTool ist aber wegen der Verwendung von VNC (s. Einleitung 1.3) zur Präsentation und Aufzeichnung von beliebigen Bildschirminhalten fähig, Folien sind nur ein möglicher Inhalt. Die Hörer sollen demnach Markierungen auf dem aktuellen Bildschirminhalt vornehmen können. Da die Präsentation von Folien die gebräuchlichste Anwendung ist wird im weiteren Verlauf weiterhin von Folien die Rede sein. Die entsprechenden Nachrichten zwischen Hörer und Dozent heißen damit auch *Foliennachrichten* (engl. *sheet message*). Man sollte aber im Hinterkopf behalten, dass es auch beliebig anderer Inhalt sein kann.

Das Nachrichtensystem muss damit die Übertragung von beliebigen Bildinhalten in Form eines pixelbasierten Grafikformats zusammen mit den Annotationen des Dozenten unterstützen. Der Client muss entsprechend die Folien und Annotationen darstellen können und Werkzeuge zur Erweiterung um weitere Annotationen bieten. Da die Hörer im allgemeinen keine Möglichkeit zur Stifteingabe an ihren Laptops besitzen, gestaltet sich das Schreiben von Text auf den Folien per Freihandtext schwierig. Deshalb sollte für die Hörer eine Eingabe von Text mittels Keyboard vorgesehen werden. Das TeleTea-

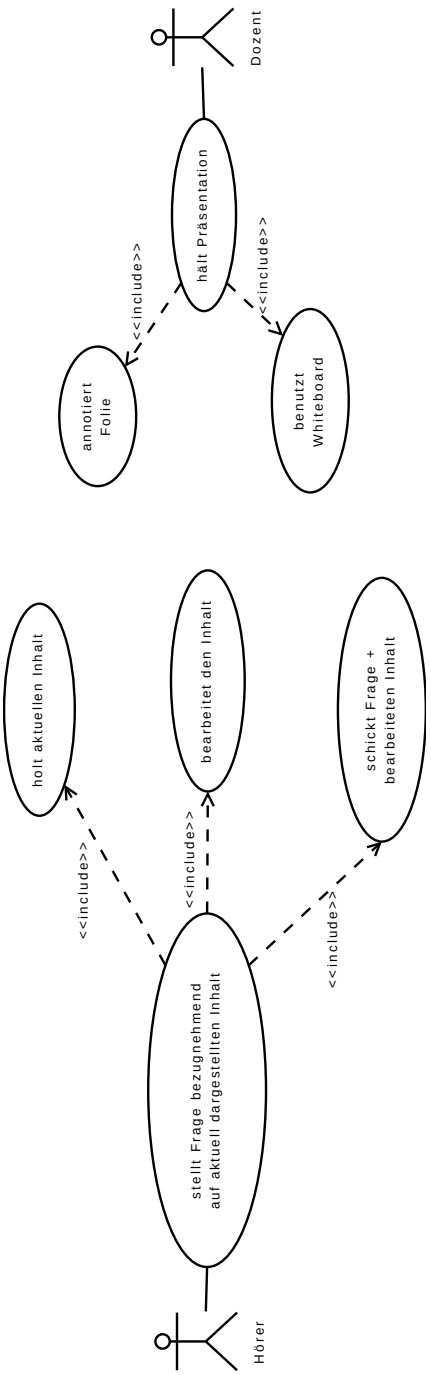


Abbildung 2.5: Anwendungsfalldiagramm: Nachricht des Hörers mit Bezug zu einer Folie oder Whiteboard-Zeichnung

chingTool bietet eine solche Art von Annotation nicht an und müsste deshalb erweitert werden.

Wenn nun die von Hörern zusätzlich annotierten Folien oder Whiteboards beim Nachrichten-Server eintreffen, müssen diese dargestellt werden. Die Textnachrichten und die Foliennachrichten sollten gemeinsam in der selben Liste dargestellt werden, damit die Ankunftsreihenfolge ersichtlich ist. Die Folien und Whiteboards können aber nicht ihrer orginalen Größe in der Nachrichtenliste dargestellt werden, so dass es nötig ist, kleine Vorschaubilder zu berechnen und diese zur Darstellung zu verwenden. Doch selbst die komplette Darstellung der Vorschaubilder kann zu groß sein, so dass in gleicher Weise wie bei den Textnachrichten zunächst nur ein Teil und erst nach Anforderung durch den Dozenten das komplette Thumbnail dargestellt wird.

Ist erstmal die Möglichkeit geschaffen, dass Hörer auf Folien Anmerkungen schreiben können, kann dies natürlich auch in den Vortrag mit eingebunden werden. Ein Beispiel aus einer Informatik-Vorlesung wäre, dass der Dozent ein Quellcode-Listing mit Fehlern auf einer Folie präsentiert und die Hörer dazu auffordert, die Fehler zu markieren und gegebenenfalls Korrekturen hinzuzufügen.

2.2.6 gleichzeitiges Arbeiten an einem Whiteboard

Ein Anwendungsfall, der sich beim Lesen des Titels dieser Arbeit („kooperatives Whiteboard“) ergibt, ist das Arbeiten des Dozenten und einem (oder sogar mehreren) Hörer(n) am selbem Whiteboard zur selben Zeit.

Zunächst bräuchte man eine Instanz, die die am gemeinsamen Whiteboard arbeitenden Teilnehmer verwaltet. Außerdem muss diese sich um die Konsistenzhaltung kümmern, damit jeder Teilnehmer dasselbe sieht. Ein direktes Verteilen der Annotationen vom erstellenden Teilnehmer zu den anderen würde dazu führen, dass die Teilnehmer die Annotationen nicht in derselben Reihenfolge bekommen, und es damit zu Darstellungsunterschieden kommen würde.

Damit nicht sofort Hörer in das Whiteboard des Dozenten zeichnen können, wenn dieser das Whiteboard im TeleTeachingTool öffnet, muss der Dozent dieses erst für das gleichzeitige Zeichnen freischalten.

Wie sinnvoll ist es jedoch, dies in das TTT aufzunehmen? Eine solche Funktion ist eher etwas im Rahmen eines Seminars oder ähnlichem, bei der in einem überschaubarem

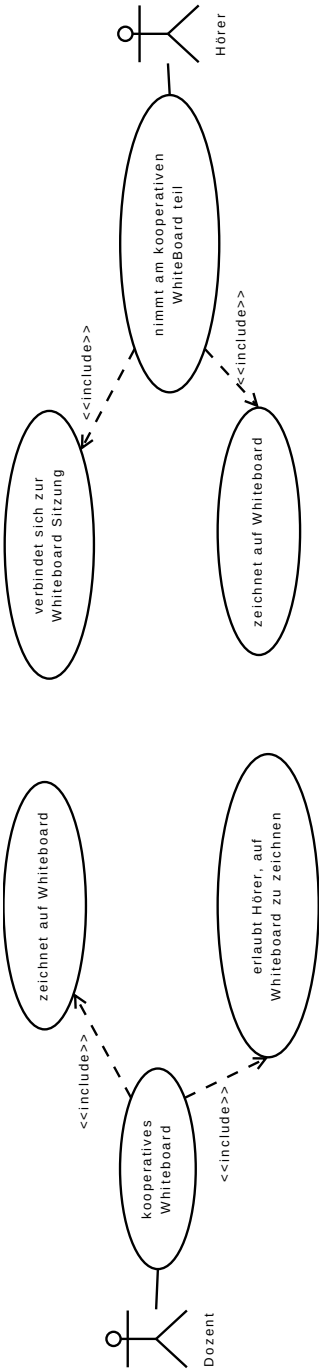


Abbildung 2.6: Anwendungsfalldiagramm: gleichzeitiges Annotatieren eines Whiteboards

Rahmen gemeinsam etwas erarbeitet werden soll. Während des Haltens eines Vortrages vor einer größeren Anzahl an Hörern wäre dies nicht sinnvoll.

Somit bleibt das gleichzeitige Arbeiten an einem Whiteboard im Rahmen dieser Arbeit außen vor.

2.3 weitere Punkte

Im folgenden werden einige weitere Anforderungen aufgestellt, die sich nicht oder nur indirekt aus den obigen Anwendungsfällen ergeben.

Der Nachrichten-Client könnte, je nach verwendetem Kommunikationsprotokoll (s. Abschnitt 3.1) auch in einer anderen Programmiersprache als das TeleTeachingTool geschrieben werden. Die Implementation sollte aber dennoch in Java erfolgen, da der Client dann in das TeleTeachingTool aufgenommen werden kann und der Hörer somit nur ein Programm zum Wiedergeben von Vortragsaufzeichnungen und zum Nachrichten schreiben installieren muss. Außerdem kann für gewisse Teile, wie zum Beispiel das Zeichnen auf Folien und Whiteboards Code aus dem TTT übernommen werden. Implementationen von Clients für andere Gerätetypen, als die in dieser Arbeit ausgewählten Laptops, müssen natürlich je nach Verfügbarkeit auf andere Programmiersprachen ausweichen.

Da der Nachrichtenserver eng mit dem TTT zusammenarbeitet (Holen von Folien und Annotationen, Darstellen von empfangen Nachrichten), soll dieser auch in das TTT aufgenommen werden. Dies hat auch den Vorteil, dass die Hemmschwelle für die Benutzung niedriger liegt als wenn dieser noch ein weiteres Programm starten müsste.

Damit der Dozent während seines Vortrages möglichst wenig durch die Nachrichten abgelenkt wird, muss dem Dozenten die Verwaltung der Nachrichten einfach gemacht werden. Eine Möglichkeit dazu ist, dass der Dozent Nachrichten zurückstellen kann, um später auf sie eingehen zu können. Die zurückgestellten Nachrichten sollten eine Markierung erhalten, damit sie von den nicht zurückgestellten unterscheidbar sind. Eine Funktion, um alle nicht zurückgestellten Nachrichten zu löschen, vervollständigt das ganze. Damit sieht dann der Dozent, zumindest bis neue Nachrichten eintreffen, nur noch die für ihn wichtigen Nachrichten.

2.4 Anforderungen

Im folgenden eine stichpunktartige Auflistung der aufgestellten Anforderungen:

- Implementation des Clients in Java, ggf. Integration in TTT
- Nachrichtenprotokoll auch für SmartPhones/PDAs geeignet
- Nachrichtenprotokoll muss zwischen Inhalten und Befehlen unterscheiden können
- Nachrichtenprotokoll muss Text als Binärdaten (Bilder) übertragen können
- Client: Eingabe von Text und das Versenden von Textnachrichten
- Client: Darstellen von Folien mit Annotationen und Whiteboards, erweitern dieser um Annotationen und Versand an Nachrichtenserver
- Client: Möglichkeit zur Teilnahme an Abstimmungen
- Client: Erstellen von Whiteboard-Zeichnungen ohne vorheriges Holen einer Folie
- Nachrichten-Server: Auslegung der Bedienoberfläche des Dozenten für Stiftbedienung (Einfachklicks, große Klickbereiche und Knöpfe)
- Nachrichten-Server: Darstellung von Nachrichten im TTT-Fenster und Aufnahme in Vortragsaufzeichnung
- Nachrichten-Server: übersichtliche Anzeige der eingetroffenen Nachrichten (zunächst nur Anzeige Teil der Nachricht)
- Nachrichten-Server: Bedienoberfläche sollte nur dem Dozenten sichtbar sein
- Nachrichten-Server: Zurückstellen von Nachrichten
- Identifizierung von Benutzern für das Blockieren von Nachrichten und die Abstimmungen
- Abstimmungen: übersichtliche Anzeige (mit aktueller Stimmenverteilung)
- Abstimmungen: einfaches Erstellen und Verwalten
- Abstimmungen: sollen geöffnet und wieder geschlossen werden können
- Abstimmungen: Darstellung des Ergebnisses im TTT-Fenster
- Abstimmungen: jeder Hörer nur eine Stimme pro Abstimmung

3 Lösungsideen

In diesem Kapitel werden für einige der eben erarbeiteten Anforderungen Lösungswege vorgestellt. Dies umfasst auch die Erläuterung von Vor- und Nachteilen des jeweiligen Wegs.

3.1 Kommunikation(sprotokoll)

Die Kommunikation zwischen den Clients und dem Nachrichtenserver sollte über eine Netzwerkverbindung eines IP-basierten Netzwerks laufen. Dafür gibt es verschiedene Möglichkeiten der Umsetzung, die in den folgenden Abschnitten vorgestellt werden.

3.1.1 Java RMI

Java bietet mit *Remote Method Invocation* (kurz RMI) eine Technik um verteilte Anwendungen zu erstellen. Dabei können Methoden von entfernten Objekten aufgerufen werden, die in anderen Java Virtual Machines liegen und damit auf unterschiedlichen Rechnern sein können. [sun-rmi]

Ein Vorteil davon ist, dass man sich als Programmierer nicht um die Netzwerkkommunikation kümmern muss. Ganz ohne Programmieren geht es natürlich auch nicht, da man u.a. die Schnittstellen der entfernten Objekte festlegen und entsprechende Objekte implementieren muss, die diese Schnittstellen auch aufweisen. Dann muss jeweils eine Instanz dieser Objekte in die RMI-Registry, auf dem Rechner, der diese Objekte anbieten soll, eintragen. Diese kann man dann unter Angabe des Namens des Objekts und des Rechners erreichen und die Methoden normal wie jede andere (lokale) Java-Methode aufrufen (abgesehen von den RemoteExceptions die Methoden des entfernten

Objekts werfen können). Als Parameter übergebene oder als Rückgabe gelieferte Objekt-Instanzen werden automatisch von der RMI Implementation serialisiert, übertragen und wieder deserialisiert. [sun-rmi-tut]

Ein Nachteil ist, dass man Clients nur für Plattformen implementieren kann, auf den Java verfügbar ist. Ein interessantes mobiles Gerät, für das es (noch) keine JVM gibt (Stand Mai 2008), ist das Apple iPhone.

Aber auch wenn eine JVM zur Verfügung steht, bedeutet dies noch nicht, dass man RMI nutzen kann. So ist RMI nicht im Standardumfang der Java 2 Platform, Mobile Edition (kurz J2ME) enthalten. Es ist nur ein Optional Package (J2ME RMI Optional Package [sun-rmiop]) und es bleibt den Herstellern der mobilen JVMs überlassen, ob sie dieses Package in ihre Produkte aufnehmen. Eine Übersicht (Stand Ende 2007) über verschiedene mobile JVMs und ihre unterstützten Profile, APIs und Packages gibt es auf [Menneisyys]. Laut dieser Tabelle unterstützt keine der dort aufgeführten mobilen JVMs das RMI Package.

3.1.2 Verwendung bestehendes Protokoll & Bibliothek

Eine weitere Implementationsmöglichkeit ist die Verwendung eines bereits bestehenden Nachrichtenübertragungsprotokolls und einer dazu passenden, schon existenten Java-Bibliothek.

Vorteil dieses Ansatzes wäre, dass man, wie beim RMI Ansatz, selbst keinen Code für die Netzwerkkommunikation entwickeln muss. Dies würde die Bibliothek erledigen. Allerdings muss man die Daten, die man versenden möchte, in eines der Transportformate konvertieren, welche die Bibliothek unterstützt (beispielsweise Text oder XML). D.h. entweder man findet eine Bibliothek, die alle Datenarten aus den Anforderungen (Text, Bilddaten, Annotationen) direkt unterstützt, oder aber man muss entsprechende Konvertierungsmethoden schreiben. (etwas mehr herausstellen, dass man bei Verwendung einer Bibliothek, sich in die Bibliothek einarbeiten muss und dass eben evtl. Konverter zu schreiben sind, oder falls die Bibliothek nicht alle Anforderungen unterstützt, diese erweitert werden muss)

Außerdem muss man bei der Suche nach einer passenden Bibliothek beachten, unter welche Lizenz diese steht. Da es noch unklar ist, unter welcher Lizenz das TeleTeachingTool steht, ist es aber schwierig zu sagen, welche Lizenzen dazu kompatibel wären.

Ein möglicher Kandidat ist das *Extensible Messaging and Presence Protocol*, kurz *XMPP*. Es bietet die Möglichkeit der Echtzeit-Kommunikation und Erkennung der Anwesenheit von Kommunikationspartnern. Das Protokoll wurde im Zuge des Open-Source Projekts Jabber [www-jabber], einem Instant-Messaging Systems, entwickelt und mittlerweile von der IETF standardisiert. [www-xmpp]

Ein anderer Kandidat stammt aus dem Bereich der verteilten Anwendungen: *SOAP* (ursprünglich für *Simple Object Access Protocol*). SOAP setzt auch auf XML und bietet den Austausch von Informationen in verteilten Umgebungen an. Damit lassen sich Nachrichten versenden oder auch Remote Procedure Calls ausführen. SOAP kann über verschiedene Wege transportiert werden, die gängigste ist die Verwendung von HTTP über TCP. Das Protokoll ist eine W3C Empfehlung. [w3c-soap] [wiki-soap]

3.1.3 Eigenes Protokoll und Implementation

Als weiterer Weg bleibt die Entwicklung und Implementation eines eigenen Kommunikationsprotokolls. Da das Protokoll direkt auf die erforderlichen Bedürfnisse angepasst werden kann, können die in Kapitel 2.4 genannten Anforderungen an das Protokoll voll unterstützt werden. Vor allem auf die Anforderung, dass das Protokoll für SmartPhones/PDAs geeignet sein soll, kann eingegangen werden, indem das Protokoll möglichst einfach gestaltet wird und nur Techniken verwendet, die auch auf solchen Geräten verfügbar sind.

Da sich das Protokoll auf das Notwendigste, was ihm Rahmen dieser Arbeit gebraucht wird, beschränken kann, hat man keinen Overhead durch Protokollteile, die nicht benötigt werden. Und je kleiner der Umfang des Protokolls ausfällt, desto leichter fallen Implementationen des Protokolls, was für die spätere Ausweitung der unterstützten Geräte von Vorteil ist.

Ein Nachteil dieses Weges ist die Mehrarbeit, die nötig ist, um einen Parser für das Protokoll und die Netzwerkkommunikation zu implementieren. Allerdings muss man sich bei den anderen Wegen auch erstmal mit der jeweiligen API auseinandersetzen und überlegen, wie man die Kommunikation umsetzt.

Allerdings ist es natürlich so, dass je mehr Code man schreiben muss, desto mehr Fehler sich einschleichen können und man damit doch wieder mehr Zeit investieren muss, um diese zu finden. Allerdings können auch in Bibliotheken Fehler in enthalten sein. Eine Eigenentwicklung hat auch den Vorteil, dass man sich nicht mit Lizenzen, unter denen das Protokoll und die Bibliothek stehen, auseinandersetzen muss.

Nun stellt sich die Frage, in welcher Form man das Protokoll entwickelt. Entweder man setzt auf die binäre Übertragung oder man sendet Text, im speziellen XML.

Ein Binärprotokoll wäre platzsparender als ein textbasiertes Protokoll. Dadurch wäre auch die Netzwerklast geringer. Da aber nicht mit einer hohen Netzwerklast zu rechnen ist, kann dies vernachlässigt werden. Der Nachteil eines Binärprotokolls ist, dass es nicht (oder besser gesagt schwierig) menschenlesbar ist. Deshalb ist die Implementation und das Debuggen schwieriger und auch Erweiterungen sind schwieriger, da man ständig die Größen und die Bedeutung der einzelnen Felder nachschlagen muss. Für ein Binärprotokoll müsste man auch einen entsprechenden Parser schreiben.

Der Vorteil der Verwendung von XML ist, dass es menschenlesbar ist, sofern die XML-Tags und die Attribute aussagekräftige Namen haben. Es ist somit intuitiver, damit zu arbeiten und deshalb auch einfacher, das Protokoll zu erweitern. Da XML mittlerweile ein weitverbreitet genutzter Standard ist, gibt es eine große Auswahl an Parsern und es sollte sich für jede Platform einer finden lassen. Einen Liste von Java Bibliotheken zum Parsen von XML auf der J2ME-Platform findet sich auf [Knudsen].

Ein Nachteil von XML ist, dass es gegenüber einem binären Protokolls mehr Resourcen benötigt. Außerdem müssen Binärdaten zunächst umcodiert werden, damit sie innerhalb eines XML Dokuments enthalten sein können. Man kann dazu beispielsweise die Base64-Kodierung verwenden. [Josefsson]

3.2 Darstellung von Nachrichten im TTT

Gemäß den Anforderungen sollen die von den Hörern gesendeten Nachrichten zunächst nur vom Dozenten sichtbar sein. Wenn dieser dann auf den Inhalt einer Nachricht eingehen möchte, sollte er den Inhalt im TTT-Fenster darstellen können. Dies soll auch mit in die Aufzeichnung des Vortrags aufgenommen werden. In die Aufnahme der visuellen Daten fließen aber nur die Daten des VNC-Servers und die erstellten Annotationen ein. Das heißt, damit Informationen mit aufgezeichnet werden, müssen diese entweder in Annotationen oder in einen VNC-Datenstrom umgewandelt werden.

Die Nachrichten können drei unterschiedliche Informationstypen enthalten: Text, Bilder (Folien) und Annotationen. Die Darstellung von Annotationen und deren Aufzeichnung sind einfach zu realisieren, da es eben schon TTT-Annotationen sind und das TTT damit umgehen kann.

Schwieriger gestaltet sich dies aber bei den beiden anderen Typen. Das TeleTeachingTool unterstützt Text als Annotation nicht. Als ein möglicher Weg kommt die Umwandlung von Text in Freehand-Annotations in Frage. Freehand-Annotations sind die Annotationen, die der Dozent erstellt, wenn er mit dem Stift Text schreibt oder Freihandzeichnungen erstellt. Sie enthalten jeweils eine Liste an 2D-Koordinaten, die den Pfad der Linien beschreiben.

Die Idee dieses Ansatzes ist nun, dass man mit Hilfe der FreehandAnnotations den Text nachzeichnet. Dazu würde man eine Übersetzungstabelle erstellen, die für jedes mögliche Schriftzeichen die entsprechend benötigten FreehandAnnotations oder nur den Pfad enthält. Allerdings wäre die Erstellung dieser Tabelle sehr zeitaufwendig.

Ein anderer Weg wäre die Erweiterung des TeleTeachingTools um Text-Annotationen. Der Text wäre dann einfach als Feld enthalten und die TextAnnotation ist, wie die anderen Annotationen, für sich selbst für die Darstellung zuständig. Dadurch dass der Text als Text in die Aufzeichnung gelangt, könnte dieser auch in der Textsuche, die das TTT bietet, berücksichtigt werden.

Ein Vorteil der Aufnahme von TextAnnotationen in das TeleTeachingTool wäre, dass, wenn man die TextAnnotationen in die Zeichenfunktionen des Clients integriert, die Hörer auch Text auf Folien schreiben können. Das Schreiben von Freihandtext mittels der Maus ist nicht einfach und so könnten die Hörer ganz normal per Tastatur Text eingeben.

Auch das Problem der Übernahme von Bilddaten in die TTT-Aufzeichnung bedarf einer Lösung. Da das TTT schon an einer Stelle Bilddaten verarbeitet, nämlich bei der Darstellung und der Aufzeichnung der VNC-Daten, könnte man versuchen, dort anzusetzen. Ursprünglich stammen die Bilddaten auch aus diesem VNC-Datenstrom. Die Idee ist, dass man die Bilddaten zunächst in einen VNC-Datenstrom umsetzt, um diesen dann in die Verarbeitungsroutinen des TeleTeachingTools für die VNC-Verbindung einzuschleußen. Abbildung 3.1 auf Seite 32 zeigt das Schema der Idee. Allerdings würden Teile der Folie sofort verschwinden, wenn neue VNC-Daten von der Verbindung zum VNC-Server hereinkommen.

Um dies zu Verhindern müsste der TTT so erweitert werden, dass man im TTT umschalten kann, ob Daten der VNC-Verbindung sofort zur Darstellung gebracht werden. Erst wenn der Dozent die Nachricht des Hörers nicht mehr dargestellt haben möchte, soll das TTT wieder die Daten des VNC-Servers darstellen. Damit der Dozent sofort wieder eine aktuelle Darstellung bekommt, müssten die ignorierten VNC-Daten gepuffert werden, um dann nach Beendigung der Darstellung der Folie diese an den VNC-Client

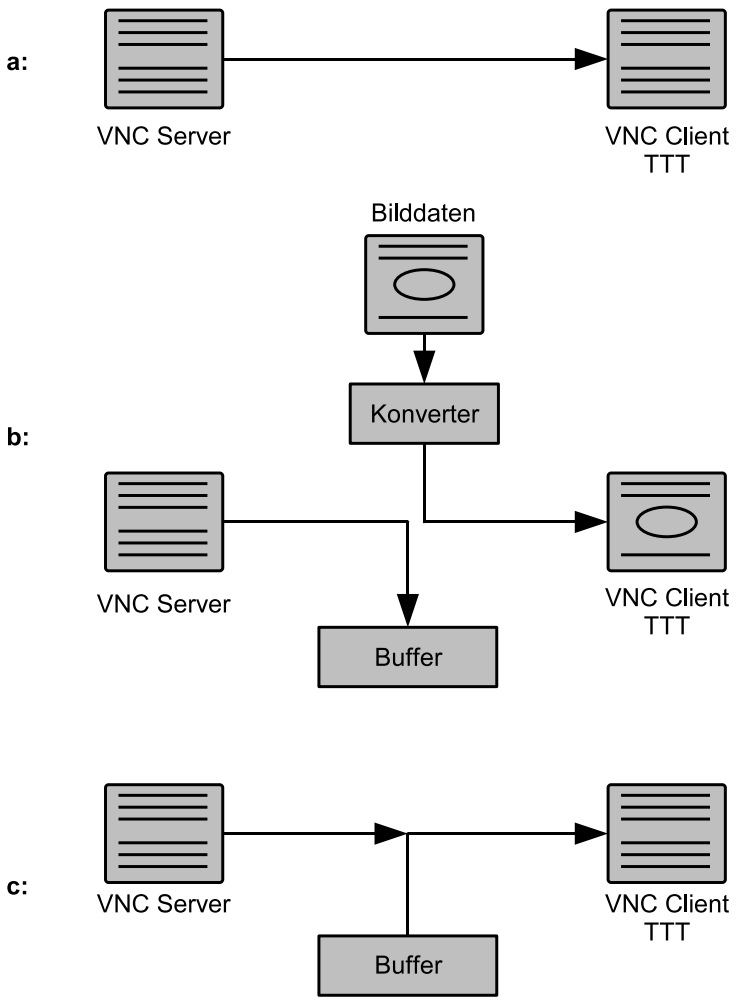


Abbildung 3.1: Verschiedene Zustände bei der Umsetzung von Bilddaten in VNC-Daten
a: normale Situation
b: während der Darstellung von Bilddaten
c: direkt nach Beendigung der Darstellung

im TTT zur Verarbeitung weiterzugegeben. Ob dieses Verfahren einsetzbar ist, hängt davon ab, ob man die Bilddaten so in VNC-Daten konvertieren kann, dass sich diese nahtlos in den VNC-Datenstrom des Servers einfügen lassen. Es könnte sein, dass neben den offensichtlich benötigten, statischen Informationen wie u.a. Bildgröße, Farbtiefe und gewählte Kodierungen, auch Informationen aus der Historie des Datenstroms notwendig sind, um Daten passend Umwandeln zu können. Beispiele dafür wären Sequenznummer oder dynamische Tabellen einer Entropiekodierung.

Analog zum zuvor erwähntem Informationstyp Text ist hier der alternative Weg die Erweiterung des TeleTeachingTools um ImageAnnotations. Die Bilddaten werden als Feld in der ImageAnnotation gespeichert.

3.3 Abstimmungen

Der Einsatz von Abstimmungen während eines Vortrages kann dem Dozenten interessante Informationen liefern. So könnte er überprüfen, ob er gewisse Teile in seinem Vortrag ausreichend erklärt hat, indem er dazu eine Multiple-Choice-Frage stellt. Oder er hätte gerne die Meinung zu einem Thema von den Hörern und erhält so eine genaue Übersicht über die Verteilung der Meinungen.

Damit nicht zuviel kostbare Vortragszeit verloren geht, sollte das Durchführen einer Abstimmung möglichst wenig stören (siehe auch die Anforderungen auf Seite 26). Das Eintippen von Frage und Antworten während des Vortrages kann schon zu langwierig sein. Man muss also Wege finden, damit eine Abstimmung schnell gestartet werden kann.

Eine Möglichkeit ergibt sich schon aus der Anforderung, dass eine Abstimmung geöffnet und geschlossen werden können soll. Der Dozent kann kurz vor Beginn seines Vortrages, nachdem er das TeleTeachingTool gestartet hat, schon die Abstimmungen eintippen, sie aber noch nicht zur Abstimmung freigeben. Erst sobald der entsprechende Punkt im Vortrag erreicht ist, gibt er sie frei.

Hat der Dozent aber die Angewohnheit, erst knapp vor Beginn seiner Vorträge zu erscheinen, ist dazu keine Zeit. Er möchte vielleicht schon weiter im Voraus, wenn er den Vortrag vorbereitet, die Abstimmungen erstellen. Diesem Wunsch kann man entsprechen, indem man dem Dozenten die Möglichkeit gibt, eine oder mehrere Abstimmungen als Datei abzuspeichern. Er kann die Datei zu seinem Vortrag mitnehmen, oder, falls er

die Abstimmungen auf dem Vortragsrechner vorbereitet hat, ist sie dort schon vorhanden, und kann dann die Datei während des Vortrags öffnen. Vor allem wenn der Dozent viele Abstimmungen durchführen möchte, ist dies zeitsparender als das Erstellen aller Abstimmungen während des Vortrags.

Beide genannten Punkte bringen aber nur etwas, wenn der Dozent schon zuvor weiß, dass er eine Abstimmung durchführen möchte. Wenn er jedoch während der Vorlesung den Einfall hat, eine starten zu wollen, muss er wieder die Frage und die Antworten eintippen. Aufwändig wird dies, wenn die Tastatur nich sofort zur Verfügung steht, wie das bei einem mit einem Eingabestift gehaltenen Vortrag durchaus sein kann. So könnte der Präsentationsrechner ein Convertible-Tablet-PC [wiki-convertible] sein, bei dem der Bildschirm nach oben zeigend auf die Tastatur geklappt wurde. Dann müsste der Dozent erst den Bildschirm in eine normale Laptop-Form zurückklappen, um die Tastatur bedienen zu können. Aber auch bei Verwendung eines normalen Rechners kann die Tastatur aufgrund von Platzmangel während des Vortrags verräumt sein. Man müsste also irgendwie auf die Eingabe per Tastatur verzichten können.

Dazu die folgende Idee: der Dozent schreibt mit dem Freehand-Tool die Fragen und Antworten in einer im TTT zur Verfügung stehenden Farbe als Freitext auf ein Whiteboard oder eine Folie. Beim Erstellen einer Umfrage gibt er dann nur noch die Farbe und die Anzahl der Antworten an. Die Hörer bekommen beim Abstimmen im Client die Farbe und einen Hinweis angezeigt, dass die Frage und Antworten auf der Videoprojektion zu sehen sind. Damit können Abstimmungen mit wenigen Klicks erstellt werden und tragen deshalb den Namen *QuickPolls*.

Ein weiterer Vorteil ist, das die Hörer dabei sehen, dass der Dozent eine Frage und die Antworten schreibt. Wenn er diese per Tastatur eintippt, sehen die Hörer, dass er tippt, aber nicht was, so dass dann dabei die Aufmerksamkeit der Hörer sinkt.

Auch das Vorbereiten von Abstimmungen im Vorfeld des Vortrags wird dadurch einfach. Der Dozent schreibt die Frage und die Antworten auf eine seiner Folien. Während des Vortrags markiert er diese Abstimmung farbig mit dem Highlight-Tool des TTT und wählt dann bei der Erstellung der Abstimmung die Farbe aus.

4 Implementation

Nach der Vorstellung der möglichen Varianten für die Implementation verschiedener Punkte im vorherigen Kapitel 3, stelle ich hier die Wege vor, für die ich mich entschieden habe. Um die Übersichtlichkeit zu steigern, werden Methoden sowohl im Text als auch in Abbildungen zumeist ohne ihre Parameter aufgeführt.

4.1 Kommunikationsprotokoll

Aus den in Abschnitt 3.1 erläuterten Wegen habe ich mich für die Implementation eines eigenen auf XML basierten Protokolls und der entsprechenden Netzwerkroutinen entschiedenen. Neben den dort angeführten Vorteilen sprach meine Erfahrung mit XML und der Implementation von Netzwerkprotokollen dafür.

4.1.1 XML Format

Die Nachrichten zwischen Hörer und Dozent werden als wohlgeformte XML-Dokumente gesendet. Das Root-Tag jeder Nachricht ist `<tttmessage>`. Für die laut Anforderungen geforderte Unterscheidung zwischen Befehls- und Inhaltsnachrichten besitzt das Root-Tag das Attribut `type`. Die möglichen Werte sind in Tabelle 4.1 aufgelistet.

Innerhalb von `tttmessage` sind dann je nach Nachrichten-Typ die entsprechenden Tags enthalten. Beim Typ `command` kann dies einer der in Tabelle 4.2 aufgeführten Befehle

`commmand`	Die Nachricht enthält (genau) einen Befehl.
`response`	Die Nachricht ist eine Antwort auf einen zuvor erhaltenen Befehl.
`content`	Die Nachricht ist eine Inhaltsnachricht, d.h. sie enthält einen Text, eine Folie und oder oder Annotationen

Tabelle 4.1: mögliche Werte des type-Attributs des tttmessage-Tags

Tag	Bedeutung
`<queryScreenSize />`	fragt nach der Größe des VNC-Bildschirms, damit der Client die Zeichenfläche entsprechend in der Größe beschränkt
`<getPolls />`	holt die aktuellen freigegebenen Abstimmungen
`<getCurrentSheet />`	holt den aktuellen Bildschirminhalt (Folie oder Whiteboard)
`<castVotes>`	enthält als Kinder die Stimmen des Users. Die einzelnen Stimmen werden so übermittelt: `<vote id="1" answer="2" />`

Tabelle 4.2: Befehls-Tags

Tag	Bedeutung
`<screenSize />`	gibt über die Attribute `width` und `height` die angefragte Größe des VNC-Bildschirms an
`<polls>`	enthält als Kinder die angefragten Abstimmungen

Tabelle 4.3: Antwort-Tags

sein. Zur Zeit werden nur Befehle vom Client zum Server gesendet. Dies ist aber keine Einschränkung des Protokolls, sondern ergab sich so.

Die Antworten auf die Befehle zeigt Tabelle 4.3. Die einzelnen Abstimmungen werden je nach Typ der Abstimmung (siehe Abschnitt 4.5) folgendermaßen übertragen:

```
<fullpoll id="1" text="Diese Vorlesung ist..." >
  <answer id="0" text="...langweilig" />
  <answer id="1" text="...interessant" />
  <answer id="2" text="...großartig!" />
</fullpoll>
<quickpoll id="2" color="4" answerCount="4" />
```

Nun zum dritten und letzten Typ: den Inhaltsnachrichten. Die Anforderungen legen fest, dass das Nachrichtenprotokoll die Übertragung von Text, Bilddaten und Annotationen unterstützen muss. Der Text wird einfach von einem `<text>`-Tag umschlossen und in die Nachricht eingefügt. Die Übertragung von Annotationen gestaltet sich auch recht einfach. Jeder Typ von Annotation wird durch ein Tag mit dem Namen der Klasse umgesetzt. Die

Eigenschaften der Annotationen werden als Attribute mit passenden Namen zum XML-Element hinzugefügt. Ein Beispiel für eine HighlightAnnotation:

```
<HightlightAnnotation color="8" startx="100" starty="140"
  endx="300" endy="260" />
```

Eine Besonderheit gibt es bei der FreehandAnnotation. Diese enthält den Pfad der Freihandlinie als Liste von 2D-Koordinaten. Dieser Pfad wird als eigenes XML-Element `path` als Kind der FreehandAnnotation in das XML aufgenommen. Die Koordinaten enthält dabei dessen Attribut `data` als eine mit Leerzeichen getrennte Aneinanderreihung der Werte. Auch TextAnnotations und ImageAnnotations (siehe Kapitel 4.3) haben Eigenschaften, die nicht als einfache Attribute abgebildet werden können. Bei TextAnnotations wird der Text an NewLine-Zeichen (`\n`) aufgesplittet und die einzelnen Textteile in `<line>`-Tags in das XML eingefügt. ImageAnnotations enthalten Bilddaten in binärer Form.

XML kann aber direkt keine Binärdaten enthalten, so dass die Binärdaten zunächst in eine textuelle Repräsentation konvertiert werden müssen. Dazu gibt es einige bekannte Algorithmen, wie die BaseXX-Algorithmen mit Base64 als bekanntesten und verbreitesten Vetreter und das aus früheren Tagen bekannte UUencode mit seinem Kodierungsverfahren. Ich habe mich dazu entschieden, Base64 dafür zu verwenden, da es wegen der Verwendung bei der Übertragung von Binärdaten über EMail weit verbreitet ist, und in einem RFC standardisiert ist. [Josefsson], [wiki-uuencode]

Es wurde auf die Definition und Verwendung eines Namespace verzichtet. Ein solcher wäre erst dann nötig, wenn die Naachrichten zusammen mit anderen XML-Formaten versendet oder verarbeitet werden.

Der XML-String einer Nachricht muss am Ende einen NewLine-Character (`\n`) besitzen. Dies ist wegen der Art des Einlesens der Nachrichten notwendig; siehe dazu auch Kapitel 4.1.3.

4.1.2 Implementation des XML Protokolls

Einige Anmerkungen zur XML-Verarbeitung in meiner Implementation. Das Parsen der XML-Nachrichten geschieht mit Hilfe des in Java 1.5 enthaltenen DOM-Parsers, d.h. das XML-Dokument wird im Speicher als Baum aufgebaut und steht dann zur Verarbeitung zur Verfügung. Ein DOM-Parser ist gegenüber einem Push- oder Pull-XML-Parser wie

z.B. der bekannte SAX-Parser Ressourcen hungriger, bietet aber einen angenehmeren Zugriff auf alle Teile des XML-Dokuments. Falls sich im realen Einsatz herausstellen sollte, dass bei einer großen Anzahl an verbunden und Nachrichten schreibenenden Hörern die Last auf dem Dozenten-Server zu groß wird, sollte man auf einen Push- oder Pull-Parser ausweichen (siehe auch Kapitel 5.1).

Das Erstellen der XML-Nachrichten geschieht nicht über eine XML-API, sondern schlicht durch Zusammenbauen der XML-Strings mittels *StringBuilder*-Instanzen von Java. Dies hat den Nachteil, dass man während des Programmierens aufpassen muss, die Wohlgeformtheit des XML-Dokuments einzuhalten, d.h. dass zu jedem öffnenden Tag auch das passende schließende Tag an richtiger Stelle vorkommt, dass die Werte der Attribute korrekt mit Anführungszeichen umschlossen sind und das in Daten, die von einem Benutzer stammen, die Zeichen, die in XML ausgezeichnet sind, durch ihre Symbolnamen (z.B. aus < wird <) ersetzt werden. Der Vorteil ist, dass man nicht zahlreiche API-Klassen verwenden muss, um den XML-String zu generieren, da Java 1.5 leider keine einfache Klasse zum Erstellen von XML-Strings bietet.

Für die Arbeit mit dem XML-Format weisen die Annotationen passende Methoden auf. Die XML-Repräsentation einer Annotation liefert deren Methode `toXMLString()` und ein Konstruktor, der als Parameter eine Referenz auf das XML-Element im DOM-Baum bekommt, erstellt aus dem XML wieder eine Annotation.

Leider ist in der offiziellen J2SE 1.5 API kein Base64-Codec enthalten, trotz wiederkehrender Nachfragen von Programmierern [sun-base64]. Es gibt aber Bibliotheken, die einen Base64-Codec enthalten, beispielsweise das Apache Commons Codec Package [apch-cdc]. Um aber möglichen Lizenzproblemen aus dem Weg zu gehen und als Programmierübung habe ich den Base64-Codec nach RFC4648 [Josefsson] selbst implementiert. Die Klasse `Base64Codec` bietet eine Methode an, um ein Byte-Array in einen Base64-String zu konvertieren und ein entsprechende Methode für den Rückweg.

4.1.3 Netzwerkkommunikation

Die Implementierung der Netzwerkkommunikation basiert auf einer Client-Server Programmierung mittels Sockets. Als Referenz zur Socket Programmierung unter Java diente [sun-tut-sckt].

Den Aufbau auf Seiten des Nachrichten-Clients zeigt das Klassendiagramm 4.1. Die Klasse `ClientConnection` besitzt eine Referenz auf den Socket, der die Verbindung zum Server bereitstellt. Der Socket wird zuvor vom `TTTMessengerClient`, der Hauptklasse

des Hörer-Clients, erstellt um gegebenfalls auf Fehler bei der Verbindungsaufnahme mit entsprechenden Dialog-Meldungen zu reagieren.

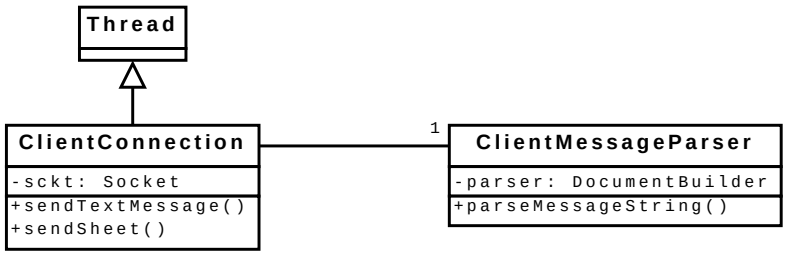


Abbildung 4.1: Client: Klassendiagramm der Netzwerkklassen

Zum Lesen vom Stream des Sockets wird ein `BufferedReader` eingesetzt. In einer Schleife werden die Zeilen der eintreffenden Nachrichten eingelesen und mit Hilfe eines `StringBuilder`s zusammengefügt. Diese Schleife läuft in einem eigenen Thread. Deshalb ist die Klasse `ClientConnection` eine Unterklasse von `Thread`. Die Erkennung, wann das Ende einer Nachricht erreicht wurde, erfolgt darüber, ob sich am Ende der letzten gerade gelesenen Zeile das schließende `</tttmessage>`-Tag befindet. Daher besteht auch die Bedingung im Nachrichtenformat, dass die Nachricht mit einem NewLine-Character abgeschlossen werden muss.

Sobald eine komplette Nachricht eingelesen wurde, wird der String an die Methode `parseMessage()` einer Instanz der Klasse `ClientMessageParser` übergeben. Diese enthält eine Instanz eines DOM-Parsers zur Verarbeitung des Strings. Falls die empfangene Nachricht eine Aktion auslösen soll, wird, nachdem aus der Nachricht passenden Java-Objekte erzeugt wurden, die entsprechende Methode im `ClientController` aufgerufen, die dann wiederum die notwendigen Schritte durchführt. Ein Beispiel: nachdem der Hörer die aktuelle Folie angefordert hat, kommt die entsprechende Nachricht vom Nachrichten-Server zurück. Aus dieser wird mit Hilfe des beim Parsen erstellen DOM-Baumes die Java-Objekte für das Bild und die Annotationen erstellt. Daraufhin wird dann die Methode `showSheet()` im `ClientController` aufgerufen.

Das Schreiben auf den OutputStream des Sockets geschieht über einen `PrintWriter`. Die Klasse `ClientConnection` bietet verschiedene öffentliche Methoden, um die unterschiedlichen Nachrichten zu versenden. Beispielsweise versendet `queryPolls()` die Nachricht zur Abfrage der aktuellen Abstimmungen oder `sendTextMessage()` schickt eine Textnachricht an den Dozenten. Dabei kümmert sich die Klasse auch um die Konvertierung

in XML-Strings, die dann mit Hilfe des PrintWriters über den Socket versendet werden.

Die serverseitige Klassenstruktur für die Netzwerkverbindung zeigt das Diagramm 4.2. Betrachtet man eine Verbindung zu einem Client, ist die Klassenstruktur genauso aufgebaut wie bei der Client-Seite. Die Klasse `TTTMessengerConnection` enthält die Referenz zum Socket, der InputStream wird wieder mittels eines `BufferedReader`s in einer Schleife, die in einem eigenen Thread läuft, zeilenweise eingelesen und sobald eine Nachricht vollständig eingelesen wurde, wird diese an einen `ServerMessageParser` zur weiteren Verarbeitung weitergegeben. Auch hier hat die Parser-Instanz eine Referenz auf einen Controller, in diesem Fall auf den `MessagingController`, der unter anderem die Methoden zum Anzeigen der empfangenen Inhaltsnachrichten und zum Holen von angeforderten Informationen oder Inhalten bereitstellt. Im Unterschied zum `ClientMessageParser` besitzt die Server-Variante ein Feld für die IP des Clients. Diese wird bei der Erstellung der Parser-Instanz übergegeben und dient zur Identifizierung der Clients. Sie wird bei Nachrichten, bei denen diese Unterscheidung notwendig ist, mit an den `MessagingController` übergeben. Dazu gehören Inhaltsnachrichten (Ignorieren des Hörers) und abgegebene Stimmen (nur eine Stimme pro Hörer).

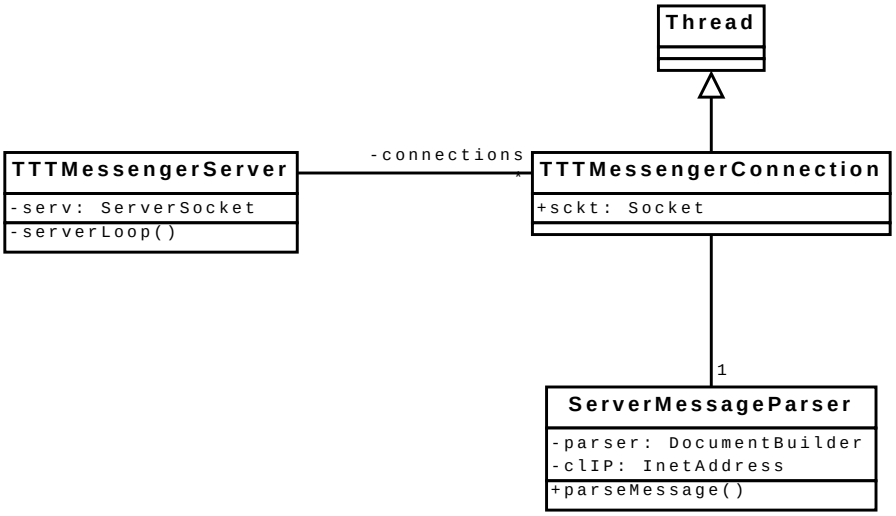


Abbildung 4.2: Server: Klassendiagramm der Netzwerkklassen

Damit sich an den Server nicht nur ein Client verbinden kann, wird pro verbundenem Client eine `TTTMessengerConnection` mit zugehörigem Thread und Parser erstellt. Dies

erledigt die Klasse `TTTMessengerServer`. Sie öffnet einen `ServerSocket` auf einem festgelegten Port und wartet in einer Schleife, die in einem eigenen Thread läuft, auf dort eingehende Verbindungswünsche. Wenn sich ein Hörer mit seinem Client verbindet, erstellt sie einen Socket zur Kommunikation mit diesem Client. Daraufhin wird eine neue Instanz einer `TTTMessengerConnection` erstellt und dabei der Socket übergeben. Die Verbindungen werden in einer Liste gespeichert, um sie bei Beendigung des Server schließen zu können.

4.2 Nachrichten-Server - Bedienung

Zur Darstellung der Bedienoberfläche des Nachrichten-Servers eigent sich am Besten der freie Raum neben dem TTT-Fenster und unterhalb der Speicherauslastungsanzeige (siehe Abbildung auf Seite 10). Dies hat auch gleich den Vorteil, dass die Oberfläche nicht für die Hörer in der Projektion sichtbar ist, da im Allgemeinen durch Einstellungen des Beamers nur das TTT-Fenster dargestellt wird. Da der freie Platz recht schmal ist aber in der Höhe doch einiges an Platz bietet, eignet sich zur Darstellung der eingegangen Nachrichten recht gut eine Liste.

Zunächst werden die eingehenden Nachrichten in Java-Objekte umgesetzt, je nach Art der Nachricht in `TTTTextMessage` oder `TTTSheetMessage`. Beide sind Unterklassen von `TTTMessage`, welche die gemeinsame Eigenschaften enthält. Dies sind der Status, ob die Nachricht zurückgestellt wurde und die IP-Adresse des Hörers, von dem die Nachricht stammt. `TTTTextMessage` besitzt zusätzlich nur noch den Text der Nachricht als String. Die Klasse für die Foliennachrichten muss hingegen mehr aufnehmen und zwar die Bilddaten der Folie, die Annotationen und ein Text, wobei nicht alle drei Daten vorhanden sein müssen. Für die Darstellung wichtig ist deren Methode `getThumbnail()`, die ein Vorschaubild in der übergebenen Breite berechnet. Dazu zeichnet die Methode intern erst ein Bild in normaler Größe und verkleinert es über die Methode `getScaledInstance()` der Klasse `ImageHelper`. Diese Methode ist schneller als die gleichlautende, die eine Instanz von Image bietet. Der Code stammt aus dem Artikel [Campbell (2007)].

Die erstellten Objekte sollen dann in einer Liste dargestellt werden. Java bietet dazu die Swing-Komponente `JList` an. Sie kann durch eigene Komponenten an verschiedene Bedürfnisse angepasst und in der Darstellung verändert werden.

Das Standard-Listenmodell einer `JList` ist nur für statische Listen gedacht, d.h. die Liste kann nach der einmaligen Erstellung nicht mehr geändert werden. Deshalb wurde ein

eigenes Listenmodell implementiert. Da außerdem unterschiedliche Threads (der Java-Event-Dispatcher sowie die verschiedenen Verbindungs-Threads) auf das Listen-Modell zugreifen, können gleich Vorkehrungen zur Synchronisierung vorgenommen werden. Das Diagramm in Abbildung 4.3 zeigt diese Klasse `TTTMessageListModel`. Sie erweitert `AbstractListModel`, welches die Verwaltung der Listener, die sich in ein `ListModel` eintragen können, übernimmt. Somit müssen nur noch die Methoden `getElementAt()` und `getSize()` implementiert werden, um es als Listenmodell verwenden zu können.

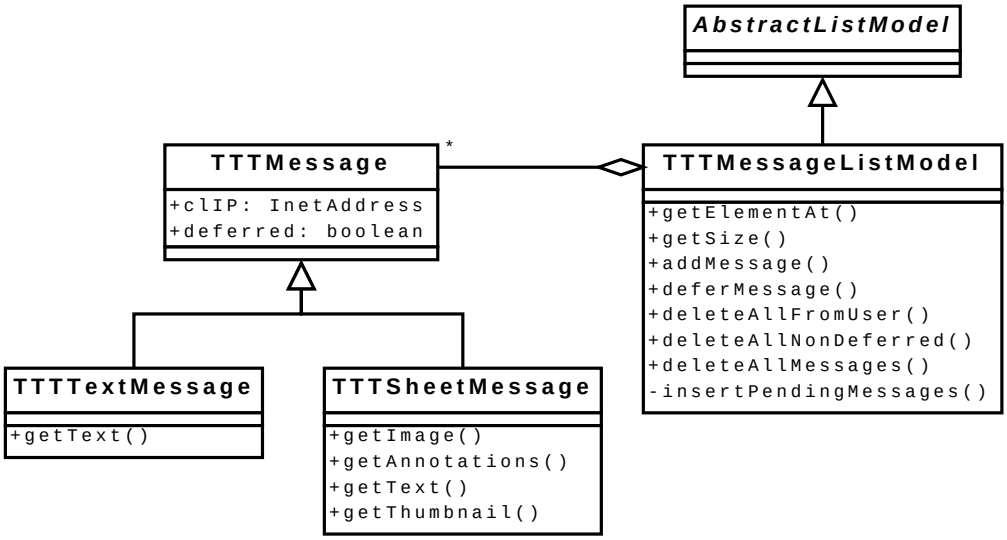


Abbildung 4.3: dsds

Als Grundlage verwendet `TTTMessageListModel` eine `ArrayList<TTTMessage>`. Damit können die beiden benötigten Methoden einfach umgesetzt werden. Desweiteren bietet `TTTMessageListModel` Methoden um die Liste verändern zu können: eine Methode zum Hinzufügen von Nachrichten, verschiedene Methoden zum Löschen aller oder einiger Nachrichten sowie eine Methode zum Zurückstellen einer Nachricht. Zurückgestellte Nachrichten werden an das Ende der Liste gesetzt und erhalten in der Darstellung eine Markierung.

Swing-Komponenten sollten nur über den Event-Dispatcher-Thread (kurz EDT) geändert werden. [Muller und Walrath (2000)] Die Methode `addMessage()` wird als einzige der Klasse von anderen Threads als dem Event-Dispatcher-Thread aufgerufen und zwar von den Verbingungsthreads. Deshalb beinhaltet die Methode eine Abfrage, ob der aktuelle Thread der EDT ist. Falls nicht wird die einzufügende Nachricht erst in die Liste

`toBeAddedList` eingetragen und dann per `SwingUtilities.invokeLater()` der Aufruf der Methode `insertPendingMessages()` auf die Event-Warteschlange gelegt. Wenn diese Methode dann vom EDT aufgerufen wird, werden die zwischengespeicherten Nachrichten in die eigentliche Nachrichtenliste aufgenommen.

Die Darstellung der einzelnen Nachrichten geschieht über einen eigenen `ListCellRenderer`, der als interne Klasse in `JTTTMessageList` enthalten ist (`MessageCellRenderer`). Über die Methode `getListCellRendererComponent()` des `ListCellRenderers` fragt die Komponente, die die Liste zeichnet (`BasicListUI`), für jeden einzelnen Eintrag eine `JComponent` ab, die dann zur Darstellung benutzt wird. Diese zurückgelieferten Referenzen auf ein JComponent-Objekt werden aber nicht zwischengespeichert, sondern immer sofort verwendet. Die Methode `getListCellRendererComponent()` wird in zwei unterschiedlichen Kontexten aufgerufen. Im ersten werden die Abmessungen der zurückgelieferten `JComponent`s als Höhe (und im Falle von zweidimensionalen Listen auch als Breite) der einzelnen Listeneinträge von der UI-Komponente zwischengespeichert (im folgenden „Höhendurchlauf“ genannt). Der andere Kontext ist, wenn die Liste gezeichnet wird. Dann wird direkt nach Aufruf von `getListCellRendererComponent` die `paint()`-Methode der zurückgelieferten `JComponent`-Instanz aufgerufen, so dass der Listeneintrag gezeichnet wird (im folgenden „Zeichendurchlauf“ genannt).

Der `MessageCellRenderer` liefert eine Instanz von `MessageListTextPanel` bei Textnachrichten und eine Instanz von `MessageListSheetPanel` zurück. Beide sind Unterklassen von `MessageListPanel`, die neben gemeinsamen Eigenschaften auch noch Zeichenmethoden für gemeinsame Teile der Darstellung enthält. So gibt es eine Methode, die einen String mit automatischen Umbrüchen und ggf. mit einer Begrenzung der Zeilenanzahl zeichnet (`drawTextLines`) oder eine, die eine Trennung zwischen den einzelnen Listenelementen zeichnet (`drawMessageSeparator()`).

Bevor der CellRenderer eine Instanz zurückliefert, setzt er die Attribute dieser Instanz mit den Informationen über das aktuelle Listenelement, damit sie den Listeneintrag zeichnen kann. Danach ruft er `calculateHeight()` auf, damit die `MessageList*Panel`-Instanz die benötigte Höhe ermittelt, so dass diese für die UI-Komponente der Liste zur Verfügung steht.

Den Anforderungen entsprechend wird die Darstellung der Nachrichten zunächst auf einen Teil beschränkt. Bei den Textnachrichten sind dies zwei Textzeilen, bei Foliennachrichten ungefähr die Hälfte der Folie. Die Größe dieses sofort sichtbaren Teils ist jeweils als Konstante in den beiden Klassen festgelegt. Damit der Dozent auch erkennt, dass nur ein Teil der Nachricht angezeigt wird, werden im unteren Bereich zwei kleine, nach unten zeigende, Dreiecke gezeichnet. Um die gesamte Nachricht zu sehen, kann der Dozent auf

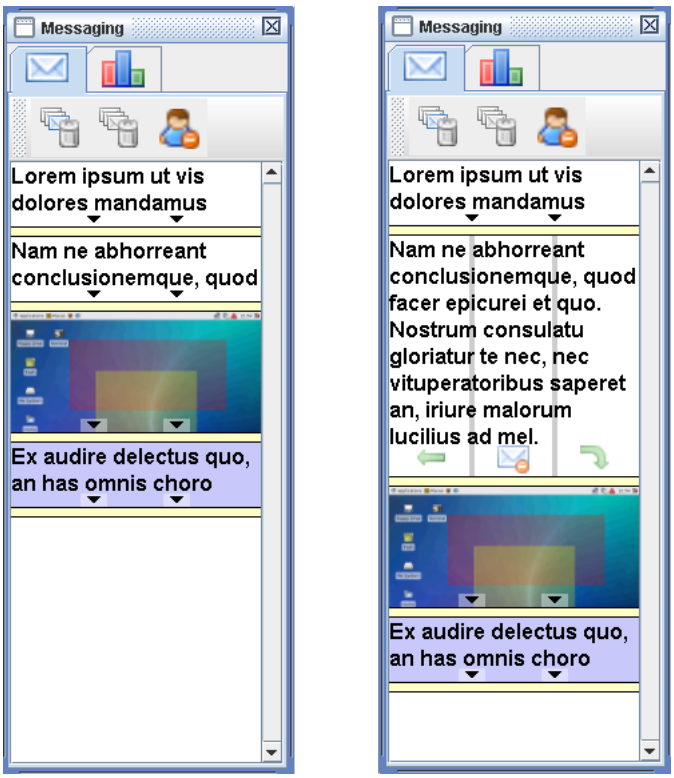


Abbildung 4.4: Darstellung von empfangenen Nachrichten

die Nachricht klicken, die dann nach unten „aufgeklappt“ wird. Außerdem werden dann drei Klickbereiche mit jeweils einem transparenten Icon sichtbar, über die der Dozent Aktionen aufrufen kann. Dies sind von links nach rechts (siehe Abbildung 4.4) die Anzeige der Nachricht im TTT-Fenster, das Löschen der Nachricht und das Zurückstellen der Nachricht. Eine zurückgestellte Nachricht wird wie schon erwähnt, an das Ende der Liste gestellt und sie erhält in der Darstellung bei Textnachrichten einen bläulichen Hintergrund und bei Foliennachrichten einen bläuchlichen Rahmen.

Beim Zeichnen des Textes muss dieser an der rechten Seite der Liste automatisch umbrochen werden. Wie man dies in Java umsetzt, ist bei der Implementation der TextAnnotationen beschrieben, da man dort dies auch benötigt (siehe Abschnitt 4.3.3 auf Seite 49).

Bei der Implementierung der bisher aufgeführten Merkmale mussten einige Hindernis-

se überwunden werden. So wird im Höhendurchlauf im Gegensatz zum Zeichendurchlauf die Breite der durch den `CellRenderer` zurückgelieferten `JComponent` nicht auf die Listenbreite gesetzt. Damit kann natürlich nicht die Höhe korrekt berechnet werden. Deswegen wurde `MessageListPanel` um das Attribut `listWidth` erweitert, das im `MessageCellRenderer` vor Aufruf von `calculateHeight()` auf die Breite der Liste gesetzt wird. Damit steht dann ein Wert zur Berechnung der Höhe des Listeneintrags zur Verfügung.

Ein weiterer problematischer Punkt ist das Zwischenspeichern der Höhen durch die UI-Komponente. Da man die Bedienoberfläche für den Nachrichtenserver und damit auch die Liste in ihrer Größe ändern kann, müssen die Einträge ihre Größe ändern können. Zieht man die Liste schmäler, benötigt ein Text mehr Zeilen zur Darstellung. Dadurch dass `BasicListUI` aber die Höhen der Zellen nicht neu einlist, wenn die Liste in ihrer Größe verändert wird und der Listeneintrag nur in dem durch die Höhe festgelegten Bereich zeichnen darf, wird der untere Teil des Textes abgeschnitten. Deshalb musste ein Weg gefunden werden, damit die Höhen neu eingelesen werden. Der erste Ansatz war der Aufruf von `updateUI()`, wenn die Liste in der Größe geändert wird. Dies war aber nur ein schlechter Workaround, da dabei jedesmal eine neue Instanz von `BasicListUI` erstellt wird und damit zwangsläufig die Höhen neu eingelesen werden. Außerdem führte dieser Ansatz zu einer NullPointerException in einer Methode von `BasicListUI`, wenn man eine Nachricht aus der Liste löschte.

Somit musste ein anderer und möglichst besserer Weg gesucht werden. Dieser fand sich in der API-Dokumentation zur Klasse `BasicListUI` in Gestalt der Methode `updateLayoutState()`. [sun-javaapi] Diese liest die Höhen neu ein, ist aber leider *protected*, und kann deshalb nicht aus `JTTTMessageList` aufrufen werden. Somit wurde eine Unterklasse `MyBasicListUI` erstellt, die als einzige eigene Methode `triggerHeightUpdate()` hat, in der einfach nur `updateLayoutState()` aufgerufen wird. Eine Instanz davon wird als eigene UI-Komponente der Liste gesetzt und in einer privaten Variable abgelegt, so dass man darüber das Neueinlesen der Höhen anstoßen kann.

Über einen `ComponentListener` wird nun bei Änderung der Größe der Liste die Methode `triggerRepaint()` aufgerufen, die zunächst das Einlesen der Höhen anstößt und danach noch `repaint()` der Liste aufruft, damit die geänderten Höhen auch dargestellt werden.

Probleme bereitete auch die Funktion, dass bei Selektion eines Listeneintrags dieser komplett dargestellt wird. Einerseits müssen natürlich wieder die Höhen aktualisiert werden, da dies bei Änderung der Selektion nicht automatisch geschieht. Dazu wird der Liste ein `ListSelectionListener` hinzugefügt, der dann bei Selektionsänderungen

`triggerRepaint()` aufruft. Andererseits stellte sich heraus, dass im Höhendurchlauf bei Aufruf der Methode `getListCellRendererComponent()` der Parameter `isSelected` immer den Wert *false* hat, egal ob der Listeneintrag selektiert ist oder nicht. Da sich aber die Höhe eines Eintrags zwischen selektiert und nicht selektiert unterscheidet, ist diese Information für die Berechnung der Höhe essentiell. Als Umweg wird nun über den Aufruf von `getSelectedIndex()` der Liste der Index des aktuell selektierten Eintrags abgefragt und mit dem Index des gerade behandelten Listenelements verglichen. Falls identisch, ist der Listeneintrag selektiert und das Attribut in `MessageList*Panel` wird entsprechend gesetzt.

Die Umsetzung der Funktionalität der Klickbereiche erfolgt über das Überschreiben der `processMouseEvent()`-Methode. Damit diese aufgerufen wird, muss sie zunächst über den Aufruf `enableEvents()` mit einem passend gesetzem Parameter aktiviert werden. In dieser Methode werden nur einfache Klicks mit der linken Maustaste (entspricht einmal Tippen mit einem Eingabestift) behandelt. Welches Listenelement sich unterhalb des Klicks befindet, erfährt man über die Methode `locationToIndex()`, die den Index des betroffenen Elements zurückliefert. Leider gibt diese Methode auch den Index des letzten Elements zurücks, wenn unterhalb von diesem geklickt wird. Diese Aktion soll aber dazu führen, dass kein Element selektiert ist. Deshalb werden über die Methode `getCellBounds()` die Abmessungen des über `locationToIndex` ermittelten Elements geholt und noch einmal überprüft, über die Koordinaten des Mouse-Events dort drin liegen. Falls nicht, war der Klick unterhalb des letzten Elements und die Selektion wird aufgehoben, ansonsten wird die vom Benutzer getroffene Aktion ausgeführt.

Weitere Aktionen kann der Dozent über eine Leiste von Knöpfen oberhalb der Liste ausführen: Löschen von Nicht-zurückgestellten Nachrichten, Löschen aller Nachrichten und Ignorieren eines Hörers. Ganz oben hat der Dozent die Möglichkeit zwischen den Nachrichten und den Abstimmungen (siehe Abschnitt 4.3). Dies wird durch eine `JTabbedPane` umgesetzt. Auch möglich wäre die Verwendung von `CardLayout` gewesen, allerdings erschließt sich durch die Karteikartenreiterdarstellung der `JTabbedPane` eher, dass man zwischen zwei Inhalten umschalten kann.

Die in der Oberfläche des Servers dargestellten Icons stammen von Mark James und stehen unter einer Creative-Commons-Lizenz. [James]

4.3 ImageAnnotation & TextAnnotation

In Abschnitt 3.2 wurden für Text und Bildinhalte jeweils zwei Lösungswege für die Darstellung von Nachrichten im TTT-Fenster vorgestellt. Jeweils einer davon war die Erweiterung des TTT um eine entsprechende Annotation. Ich habe mich für diesen Weg entschieden, da es zum einen der konzeptionell bessere Weg ist. Zum anderen sind sowohl die Konvertierung von Bilddaten in VNC-Daten als auch die Umsetzung von Text in FreehandAnnotations aufwendig zu implementieren.

4.3.1 Konzept

Zunächst wird erläutert, was die Text- und ImageAnnotations können sollen. ImageAnnotations sollen dazu eingesetzt werden, Folien aus empfangenen Nachrichten im TTT-Fenster darstellen zu können. Sie müssen also die Folie als Bild und die Position, an der dieses dargestellt werden soll, enthalten. Um die Netzlast und Übertragungszeiten gering und die TTT-Aufnahmen klein zu halten, sollten die Bilder in den ImageAnnotations in einer komprimierten Form übertragen bzw. abgespeichert werden. Damit wiederholtes Kodieren und Dekodieren vermieden wird, soll eine einmal durchgeführte Komprimierung beibehalten werden. Wenn beispielsweise eine Inhaltsnachricht mit einer Folie vom Server empfangen wird, enthält diese Nachricht das Bild schon in komprimierter Form. Die daraus erstellte ImageAnnotation soll diese komprimierten Bilddaten einfach in einem Feld abspeichern, so dass diese dann beim Speichern der Aufnahme verwendet werden können.

TextAnnotations sollen Text an einer bestimmten Stelle in einer der TTT-Zeichenfarben darstellen. Somit braucht die TextAnnotation entsprechende Felder für Position, Farbe und natürlich den Text. Um den Text besser sehen zu können, wenn dieser auf anderen Bildinhalten liegt, soll hinter den Text ein halbtransparenter Hintergrund gelegt werden. Damit läßt sich der Folieninhalt unter dem Text noch erkennen, der Text wird aber mehr vom Folieninhalt abgesetzt.

TextAnnotations kommen in zwei unterschiedlichen Kontexten vor: zum einen beim Darstellen von Text im TTT-Fenster, der aus empfangenen Textnachrichten oder aus der Darstellung von Abstimmungsergebnissen stammt. Zum anderen im Nachrichten-Client, wenn der Benutzer eine TextAnnotation auf einer Folie oder einem Whiteboard erstellt.

Im ersten Fall muss ein automatischer Zeilenumbruch geschehen, falls ein Text zu lang für die Darstellung in einer Zeile im TTT-Fenster ist. Dazu muss der TextAnnotation mitgeteilt werden, wie breit sie ist oder maximal sein darf, damit sie entsprechend das Layout des Textes berechnen kann. Im zweiten Fall möchte man dem Benutzer die Freiheit geben, manuell die Umbrüche festzulegen, damit der Text u.U. passend zur darunterliegenden Folie gestaltet werden kann. Wenn man zum Beispiel links neben einer Zeichnung einen Kommentar platzieren möchte, dann kann der Benutzer vor erreichen der Zeichnung selbst eine neue Zeile beginnen, um so zu verhindern, dass die Zeichnung verdeckt wird.

Somit muss eine TextAnnotation zwei Modi unterstützen: zum einen das automatische Auslegen des Textes innerhalb einer festgelegten Breite und zum anderen das Zeichnen des Textes mit manuellen Zeilenumbrüchen. Unterschieden werden diese Modi dadurch, ob eine maximale Breite für den Text angegeben ist.

4.3.2 Anforderung seitens des TTTs an Annotationen

Neue Annotationen müssen Unterklasse von `ttt.messages.Annotation` sein, die wiederum selbst Unterklasse von `ttt.messages.Message` ist. Eine neue Annotation muss demnach folgende Methoden implementieren:

Rectangle getBounds() liefert die Abmessungen der Annotation zurück

boolean contains(int x, int y) überprüft, ob die Annotation die übergebene Koordinate enthält; wird beim Löschen von Annotationen aufgerufen

int getEncoding() gibt eine eindeutige Nummer des Typs der Annotation zurück; die Nummern werden als Konstanten in der Klasse `Constants` verwaltet.

paint(Graphics2D g) zeichnet die Annotation

int getSize() gibt die Anzahl der Bytes an, die die Annotation beim Speichern benötigt

write(DataOutputStream out, int writeTimestamp) schreibt die Annotation auf den übergebenen Datenstrom

writeToFlash(FlashContext ctxt) setzt die Annotation in Flash Objekte um; wird bei der Konvertierung von Aufnahmen in das Flash-Format verwendet

String toXMLString() gibt das XML-Element, dass diese Annotation beschreibt, zurück; wird beim Nachrichtenversand verwendet

Außerdem müssen noch zwei Konstruktoren implementiert werden: einer mit einem `int` und einem `DataInputStream` als Parameter, der zum Erstellen der Annotation aus den Binärdaten einer Aufnahme dient. Und einer mit einem `org.w3c.Element` als Parameter, der eine Annotation aus ihrer entsprechenden XML-Repräsentation erstellt.

Die `toXMLString()`-Methode und der Konstruktor mit dem XML-Element sind nicht ursprünglicher Bestandteil des TTT, sondern die Erweiterungen für die Nachrichtenübermittlung im Rahmen dieser Arbeit.

4.3.3 Implementation

Hier wird nun beschrieben, wie die jeweiligen Konzepte und die eben genannten Anforderungen in die Klassen `ImageAnnotation` und `TextAnnotation` umgesetzt wurden.

Die Klasse ImageAnnotation besitzt neben den beiden `int` Feldern für die Position ein Feld `imgData`, das die Daten des komprimierten Bildes speichert. Da das Flash-Interface nur mit JPG-Dateien umgehen kann, kommt auch nur diese Komprimierung in Frage. Daneben hat die Klasse noch das Attribut `bImage`, welches das Bild als `BufferedImage` aufnimmt, damit es sofort zum Zeichnen zur Verfügung steht.

Die binäre Repräsentation besteht aus zwei `short`s für die Position, einem `int` für die Anzahl der Bytes, die das komprimierte Bild hat und dann die Daten des Bilds in ihrer komprimierten Form. Der Konstruktor zum Einlesen und die Methode zum Schreiben sind entsprechend aufgebaut. In der XML-Repräsentation enthalten die Attribute `posx` und `posy` die Position und das Attribut `data` die Bilddaten, wobei diese zuerst mittels des Base64-Codecs in eine in XML erlaubte Form gebracht werden. Die Implementation der `writeToFlash`-Methode basiert auf dem Code der bereits bestehenden Annotationsklassen und dem Beispiel für das Einfügen von Bildern auf der Webseite der Java-Flash-Bibliothek. [www-javaswf1] Der Rest der Klasse ist schnell implementiert; `contains()` und `getBounds()` berechnen sich aus Position und Größe des Bildes, und in der `paint()`-Methode reicht ein Aufruf um das Bild zu malen.

Dagegen ist dieser Teil bei der TextAnnotation aufwändiger zu implementieren. Der Modus mit den manuellen Zeilenumbrüchen ist noch recht einfach. Man setzt im Graphics-Objekt den Font, mit der der Text gezeichnet werden soll und ruft dann pro Zeile `g.drawString()` auf. Damit sich die Textzeilen nicht überlappen oder zu weit auseinander stehen, benötigt man die Höhe einer Zeile des eingesetzten Fonts. Dazu stellt Java die

Klasse `FontMetrics` zur Verfügung, die zahlreiche Maße liefern kann, u.a. über die Methode `getHeight()` die Höhe einer Zeile. Eine Instanz dieser Klasse für einen bestimmten Font bekommt man über das `Graphics`-Objekt mittels `getFontMetrics(Font)`. [sun-tut-font]

Im Modus, bei dem eine maximale Breite festgelegt wurde, müssen die Positionen der Zeilenumbrüche im Text berechnet werden, damit man die einzelnen Zeilen zeichnen kann. Dazu bietet Java die Klasse `LineBreakMeasurer` an. Sie benötigt zum Berechnen der Zeilenumbrüche den Text in Form eines `AttributedCharacterIterator`s. Dazu erstellt man zunächst aus dem Text einen `AttributedString`, dem man noch die benötigten Font-Attribute übergeben muss, und ruft dann die Methode `getIterator()` auf. Damit kann man dann mit Hilfe des `LineBreakMeasurer`s in einer Schleife die einzelnen Textteile berechnen und diese gleich zeichnen. [sun-tut-line]

Da die Abmessungen abhängig vom Text und dessen Layout sind, berechnet die Methode `calculateBounds()` die Größe des Textes und legt diese im privaten Feld `bounds` ab, so dass sie für die Methoden `contains()` und `getBounds()` zur Verfügung stehen. Sie ist ähnlich zur `paint`-Methode aufgebaut nur mit dem Unterschied, dass natürlich die Funktionsaufrufe zum Zeichnen wegfallen. Obwohl die Abmessung auch nach dem Zeichnen zur Verfügung stehen würden, ist eine eigene Methode zur Berechnung notwendig, da `getBounds()` noch vor dem allerersten Zeichnen aufgerufen wird. Außerdem muss die Zeichenmethode den halbtransparenten Hintergrund noch vor dem Text malen, so dass auch dort schon die Abmesseungen bekannt sein müssen.

Die unterschiedlichen Repräsentationen sind alle recht einfach aufgebaut. Die binäre besteht aus einem `byte` für die Farbe, drei `short`s für die Position und den Wert der maximalen Breite, einem weiteren `short` für die Anzahl der Bytes des Textes und schließlich die Daten des Textes selbst in UTF-8 Kodierung. Das XML-Element `TextAnnotation` hat die entsprechenden Attribute für Farbe, Position und Breite. Der Text selbst wird an den eventuell enthaltenen Zeilenumbrüchen (`\n`) aufgeteilt und jeweils als XML-Element `<line>`, dass den jeweiligen Textteil enthält, als Kind der `TextAnnotation` hinzugefügt.

Das Umsetzen der TextAnnotation in Flash richtet sich auch hier nach dem entsprechenden Beispiel auf der Webseite der Java-Flash-Bibliothek [www-javaswf2]. Zunächst muss man eine Font-Definition für die Schriftart, die man verwenden möchte, aus einem Flash-File laden. Daraus kann man dann ein `Font`-Objekt erstellen, das man wiederum für die `Text`-Objekte für die einzelnen Zeilen benötigt. Leider ließ sich der Modus mit der maximalen Breite und dem automatischen Umbruch nicht in Flash umsetzten. Meine Versuche, den automatischen Zeilenumbruch darüber zu erreichen, den Text

in ein `EditField` mit passender Größe zu schreiben, scheiterten daran, dass der dem `EditField` zugewiesene Text beim Abspielen der exportierten SWF Datei nicht sichtbar war. Eine andere Möglichkeit wäre noch, dass man die Aufteilung des Textes auf Zeilen aus der Java-Zeichnen-Methode übernimmt. Da sich die in Java und Flash für die Darstellung verwendeten Schriftarten unterscheiden, würde dies aber auch nicht immer zu zufriedenstellenden Ergebnissen führen.

Außerdem bietet die Klasse TextAnnotation noch einige Methoden, die für das Erstellen von TextAnnotation durch einen Benutzer notwendig sind. Mehr dazu in Abschnitt 4.3.5

4.3.4 notwendige Anpassungen TTT

Das alleinige Hinzufügen der beiden neuen Annotations-Klassen in das `ttt.messages`-Package reicht nicht aus, damit der TTT mit diesen vollständig arbeiten kann, sondern es müssen noch an ein paar Stellen im Code des TTT Erweiterungen vorgenommen werden. In der Klasse `Constants` müssen eindeutige Nummern für die Annotationen als Konstanten erstellt werden, die diese dann bei Aufruf von `getEncoding()` zurückliefern. Desweiteren enthält die Klasse die Methode `encodingToString()`, die zu einer Numer den Namen einer Message als String zurückliefert. Hier muss man die `switch`-Anweisung passend erweitern.

Die statische Methode `readMessage()` der Klasse `Message` liest von einem Datenstrom einer Live-Übertragung oder einer Aufnahme und setzt die an der aktuellen Position enthaltene Message in das entsprechende Objekt um. Deshalb müssen in die dortige `switch`-Anweisung die neuen Annotationen mit ihren Datenstrom-Konstruktoren eingetragen werden.

Abschließend gibt es noch in der Klasse `Messages` zwei Stellen, an denen man die neuen Annotationen mit Einfügen muss. Zum einen in der Methode `readMessages()` in eine `switch`-Anweisung, zum anderen in der Methode `statistics()` in ein Array.

Da die Erweiterung des TTT um neue Annotationen nirgends dokumentiert ist, wurden die obigen Stellen durch Suche nach dem Vorkommen von Konstanten, Aufrufen von `getEncoding()` und Aufrufen von Konstruktoren bereits vorhandener Annotationen gefunden.

4.3.5 Text-Werkzeug im Nachrichten-Client

Nach der Erweiterung des TTT mit TextAnnotationen kann es den Hörern auch ermöglicht werden, Text mit ihrer Tastatur auf Folien zu schreiben. Dazu benötigen sie aber ein entsprechendes Werkzeug in der Werkzeugleiste im Nachrichten-Client.

Die Bedienung des TextAnnotation-Werkzeugs unterscheidet sich etwas von der Bedienung der anderen Annotationswerkzeuge. Letztere werden durch Drücken der linken Maustaste, ziehen der Maus zur Erstellung der gewünschten Form und Loslassen der Maustaste erstellt. Da es aber unpraktisch ist, während des Eintippens von Text die linke Maustaste zu halten, muss sich das Erstellen etwas anders abspielen.

Nach Auswahl des TextAnnotation-Werkzeugs klickt der Benutzer einmal an die Stelle, an die er Text schreiben möchte. Beim Drücken der Maustaste wird wie bei den anderen Werkzeugen die `set()`-Methode aufgerufen und in diesem Fall eine TextAnnotation an dieser Position erstellt. Dabei wird noch eine Variable (`textMode`) gesetzt, die anzeigt, dass eine Texteingabe stattfindet. Beim Loslassen der Maustaste wird jedoch nicht wie bei den anderen Annotationen `finishPainting()` aufgerufen. Der Benutzer hat nun ein leere TextAnnotation erstellt, in die er nun Text eingeben kann. Die Methode `processKeyEvent` behandelt die auftretetenden Tastaturereignisse und fügt die eingegebenen Zeichen an die TextAnnotation an (`TextAnnotation.addChar(c)`). Wenn der Benutzer die Backspace-Taste drückt, dann wird das letzte Zeichen der TextAnnotation gelöscht (`TextAnnotation.deleteLastChar()`).

Wenn der Hörer nun ein zweites Mal auf die Zeichenfläche klickt, wird `finishPainting()` aufgerufen, die `textMode`-Variable wieder zurückgesetzt und damit das Erstellen der TextAnnotation abgeschlossen. Sollte der Benutzer keinen Text eingegeben haben, dann wird die TextAnnotation wieder gelöscht.

Damit der Benutzer einen visuellen Hinweis darauf hat, wann er Text in eine TextAnnotation schreiben kann und wann er die Eingabe beendet hat, besitzt die TextAnnotation einen roten Rahmen solange sie noch in Bearbeitung ist. Im Code ist dies so umgesetzt, dass ein rotes Rechteck gezeichnet wird, wenn das `temporary`-Flag gesetzt ist.

4.4 Anbindung Nachrichten Server an TTT

Der Nachrichten-Server, der beim Dozenten im TTT läuft, benötigt vom TTT-System einige Informationen, wie z.B. die aktuellen Annotationen, um diese den Clients schicken zu können. Außerdem muss er Nachrichten oder Ergebnisse von Abstimmungen an das TTT-Fenster übergeben können, so dass diese dargestellt und in die Vortragsaufzeichnung übernommen werden können.

Im einzelnen werden folgenden Funktionalitäten seitens des Servers vom TTT benötigt:

1. Holen der aktuellen Annotationen, die der Dozent erstellt hat; benötigt, für den Versand von annotierten Folien und Whiteboard-Zeichnungen an Hörer
2. Holen des aktuellen Bildschirminhalts der VNC-Sitzung; benötigt für den Versand von Folien an Hörer
3. Abfrage, ob im Moment der Whiteboard-Modus aktiv ist; falls er aktiv ist, reicht es aus, dass der Server nur die Annotationen an Hörer schickt
4. Abfrage der aktuellen Bildschirmgröße der VNC-Sitzung; Clients fragen diesen Wert ab, damit diese die Zeichenfläche für die Hörer entsprechend beschränken
5. Hinzufügen von Annotation zu den aktuell im TTT-Fenster dargestellten, und zwar so, dass diese auch mit in die Vortragsaufzeichnung aufgenommen werden; benötigt für die Darstellung von Nachrichten und Abstimmungsergebnissen
6. Aktivierung des Whiteboard-Modus mit einem (neuen) leeren Whiteboard; benötigt für die Darstellung von Nachrichten mit Folien und bei der Darstellung von Abstimmungsergebnissen

Nun müssen die Stellen im TeleTeachingTool gesucht werden, an denen die entsprechenden Daten vorliegen oder wo die an den TTT zu übergebenden Daten verarbeitet werden können. Je nachdem, ob schon passende Methoden vorhanden sind, können diese verwendet werden, andernfalls muss der TTT entsprechend erweitert werden.

Für Punkt 1 wurde zunächst die Stelle im Programmmcode des TTT gesucht, an der die aktuellen Annotationen aufbewahrt werden. Da die aktuellen Annotationen vom TTT dargestellt werden, wurde zunächst nach der Klasse gesucht, die die Annotation

im TTT-Fenster zeichnet, da diese auf jeden Fall eine Referenz auf die Annotationen besitzen muss. Ausgehend von der `paint(Graphics)`-Methode der Annotationen und der Suche nach Klassen, die diese aufrufen, wurde als Ergebnis die Klasse `GraphicsContext` gefunden. Diese besitzt eine `ArrayList<Annotation>`, in der sich die aktuellen Annotationen befinden.

Es existiert sogar schon eine öffentliche Methode `getCurrentAnnotations()`, diese liefert aber nur die Referenz auf die Liste zurück. Da die Verarbeitung von Nachrichten von Hörern und damit im speziellen die Anfrage nach der aktuellen Folie und den Annotationen in einem eigenen Thread läuft, sollte man mit dieser Referenz auf die Liste nicht arbeiten. Es würde zu einer `ConcurrentModificationException` kommen, wenn gerade in einem Thread über alle Elemente der Liste iteriert wird und die Liste aber gleichzeitig durch einen anderen Thread verändert wird, was zum Beispiel der Fall wäre, wenn der Dozent gerade eine neue Annotation erstellt hat, während über die Annotationsliste iteriert wird, um die XML-Repräsentationen zu erstellen.

Deswegen benötigt man eine neue Methode, die eine Kopie der Liste zurückliefert. Die Methode `getCurrentAnnotationsAsArray()` übernimmt diese Arbeit; sie liefert eine Kopie der Annotationsliste als Array zurück. Da aber immer noch die Bearbeitungsmethoden und die Kopieerstellung von unterschiedlichen Threads aufgerufen werden, müssen diese synchronisiert werden. Dazu werden die Code-Teile in der Klasse `GraphicsContext`, die mit der Liste arbeiten, in `synchronized(currentAnnotations)`-Blöcke gesetzt, damit immer nur ein Thread gleichzeitig auf die Liste zugreifen kann. (siehe Listing 4.1)

```
public Annotation[] getCurrentAnnotationsAsArray() {
  Annotation[] annots = null;
  synchronized(currentAnnotations) {
    annots = new Annotation[currentAnnotations.size()];
    currentAnnotations.toArray(annots);
  }
}
```

Listing 4.1: Kopieerstellung der Annotationliste

Um sicherzustellen, dass die Liste nur innerhalb der Klasse `GraphicsContext` bearbeitet wird, habe ich die Methode `getCurrentAnnotations()` auskommentiert und die einzige Stelle, an der diese Methode aufgerufen wird, mit Hilfe der neuen Methode `getCurrentAnnotationsAsArray()` umgeschrieben.

Beim Durchsehen der Klasse `GraphicsContext` bieten sich gleich noch weitere Methoden für andere Punkte an. Den aktuellen Bildschirminhalt (Punkt 2) kann man sich mit `getScreenShotWithoutAnnotations()` holen. Und über `isWhiteboardEndabled()` (der Tippfehler ist so im Code des TTT enthalten) läßt sich abfragen, ob gerade der Whiteboard-Modus aktiv ist (Punkt 3). An die Größe der VNC-Sitzung kommt man indirekt über die öffentliche Variable `prefs` des `GraphicsContext`. Diese verweist auf eine Instanz der Klasse `ProtocolPreferences` mit den öffentlichen Feldern `framebufferWidth` und `frameBufferHeight` (Punkt 4).

Für Punkt 5 musste ich herausfinden, wie die Annotationen in die Aufnahme gelangen, da über die Methode `addAnnotation()` der Klasse `GraphicsContext` die Annotation zwar angezeigt wird, aber nicht mit in die Aufzeichnung übernommen wird. Dazu fand ich zunächst die Stelle, an der die Annotationen instanziiert werden, wenn der Dozent eine erstellt, indem ich nach Aufrufen des Konstruktors der `FreehandAnnotation` suchte. Dies geschieht in der Klasse `PaintListener`. Solange der Dozent noch dabei ist eine Annotation zu zeichnen, wird diese zunächst in einer Variablen der Klasse gespeichert. Sobald dieser seine Eingabe beendet hat, wird die Methode `finishPainting()` aufgerufen, die die wiederum die Methode `writeMessage()` mit der Annotation als Parameter aufruft und in der Oberklasse `RFBKeyAndMouseListener` definiert ist. Dort wird die Methode `handleMessage()` einer Instanz der Schnittstelle `MessageConsumer` aufgerufen. Um nun Herauszufinden, welches Objekt dies ist, musste man nur noch nach dem Aufruf des Konstruktors für `PaintListener` suchen, da dort die Referenz auf den `MessageConsumer` übergeben wird. Dieser fand sich in einem der Konstruktoren der Klasse `Player` und dort wird eine Referenz auf eine Instanz der Klasse `RfbProtocol` an `PaintListener` übergeben. Wenn man sich nun die `handleMessage()`-Methode von `RfbProtocol` ansieht und noch ein bißchen die weiteren Aufrufe verfolgt, dann erkennt man, dass diese Methode genau die gesuchte ist. Man übergibt ihr als Parameter eine Annotation und diese wird in die Darstellung mit aufgenommen und in die Aufzeichnung mit eingefügt. `RfbProtocol` ist eine Unterklasse von `GraphicsContext`, somit würde der Nachrichten-Server für die Punkte 1 - 5 bisher nur eine Referenz auf die Instanz von `RfbProtocol` benötigen.

Um einen Weg zu finden, ein neues Whiteboard erstellen zu können betrachtet man sich zunächst die Abläufe bei Bedienung der Whiteboard-Knöpfe in der Werkzeugleiste im TTT-Fenster. Bei allen drei Knöpfen wird beim Drücken dieselbe `actionPerformed()`-Methode eines anonymen `ActionListener`s in der Klasse `PaintListener` aufgerufen. Dort wird über eine `if-else-if`-Kette die Quelle der Aktion herausgefischt. Im Falle unserer drei Knöpfe wird dann eine der Methoden `toggleWhiteboard()`, `previousWhiteboard()` und `nextWhiteboard()` aufgerufen. Alle drei Wege führen dazu, dass

eine `WhiteboardMessage` erstellt wird und an die Darstellung und die Aufnahme weitergeleitet wird. Die Idee, diese `WhiteboardMessage`s selbst zu erstellen und dann durch die vorhin gefundene Methode `handleMessage()` verarbeiten zu lassen, scheitert an dem Seitensystem der Whiteboards (siehe 1.3), da man die Nummer des nächsten freien Whiteboards benötigt, um dieses Anzeigen lassen zu können. Die Nummern werden aber nur im `PaintListener` verwaltet.

Die eigentlich gesuchte Funktionalität, ein leeres Whiteboard zu bekommen, ist auch in `PaintListener` noch nicht vorhanden. Ich habe die Klasse deshalb zunächst um die Variable `nextFreeWhiteboardNumber` erweitert, die die Nummer des nächsten Whiteboards speichert, das noch nicht benutzt wurde. Diese wird in den `enableWhiteboard()`-Methoden aktuell gehalten. Zusätzlich implementierte ich die Methode `newWhiteboardPage()`, die einfach `enableWhiteboard()` mit der nächsten freien Nummer aufruft.

Damit der Nachrichten-Server nur einen Zugriffsweg auf den TTT benötigt, wäre es gut, wenn die Klasse `RfbProtocol` eine Methode `newWhiteboard()` bekommt, so dass die Instanz dieser Klasse für alle Belange des Nachrichtenserver an den TTT ausreicht. Dazu erweitert man die Klasse `RfbProtocol` um eine private Variable des Typs `PaintListener` zusammen mit einer enstprechenden `set`-Methode. Im Konstruktor der Klasse `Player`, der für das Starten einer Dozentensitzung des TTT zuständig ist, wird dann, nachdem dort die Instanzen von `RfbProtocol` und `PaintListener` erstellt wurden, die set-Mehode aufgerufen. `RfbProtocol` besitzt dann eine Referenz auf den `PaintListener`, so dass die `newWhiteboard()`-Methode in ersterem die entsprechende Methode in letzterem aufrufen kann.

Damit bleibt nur noch die Frage übrig, wie der Nachrichten-Server an die Instanz von `RfbProtocol` kommt. Die Instanz wird im Konstruktor von `Player` erstellt, der wiederum von einer Methode der Klasse `TTT` beim Erstellen der Dozentensitzung aufgerufen wird. Die Klasse `TTT` enthält aber auch die Methode zum Starten des Nachrichten-Servers mit seiner Oberfläche. Somit kommt man zum Ziel, wenn man die `Player`-Klasse um eine Variable und eine dazugehörige get-Methode für die `RfbProtocol`-Instanz erweitert und dann die in der `TTT`-Klasse erstellte Player-Instanz wiederum in einer Variable hinterlegt. Aus letzterer kann somit die Methode zum Starten des Nachrichten-Servers die `Player`- und darüber auch die `RfbProtocol`-Instanz holen und dem Nachrichten-Server übergeben.

Abschließend nochmal eine Übersicht über die gefunden oder neu erstellten Methoden in `RfbProtocol`, die für die einzelnen Punkte gefunden wurden:

1. Holen der aktuellen Annotationen über `getCurrentAnnotationsAsArray()`

2. Holen des aktuellen Bildschirminhaltes: `getScreenShotWithoutAnnotations()`
3. Abfrage, ob gerade ein Whiteboard dargestellt wird: `isWhiteboardEndabled()`
4. Bildschirminhaltsgröße: über die öffentliche Variable `prefs`
5. Hinzufügen von Annotationen in Darstellung und Aufnahme: `handleMessage()`
6. Leeres Whiteboard: `newWhiteboard()`

Als sehr nützlich zur Suche stellten sich die von der Entwicklungsumgebung Eclipse [www-eclipse] bereitgestellten Funktionen „Open Call Hierarchy“ und „Open Declaration“ sowie die Suche nach Referenzen heraus.

4.5 Abstimmungen

In diesem Abschnitt wird gezeigt, wie das Durchführen und die Verwaltung von Abstimmungen im Nachrichten-Server umgesetzt wurde. In Abschnitt 3.3 wurden zwei Wege aufgezeigt, um dem Dozenten die Arbeit mit Abstimmungen zu vereinfachen. Die Idee mit den `QuickPolls`, bei der der Dozent die Fragen und Antworten farbig schreibt oder markiert und dann nur noch eine Abstimmung mit der passenden Farbe und der Anzahl der Antworten angibt, wurde von mir implementiert.

Hingegen fiel die Entscheidung gegen die Implementierung des Abspeicherns von Abstimmungen in einer Datei. Damit der Dozent von dieser Funktion profitiert, muss er sich vor dem Vortrag Zeit nehmen, die Abstimmungen zu erstellen. Dann muss er auch noch daran denken, die Datei mit zur Vorlesung zu nehmen oder irgendwo so abzulegen, dass er während des Votrags vom Präsentationsrechner aus zugreifen kann. Und dann muss er auch noch zum Öffnen der Datei durch den Dateisystembaum navigieren. Insgesamt auch nicht gerade wenig Bedienschritte. Dagegen steht der recht hohe Implementationsaufwand. Der Dozent würde einen extra Dialog im TTT benötigen, mit der er einfach und übersichtlich seine Abstimmungen erstellen kann. Der extra Dialog ist nötig, da der Nachrichten-Server nur gestartet werden kann, wenn eine Dozentensitzung aktiv ist. Der Dozent möchte aber eine solche nicht extra starten, nur um Abstimmungen vorzubereiten. Dann wären noch die Methoden zum Speichern und wieder Laden der Abstimmungen nötig. Alles in allem nichts schweres, aber doch eben einiges zu implementieren. Somit bietet diese Idee nicht genügend Bedienungsvorteile, wenn man den Implementierungsaufwand gegenüberstellt.

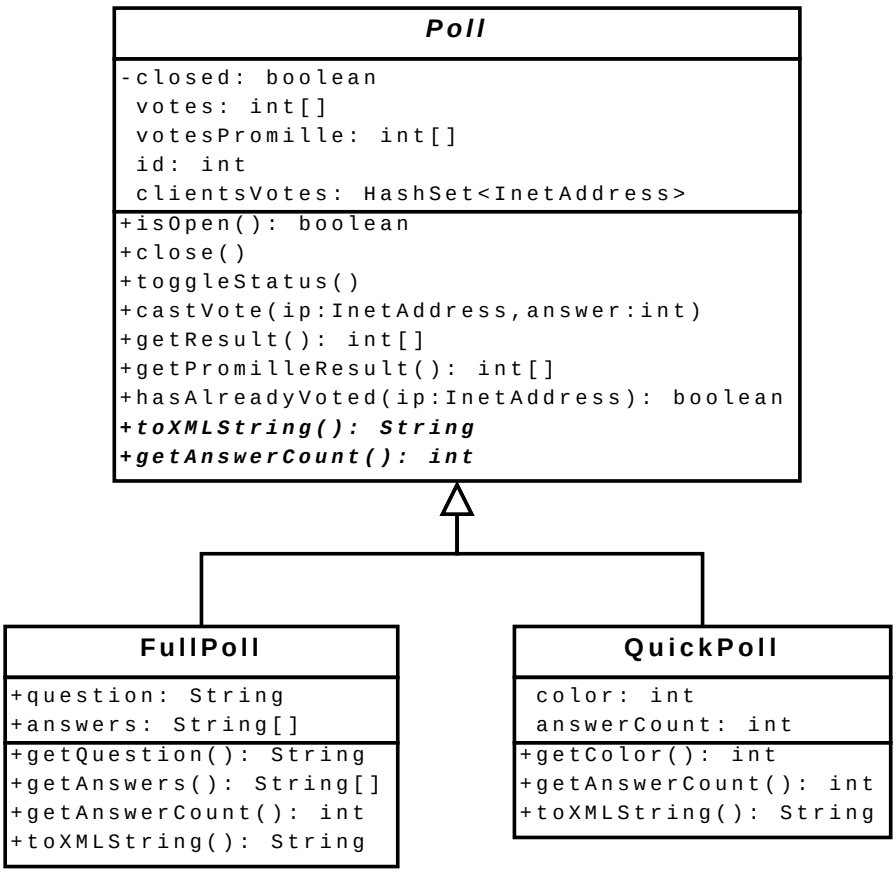


Abbildung 4.5: UML-Klassendiagramm der Abstimmungsklassen

Damit müssen die normalen Abstimmungen (`FullPolls`) und die QuickPolls umgesetzt werden. Beides sind Abstimmungen, womit sie gemeinsame Eigenschaften besitzen, was sich natürlich auch entsprechend in die Klassenstruktur umsetzt. Das Klassendiagramm 4.5 zeigt die Oberklasse `Poll` mit ihren Attributen und (öffentlichen) Methoden. Jede Abstimmung bekommt eine eindeutige Nummer zugewiesen, wobei diese in der jetzigen Implementierung der Position in der Liste der Abstimmungen entspricht. Die Nummer dient zur Identifikation der unterschiedlichen Abstimmungen, wenn Hörer ihre Stimmen abgeben.

Für die Anforderung (s. Abschnitt 2.4), dass die Abstimmungen geöffnet und geschlossen werden können, existiert die private boolesche Variable `closed` und dazu gehörige

Methoden zum Ändern und zur Abfrage des Status. Das Feld `votes` nimmt die Anzahl der Stimmen je Antwort auf, wobei Stimmen über die Methode `castVote()` abgegeben werden. Dabei wird dann auch das Feld `votesPromille` aktualisiert, dass die Anteile der Stimmen je Antwort an der Gesamtstimmenanzahl aufnimmt. Dies dient dazu, dass bei einer graphischen Anzeige der Umfrageergebnisse die Länge der Ergebnisbalken schneller berechnet werden kann (siehe auch Abschnitt weiter unten).

Das `HashSet<InetAddress> clientsVoted` dient zur Umsetzung der Anforderung, dass jeder Hörer nur einmal eine Stimme je Abstimmung abgeben kann. Dazu wird die IP des Hörers in das HashSet eingetragen, wenn er seine Stimme abgibt. Bei Abgabe einer Stimme wird überprüft, ob die IP schon im HashSet enthalten ist und gegebenenfalls die Stimme ignoriert. Außerdem kann über die Methode `hasAlreadyVoted()` abgefragt werden, ob ein Hörer schon an dieser Abstimmung teilgenommen hat. Dies dient dazu, dass nur Abstimmungen an den Hörer gesendet werden, an denen er noch nicht teilgenommen hat. Zusätzlich legt `Poll` auch noch zwei Methoden `getAnswerCount()` und `toXMLString()` fest, die aber erst in den beiden Unterklassen implementiert werden können.

Die Darstellung und die Verwaltung der Abstimmungen soll analog zu den Nachrichten in einer Liste erfolgen. Zum einen ist dann eine einheitliche Bedienbarkeit gegeben, zum anderen kann man bei der Implementierung der Liste vom Code der Nachrichtenliste abschreiben. Für genaue Details zur Implementation der Liste sei deshalb auf den Abschnitt 4.2 verwiesen.

Hier heißt die Klasse, die `JList` erweitert, `JPollList` und sie ist genauso wie die Klasse für die Nachrichtenliste aufgebaut. D.h. es wird ein eigenes ListModel erstellt, ein eigener CellRenderer gesetzt und die MouseEvents werden für die Bedienung der Liste behandelt.

Die Darstellung der Abstimmungen in der Liste erfolgt demnach auch wieder über Objekte, die `JPanel` erweitern und die dessen `paint()`-Methode überschreiben. Für die beiden unterschiedlichen Arten an Abstimmungen existieren zwei Klassen `FullPollPanel` und `QuickPollPanel`, wobei beide Subklassen von `PollPanel` sind. `PollPanel` beinhaltet verschiedene Methoden und Konstanten zum Zeichnen einzelner Teile der Abstimmungen, die von beiden Unterklassen verwendet werden. So zum Beispiel `drawResultBar()`, das einen Balken mit einer Länge entsprechend der Stimmenanzahl zeichnet, oder `drawTextLines`, welche einen String mit automatischen Umbrüchen zeichnet.

Bei `FullPolls` werden zunächst die Frage, und dann nacheinander die Antworten zusammen mit ihren Balken und der Stimmenzahl dargestellt. Bei QuickPolls wird nur

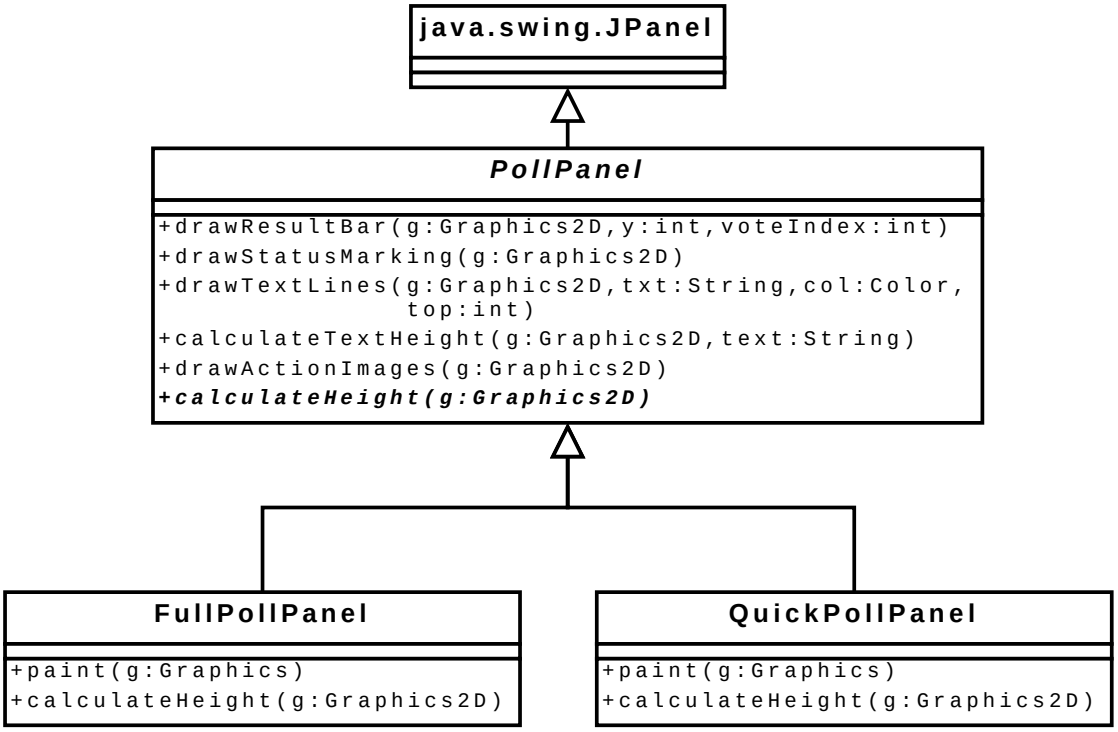


Abbildung 4.6: Klassen zum Zeichnen der Abstimmungen in der Abstimmungsliste

„QuickPoll“ und die Balken mit den Stimmenzahlen und zusätzlich als Hintergrund die zum QuickPoll gehörende Farbe dargestellt. Um erkennen zu können, ob eine Abstimmung geöffnet oder geschlossen ist, wird um diese entweder ein roter (geschlossen) oder grüner (offen) halbtransparenter Rahmen gezeichnet. (siehe Abbildung 4.7 auf Seite 61)

Das Listenmodell zur `JPollList` hat neben den beiden nötigen Methoden für das `ListModel`-Interface weitere. Diese werden vom `MessagingController` zur Verwaltung der Abstimmungen aufgerufen, wie z.B. `createNewQuickPoll()` zum Erstellen einer neuer QuickPoll oder `toggleStatus()` zum Ändern des Status einer Abstimmung.

Zur Verwaltung der Nachrichten stehen in einer Werkzeugliste zwei Knöpfe zur Erstellung der beiden Abstimmungsarten zur Verfügung. Zusätzlich erscheinen bei der Selektion einer Abstimmung in der Liste wie bei der Nachrichtenliste „Klickbereiche“ mit Icons. Hier sind es zwei Klickbereiche, wobei der rechte zum Öffnen und Schlie-

ßen dient und der linke Bereich die Abstimmung mit ihrem Ergebnis im TTT-Fenster anzeigt.

FullPolls werden im TTT-Fenster auf einem neuen leeren Whiteboard angezeigt. Die Fragen, Antworten und Zahlen werden in `TextAnnotations` umgesetzt, die Balken als `HightlightAnnotations`. Etwas anders ist die Darstellung von QuickPolls. Dort werden die Balken und die Stimmenzahlen im unteren Bereich der aktuellen Folie eingefügt. Dies hat den Hintergrund, dass das Abstimmungsergebnis auf der selben Folie oder demselben Whiteboard stehen kann, wie die Frage. Der Dozent muss sich aber beim Schreiben der Frage und der Antworten auf den oberen Teil beschränken, so dass im unteren Teil genügend Platz für die Balken bleibt. Bild 4.7 zeigt ein Beispiel.

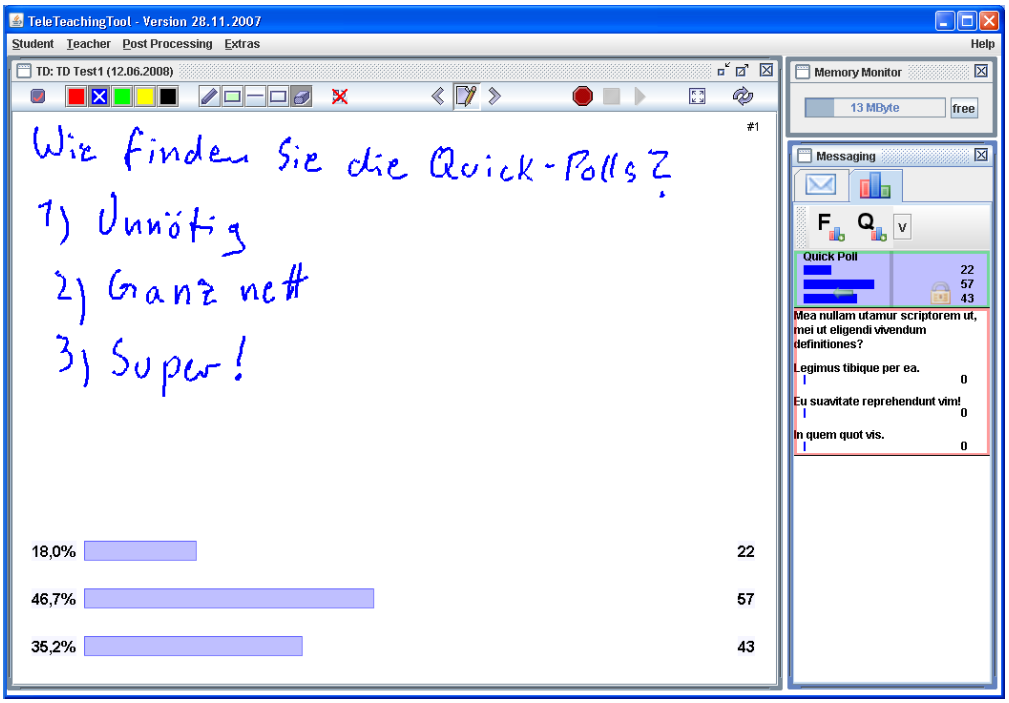


Abbildung 4.7: Anzeige des Ergebnisses eines QuickPolls

Eine Funktion zum Löschen von Abstimmungen wurde nicht implementiert, da eine solche Funktion nicht unbedingt nötig ist. Es ist unwahrscheinlich, dass die Gesamtanzahl der in einem Vortrag durchgeführten Abstimmungen, jemals zweistellig sein wird, so dass man nicht durch Löschen von bereits durchgeführten Abstimmungen eine Übersichtlich-

keit herstellen müsste. Zum anderen basiert die ID einer Abstimmung, wie oben bereits erwähnt, auf der Position der Abstimmung in der Abstimmungsliste. Würde man nun Abstimmungen löschen, würden die IDs der bereits bestehenden Abstimmungen zwar nicht verändert, da diese nur beim Erstellen einer Abstimmung gesetzt wird. Aber die ankommenden Stimmen würden den falschen Abstimmungen zugeordnet, da sich darauf verlassen wird, dass die ID mit der Position übereinstimmt. Wenn doch das Löschen von Abstimmungen seitens der Dozenten gewünscht wird, dann müsste man die ID durch eine Zufallszahl ersetzen und im `JPollList`-Model noch eine HashMap haben, in der die Zuordnung von ID zu Abstimmung mitgeführt wird.

4.6 Nachrichten-Client

Die Oberfläche des Nachrichten-Clients sollte für den Hörer möglichst selbsterklärend und übersichtlich sein. Laut den Anforderungen muss er Text-Nachrichten versenden, Folien und Whiteboards mit eigenen Annotationen erweitern und diese dann wieder versenden und an Abstimmungen teilnehmen können.

Zunächst war für die Benutzeroberfläche des Clients eine Aufteilung in vier Bereiche mittels Karteikarten vorgesehen. Diese Bereiche hätten den vier unterschiedlichen Anforderungen entsprochen: Versenden von Text-Nachrichten, Annotieren und Wiederversenden von Dozenten-Folien und -Whiteboards, Erstellen und Versenden von eigenen Whiteboard-Zeichnungen und als viertes das Mitwählen bei Abstimmungen.

Dies wurde aber zugunsten einer besser bedienbaren Oberfläche fallen gelassen. Es reicht eine Zeichenfläche aus, auf der der Hörer Annotationen erstellen kann. Ob dies nun ein selbst erstelltes Whiteboard ist oder ob er auf einer Folie des Dozenten Annotationen erstellt, hängt nur davon ab, ob er sich die aktuelle Folie holt, oder er diese wieder verwirft, damit er ein leeres Whiteboard bekommt. Das Texteingabe-Fenster kann auch recht klein sein, da die Hörer keine Romane schreiben sollen, sondern eher kurze Fragen stellen sollen. Da man mit der aktuellen Oberfläche die Zeichenfläche und das Texteingabefenster gleichzeitig sieht, kann man dem Benutzer auch die Möglichkeit geben, Text und eine Folie oder ein Whiteboard zusammen in einer Nachricht zu versenden.

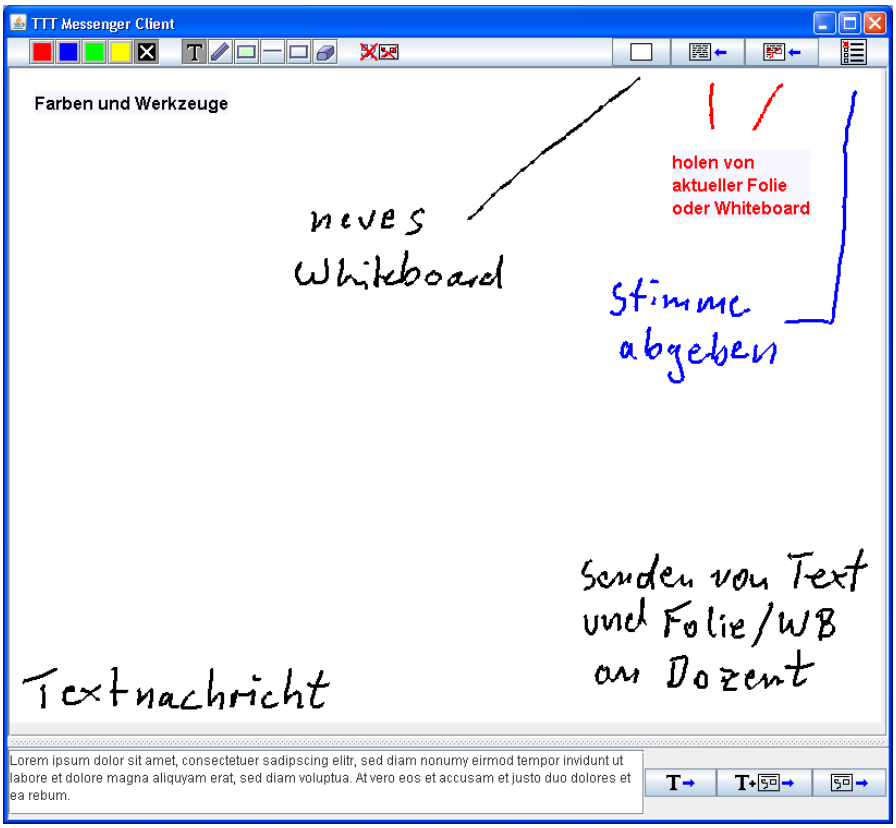


Abbildung 4.8: Oberfläche des Nachrichten-Clients

4.6.1 Zeichenfläche

Da eine Zeichenfläche für Annotationen benötigt wird, und eine solche schon im TTT vorhanden ist, wurde im Code des TTT nach den passenden Stellen gesucht, um diese in den Client übernehmen zu können.

Im TTT ist die bereits bekannte Klasse `GraphicsContext` für das Zeichnen der Annotationen zuständig. Sie ist Unterklasse von `JComponent` und überschreibt zum Zeichnen die `paintComponent()`-Methode. Das Zeichnen der Annotation geschieht recht einfach, indem für jede Annotation deren `paint()`-Methode aufgerufen wird.

Im Nachrichten Client wurde einen Klasse `JAnnotationPanel` erstellt, die auch Un-

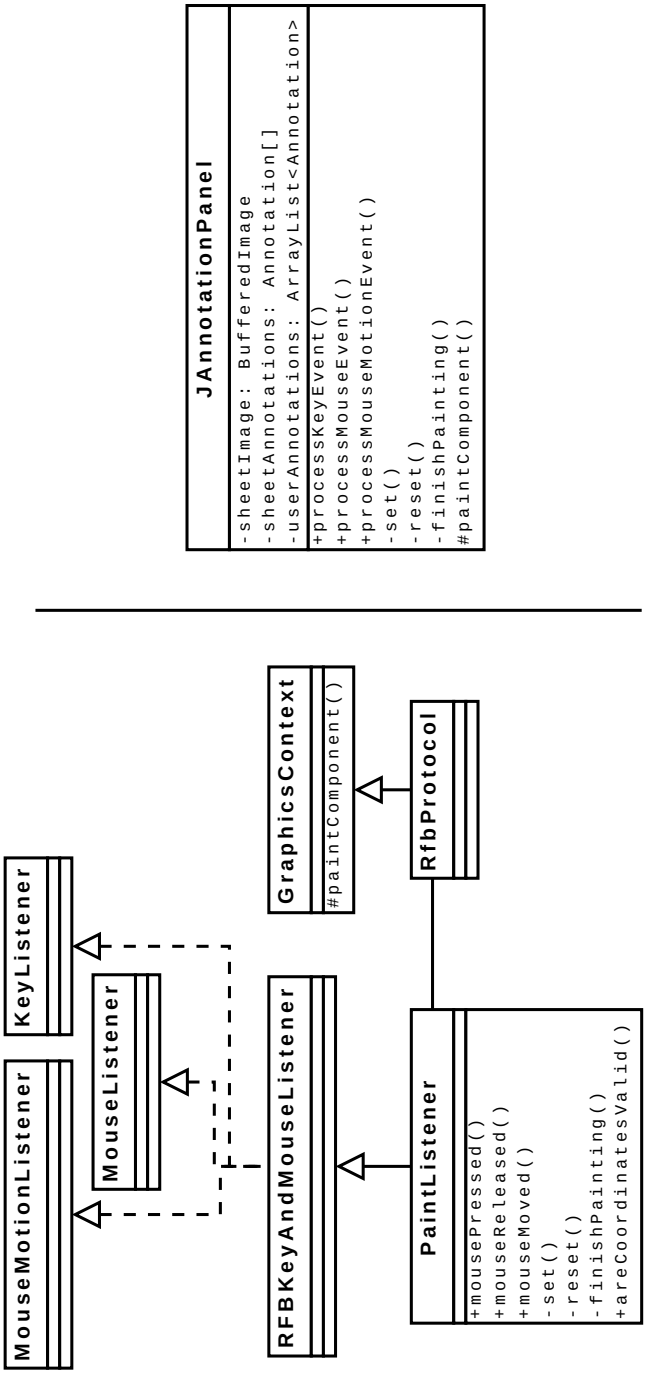


Abbildung 4.9: beteiligte Klassen zum Zeichnen von Annotation im TTT und im Nachrichtenclient

terklasse von `JComponent` ist und die `paintComponent()`-Methode überschreibt. Zunächst wird dort entweder die Folie gezeichnet (falls eine vorhanden ist) oder ein weißes Rechteck (Whiteboard). Die Bilddaten einer Folie werden zum einen in der binären Form, wie sie per Nachricht eingetroffen ist, als byte-Array `bImgData`, zum anderen in der Variable `sheetImage` als `BufferedImage` gespeichert. Danach wird die Methode `paintAnnotations()` aufgerufen, die die Annotationen über ihre `paint()`-Methode zeichnet, wobei es drei Variablen gibt, die Annotationen enthalten können:

- `sheetAnnotations` ist die Liste, die die Annotationen enthält, die zu einer vom Dozenten geholten Folie oder einem Whiteboard gehören
- `userAnnotations` ist die Liste der Annotationen, die der Hörer erstellt hat
- `annotation` enthält eine Annotation, wenn der Hörer gerade in Begriff ist, eine zu zeichnen

Nun fehlt noch der Teil, dass die Benutzer auch neue Annotationen erstellen können. Im TTT ist die Klasse `PaintListener`, der als Keyboard- und MouseListener bei `RfbProtocol` eingetragen wird, für die Behandlung der Keyboard- und Mouseeingaben zuständig. Bei allen Mouse-Events wird zunächst überprüft, ob die Koordinaten des Events außerhalb der Größe der Zeichenfläche liegen (`areCoordinatesValid()`). Das Erstellen einer Annotation läuft folgendermaßen ab: beim Drücken der linken Maustaste wird die Methode `set()` aufgerufen, die als Parameter die Koordinaten des Mouse-Events übergeben bekommt. Diese erstellt die vom Benutzer ausgewählte Annotation an dieser Stelle, setzt das `temporary`-Flag der Annotation und speichert diese in der Variable `annotation`. Wenn der Benutzer nun die Maus bewegt, erhält der `PaintListener` „MouseDragged“-Events. Bei der Behandlung dieser Events wird die Methode `reset()` aufgerufen, die die Koordinate der temporären Annotation als Endpunkt übergibt oder im Falle einer FreehandAnnotation als neuen Wegpunkt des Pfades hinzufügt. Wenn der Benutzer die Maustaste losläßt wird die Methode `finishPainting()` aufgerufen. Diese setzt das `temporary`-Flag der Annotation zurück, fügt die Annotation in die Liste der Annotationen ein und setzt die `annotation Variable` wieder auf `null`.

Die Methoden `set()`, `reset()` und `finishPainting()` wurden fast 1:1 in den Nachrichten-Client übernommen. Unterschiede ergaben sich nur dadurch, dass die Hörer dort auch die neuen TextAnnotationen (siehe Abschnitt 4.3) erstellen können. Im Unterschied zur Implementierung im TTT werden die Maus- und Tastaturereignisse nicht über extra Listener-Objekte behandelt, sondern direkt durch das Überschreiben der Methoden `processMouseEvent()`, `processMouseMotionEvent()` und `processKeyKeyEvent()` in `JAnnotationPanel`. Damit diese Methoden aufgerufen werden, müssen sie erst einmalig

durch Aufruf von `enableEvents()` mit der passenden Maske als Parameter aktiviert werden.

Ein weiterer Unterschied ergibt sich beim Löschen von Annotationen. Im TTT werden bei Auswahl des Löschwerkzeuges in der `set()`- und `reset()`-Methode `DeleteAnnotation`s erstellt, und erst bei Abarbeitung dieser werden die betroffenen Annotationen gelöscht. Dies dient dazu, dass die Löschaktionen auch in die Aufnahme gelangen. Im Nachrichten-Client benötigt man dies nicht, so dass die Annotationen direkt über den Aufruf von `removeAnnotationsAt()` in `set()` und `reset()` gelöscht werden.

Damit ein Hörer auch unterschiedliche Annotationen erstellen kann, benötigt er eine Werkzeugleiste. Diese wurde auch aus dem TTT übernommen (dort `PaintControls`, im Client `JClientPaintControls`), jedoch wurde sie um weitere Knöpfe erweitert, u.a. für das Holen der Folien und Whiteboards und einem Knopf zum Löschen der Annotationen, die bei der Anfrage nach einer annotierten Folie enthalten waren. Da diese wie oben schon erwähnt in einer eigenen Liste gespeichert sind, können sie nicht über das Löschwerkzeug gelöscht werden.

4.6.2 Allgemeines

Im folgenden wird kurz der grobe Aufbau des Nachrichten-Clients beschrieben. Dieser ist im UML-Klassendiagramm in der Abbildung 4.10 ersichtlich.

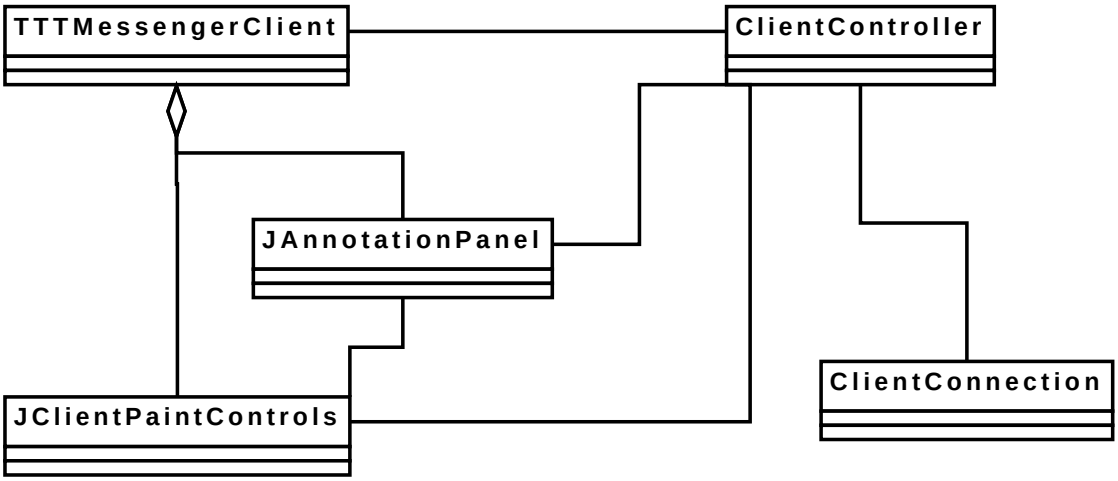


Abbildung 4.10: Die wichtigsten Klassen des Nachrichtenclients

Die Klasse `TTTMessengerClient` baut die allgemeine Oberfläche auf und verwendet dazu eine Instanz von `JClientPaintControls` für die Werkzeugleiste und ein `JAnnotationPanel` als Zeichenfläche. Alle drei Instanzen besitzen jeweils eine Referenz auf den `ClientController`, da dieser für die Ausführung der Aktionen, die verschiedenen Knöpfe der Oberfläche auslösen, zuständig ist. Sofern der Benutzer eine Nachricht senden will oder Daten angefordert hat, verwendet der `ClientController` die `ClientConnection`. Diese ist für die Kommunikation mit dem Server zuständig und erstellt die Nachrichten in ihrer XML-Repräsentation. `JClientControls` hat auch eine Referenz auf `JAnnotationPanel`, damit letzteres direkt erfährt, welches Zeichenwerkzeug und welche Farbe der Benutzer ausgewählt hat.

4.6.3 VoteDialog

Die Teilnahme an Abstimmungen wird über ein eigenes Dialogfenster realisiert, dass nach Drücken auf den entsprechenden Knopf in der Werkzeugleiste erscheint. Genaugenommen erscheint es erst, wenn die Antwort mit den aktuellen Abstimmungen eingetroffen ist.

Die Klasse `JOptionPane` bietet eine elegante Möglichkeit, Dialoge mit eigenem Inhalt anzuzeigen. Der Parameter *message* der verschiedenen Methoden dieser Klasse zum Zeigen von Dialogen ist vom Typ Object und darf eben auch eine `JComponent` sein, die dann im Dialog dargestellt wird. Man spart sich damit das Schreiben eines vollständigen eigenen Dialogs. [sun-api-dlg]

Die Darstellung übernimmt damit eine Instanz der Klasse `VotePanelDialog`, Unterklasse von `JPanel`, wobei im Konstruktor eine Liste der empfangenen Abstimmungen übergeben wird. Pro Abstimmung wird ein `JPanel` mit Rand erstellt, und diese mit Hilfe eines `GridLayout`-Managers angeordnet. Ein `JPanel` enthält dabei ein `JLabel` für die Frage, einen `JRadioButton` für die Möglichkeit, sich seiner Stimme für diese Umfrage zu enthalten, und für jede Antwort ein weiterer `JRadioButton`. Die `JRadioButton`s sind über eine `ButtonGroup` miteinander verbunden. Da QuickPolls keinen Text für die Fragen und Antworten enthalten, wird bei diesen statt der Frage ein Hinweis angezeigt, dass die Frage und die Antworten auf der Vortragsprojektion zu finden sind, und bei den Antworten steht einfach nur „ - answer x -“. In Abbildung 4.11 ist der Dialog zu sehen.

Außerdem erhält das JPanel bei einem QuickPoll die Farbe der Abstimmung zugewiesen, wobei aber nicht die jeweilige Annotationsfarbe direkt verwendet werden kann, da diese

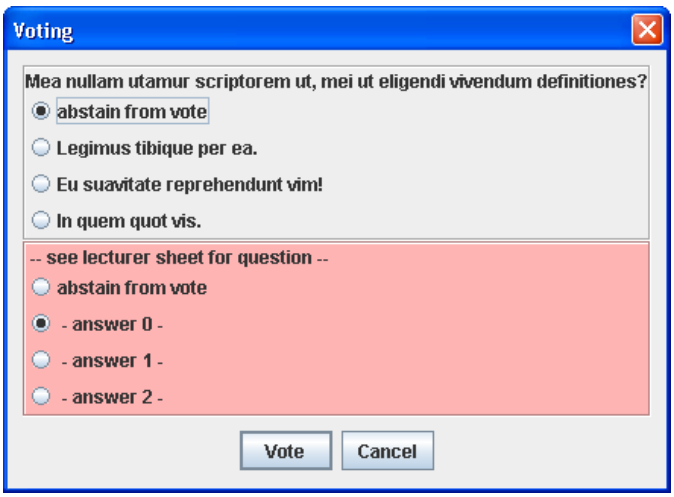


Abbildung 4.11: Darstellung der Abstimmungen im Nachrichten-Client

als Hintergrund zu knallig wären. Leider kann man aber auch nicht die in der Klasse `Annotation` zu den Farben korrespondierenden, halbtransparenten Varianten nehmen, da es bei Verwendung von Farben mit einem Alpha-Kanal als Hintegrundfarbe zu einem Darstellungsfehler mit den `JRadioButtons` kommt. Somit mussten eigene Farben definiert werden.

Nachdem der Hörer seine Stimmen abgegeben hat und den Dialog über „Vote“ geschlossen hat, müssen die Stimmen ermittelt werden, damit sie an den Server geschickt werden können. Dazu stellt `VotePanelDialog` die Methode `getVotes()` zur Verfügung. Diese benutzt ein zwei-dimensionales Array, in welchem Referenzen auf alle JRadioButtons abgelegt sind. Sie geht dann dieses Array durch, überprüft welche `JRadioButtons` selektiert wurden und erstellt dann pro abgegebener Stimme ein `Vote`-Objekt, welches die ID der Abstimmung und die Nummer der Antwort enthält. Die in einer Liste gesammelten Stimmen werden dann von `getVotes` zurückgegeben.

4.7 Sonstiges

Im Anwendungsfall „Senden einer Textnachricht“ in Kapitel 2.2.1 wurden verschiedene Möglichkeiten vorgestellt, wie der Client den Nachrichten-Server finden kann. Ich habe mich für die Variante entschieden, bei der die Hörer die IP oder den Namen des Rechners

eingeben müssen. Falls der Präsentationsrechner keinen festen Namen hat, müssen die Hörer die IP erfahren. Dazu wird diese nachdem Start des Nachrichten-Servers in die Videoprojektion eingeblendet, so dass diese während des gesamten Vortrags zur Verfügung steht. Da der Nachrichten-Server aber bei mehreren aktiven Netzwerkverbindungen nicht weiß, über welche die Clients sich verbinden können, muss der Dozent diese Entscheidung treffen.

Dazu wird beim Start des Nachrichten-Servers ein Dialog-Fenster angezeigt, in dem alle aktiven Netzwerkschnittstellen des Präsentationsrechners mit ihrer jeweiligen IP und ihren Namen aufgeführt sind. Implementiert wurde dies wieder über eine eigene Klasse, die `JComponent` erweitert und die dem Aufruf des Dialogs über `JOptionPane` mitgegeben wird. Diese Klasse ist `NetworkInterfaceDialog` und sie holt sich über die statische Methode `getNetworkInterfaces()` der Klasse `NetworkInterface` eine Aufzählung der Schnittstellen. Die Darstellung erfolgt dann über `JRadioButton`s.

Eine Anforderung, deren Umsetzung schon in den vorherigen Abschnitten verwendet wurde, ist die Identfizierung von Benutzern. Diese erfolgt über die IP, die das Gerät des Benutzers hat. Wenn der Dozent einen User ignorieren möchte, wird dessen IP gespeichert und weitere Nachrichten von dieser IP werden nicht dargestellt. Bei den Abstimmungen enthält jede eine Liste mit den IPs, von denen bereits eine Stimme kam.

5 Evaluation

Nach der Implementation stellt sich die Frage, ob die Ziele der bisherigen Bemühungen auch erreicht wurden. Im Titel dieser Arbeit sind die Schlagworte „Instant Messaging“ und „kooperative Whiteboardbenutzung“ enthalten. Ersteres ist mit dem implementierten System möglich, da die Hörer Nachrichten in Form von Text oder annotierten Folien oder Whiteboards an den Dozenten senden können und dieser die Nachrichten sofort angezeigt bekommt. Gegenüber einem „normalen“ Instant Messaging wie ICQ, Jabber u.a., bei dem sich zwei oder mehr Personen unterhalten, ist dies hier nicht möglich, da der Dozent keine Nachrichten an einzelne oder alle Hörer senden kann. Es wurde absichtlich darauf verzichtet (siehe Anwendungsfall dazu, Abschnitt 2.2.3 auf Seite 17), ließe sich aber mit dem bestehenden Kommunikationsprotokoll implementieren.

Auch die kooperative Whiteboard Benutzung ist vorhanden. Ein Hörer kann Whiteboards von Dozenten holen, diese erweitern und wieder zurückschicken. Der Dozent kann die Nachricht mit dem Whiteboard öffnen und auch wieder bearbeiten. Daraufhin kann sich wiederum der Hörer das Whiteboard holen mit allen Annotationen holen. Kurz gesagt, der Dozent und die Hörer können nacheinander gemeinsam an einem Whiteboard arbeiten.

Inwiefern läßt sich nun das Nachrichtensystem tatsächlich in Vorträgen einsetzen? Geplante Einsatzgebiete sind vorrangig Vorträge und Vorlesungen, bei denen die mündliche Kommunikation mit dem Dozenten schwierig, umständlich oder nicht möglich ist. Dazu zählen Vorlesungen, die live in andere Räume, Gebäude oder Universitäten übertragen werden. Hier ist der Einsatz möglich, sofern die entfernten Hörer, die natürlich geeignete Geräte dabei haben müssen, Zugang zum Netzwerk haben und darüber eine TCP-Verbindung zum Präsentationsrechner des Dozenten aufbauen können und dürfen. Dies ist die einzige technische Voraussetzung für den Einsatz des Nachrichtensystems.

Auch für große Vorlesungen mit einem hohen Geräuschpegel ist das Nachrichtensystem gedacht. Inwieweit eine große Anzahl an Hörern, die am Nachrichtensystem teilnehmen, Einfluß auf die Einsetzbarkeit haben, muss eine entsprechende Evaluation zeigen (siehe

folgenden Abschnitt 5.1). Per se gibt es keine Beschränkung der Anzahl der Clients, Einschränkungen können sich aber durch die benötigten Resourcen pro Client ergeben. Hier außerdem zu beachten ist, ob im Hörsaal ausreichend viele Netzzugänge verfügbar sind.

Bei kleineren, nicht übertragenen Vorlesungen stellt sich weniger die Frage der Einsetzbarkeit des Nachrichtensystems. Diese ist gegeben, sofern die Hörer sich mit dem Server verbinden können. Sondern es ist fraglich, ob es sinnvoll ist, es einzusetzen, da in einem solchen Rahmen Fragen auch mündlich gestellt werden können. Dafür sprechen könnten aber die Möglichkeiten der gemeinsamen Whiteboard Benutzung und der Durchführung von Abstimmungen.

Ob das Nachrichtensystem regelmäßig eingesetzt wird, hängt auch zu einem guten Teil von der Bedienbarkeit sowohl des Clients auf der Hörerseite als auch des Servers auf Seite des Dozenten. Die Oberfläche sollte intuitiv erfassbar sein und die Funktion von Eingabeknöpfen sollten sich aus dem präsentierten Icon oder spätestens aus dem zugehörigen Tooltip ergeben. Ob dies erreicht wurde, läßt sich nur durch eine entsprechende Evaluation ermitteln. In Kapitel 5.2 wird vorgestellt, wie diese sich durchführen ließe.

Leider konnten aus zeitlichen Gründen die Evaluationen nicht durchgeführt werden. Deshalb wird in den folgenden Passagen nur erklärt, wie diese durchgeführt werden können, und im Falle der Performance-Evaluation werden mögliche Ergebnisse und deren Konsequenzen geschildert.

5.1 Performance-Evaluation

Hierbei soll herausgefunden werden, ob es bei einer größeren oder sogar schon kleineren Anzahl an Hörern, die sich zum Server verbinden zu Problemen hinsichtlich der Auslastung des Speichers oder des Prozessors auf dem Präsentationsrechners kommt. Auch das Netzwerk, über das die Clients mit dem Server verbunden sind, könnte eine hohe Belastung erfahren, wenn viele Hörer gleichzeitig z.B. eine Folie anfordern.

Zunächst muss untersucht werden, ob schon eine große Anzahl an aufgebauten aber inaktiven Verbindungen zu Problemen führt, da die Clients die ganze Zeit mit dem Server verbunden sind, auch wenn nichts gesendet oder angefordert wird. Es werden pro Verbindung doch einige Objekte angelegt: eine `TTTMessengerConnection` mit einem eigenen Thread und dazu ein `ServerMessageParser` mit einem XML-`DocumentBuilder`.

Falls dem so ist, gibt es zwei Maßnahmen zur Besserung. Zum einen ändert man die Clients so, dass sie sich nur verbinden, wenn sie eine Nachricht senden oder eine Anfrage stellen. Die Verbindung wird nach Abschluß des Sendes oder Erhalt der Antwort auf die Anfrage wieder abgebaut. Damit würde, sofern sich die Anfragen und Nachrichten der Hörer zeitlich halbwegs verteilen, zu einer deutlichen Abnahme der gleichzeitig offenen Verbindungen führen. Allerdings würden ständig bei Aufbau einer Verbindung neue Objekte erstellt und bei Abbau wieder gelöscht.

Dies kann durch die zweite Maßnahme verhindert werden. Man legt bei Start des Servers einen Pool mit Threads an, die die Anfragen und Nachrichten der Clients abarbeiten. Jeder Thread besitzt einmal die nötigen Objekte zum Verarbeiten der Nachrichten. Falls alle vorhandenen Threads belegt sind, muss ein Verbindungsversuch eines Clients warten, bis einer frei ist, der sich darum kümmern kann. Der Vorteil ist, dass man damit eine feste Anzahl an Objekten hat und man die Speicher- und Performanceauslastung über die Anzahl der Threads im Pool steuern kann. Aber es können Wartezeiten für Clients entstehen, wenn diese sich verbinden möchten, aber kein Thread zur Verfügung steht. Diese Wartezeiten sollten aber recht klein sein, da die Zeit zum Abarbeiten einer Nachricht eines Clients recht kurz ist. Außerdem wären Wartezeiten auch vorhanden oder der Client könnte sich gar nicht verbinden, falls der Rechner mit zu vielen Verbindungen ausgelastet wäre. Ein weiterer Nachteil ist, dass die Implementierung aufwendiger wäre, als die jetzige.

Ein weiteres Performanceproblem könnte entstehen, wenn das Verarbeiten der Nachrichten und der Anfragen der Clients zu aufwendig oder zu ineffizient ist. Ob dies so ist, müsste ein Test zeigen, bei dem eine größere Zahl an Clients gleichzeitig Nachrichten an den Server schicken. Eine mögliche Ursache kann die Verwendung eines DOM-basierten XML-Parsers bei der Verarbeitung der eingehenden Nachrichten sein. Dieser benötigt mehr Resourcen als ein Push- oder Pull-basierter XML-Parser. Als Abilfe könnte man die Verarbeitung mit Hilfe des push-basierten SAX-Parsers umschreiben. Ein weiterer Punkt zur Steigerung der Abarbeitungszeit wäre das Einführen eines Caches für häufig benötigte Antworten auf Anfragen. Anbieten würde sich in erster Linie das Zwischenspeichern der aktuellen Folie oder Whiteboards mit den Annotationen des Dozenten. Damit müsste nicht für jede Anfrage erneut die aktuelle Folie vom TTT geholt werden. Der Cache muß natürlich aktualisiert werden, damit die Hörer nicht veraltete Daten zu gesicht bekommen. Die Aktualisierung der Foliendaten könnte man an die Folienerkennung des TTT koppeln. Die Annotationen, die sich häufiger ändern, als die Foliendaten, könnte man in festen Zeitabständen, beispielsweise alle 10 Sekunden, aktualisieren.

5.2 Benutzer-Evaluation

Die Evaluation soll zeigen, wie Benutzer mit dem Nachrichtensystem zurecht kommen, ob ihnen die Bedienung zusagt und ob sie sich vorstellen können, es regelmäßig einzusetzen. Manche haben vielleicht auch Ideen für weitere Einsatzmöglichkeiten oder Verbesserungen.

Die Evaluation sollte zunächst erstmal während einer kleineren und danach erst in einer größeren Vorlesung durchgeführt werden, die sowieso schon mit dem TTT gehalten werden, damit sich die Hörer bei ihrer Bewertung nur auf das neue Nachrichtensystem beschränken. Da es schwierig ist, für alle Hörer jeweil ein Gerät zum Ausführen des Clients zur Verfügung zu stellen, sollte eine oder zwei Vorlesungen zuvor eine Vorankündigung gemacht werden, damit die Hörer ihre Laptops, falls vorhanden, mitbringen können. Außerdem sollte man ihnen schon eine mitteilen, wo sie das TTT mit dem integrierten Client beziehen können, damit sie diesen schon installieren können. Zu Beginn der Vorlesung gibt man eine kurze Erklärung über das Nachrichtensystem und dessen Möglichkeiten.

Während der Vorlesung beobachtet man die Hörer, wie sie mit dem Client umgehen und inwieweit sie die Funktionen des Nachrichtensystems nutzen. Am Ende der Vorlesung bittet man die Hörer Fragen zu beanworten. Dies kann über einen papiernen Evaluationsbogen geschehen, den man austeilt, oder über eine Webseite. Mögliche Fragen an die Hörer können sein:

- Hat sich dich Kommunikation mit dem Dozenten verbessert?
- Ist die Bedienung des Clients einfach?
- Könnten Sie sich vorstellen, dass Sie mit dem Nachrichten-Client öfters Fragen stellen?
- Würden Sie extra einen Laptop mit zur Vorlesung nehmen, um dem Client einsetzen zu können?
- Finden Sie, dass durch das Nachrichtensystem die Hemmschwelle Fragen zu stellen gesenkt wird?
- Konnten Sie mit Hilfe des Nachrichtensystems Ihre Frage besser formulieren (z.B. durch Markierungen und Annotationen auf einer Folie des Dozenten)?
- Konnte mit dem Dozenten besser über seine Folien diskutiert werden (z.B. weil man die Möglichkeit hatte, seine Gedanken aufzuzeichnen o.ä.)?

Die Antworten werden, wie bei Umfragen üblich, als Kreuz in einer fünfstufigen Rangfolge von positiv über neutral zu negativ gegeben. Außerdem sollte man den Hörern ein freies Feld zum Niederschreiben von Bemerkungen und Vorschlägen geben.

Auch der Dozent wird während der Vorlesung beobachtet, wie er zurecht kommt. Am Ende der Vorlesung bekommt auch er einen Fragebogen oder wird direkt befragt:

- Sind sie mit der Bedienung des Nachrichten-Servers zurecht gekommen?
- Hat sie die Verwaltung (Löschen, Zurückstellen) der eingegangen Nachrichten zu sehr abgelenkt?
- Finden Sie, dass sich die Kommunikation mit den Studenten durch das Nachrichtensystem verbessert?
- Konnten Sie Fragen von Hörern besser verstehen (*nicht* im Sinne von akustisch verstehen), wenn diese zu ihren Fragen die zugehörigen Folien mitgeschickt haben?
- Wie finden Sie die Möglichkeit, Abstimmungen durchführen zu können?
- Können Sie sich vorstellen, das Nachrichtensystem regelmäßig in Ihrer Vorlesung zu verwenden?
- Sehen Sie Nachteile beim Einsatz? Welche wären das?

Auch der Dozent sollte natürlich die Möglichkeit bekommen, nach Abschluß der Befragung seine allgemeine Meinung kund zu tun und ggf. Verbesserungsvorschläge zu nennen.

6 Zusammenfassung

6.1 aktueller Stand

Mit dem Nachrichtensystem gibt es nun einen elektronischen Rückkanal von den Hörern zum Dozenten. Wie bereits ausgeführt, kann dies in manchen Situationen der einzige Weg für die Hörer zur Kommunikation mit dem Dozenten sein. Die Implementation des Nachrichtensystems setzt dabei alle in Kapitel 2 erarbeiteten Anforderungen um.

Einschränkungen für den Einsatz können sich zunächst daraus ergeben, wieviel Resourcen die derzeitige Implementierung benötigt. Es könnte somit sein, das derzeit nur eine begrenze Anzahl an Hörern sich verbinden können. Ob dem so ist, würde die Perfomance-Evaluation (siehe Kapitel 5.1) zeigen und ggf. müsste man die dort aufgezeigten Wege zur Besserung umsetzen.

Ansonsten gibt es natürlich immer Raum für kleine Verbesserungen. Bei der Erstellung der XML-Strings für die Nachrichten könnte man doch auf eine XML-API setzen. Dies hätte den Vorteil, dass man sich nicht selbst um die Behandlung von in XML ausgezeichneten Zeichen in Eingaben der Benutzer kümmern muss. Man sollte sich aber eine Klasse schreiben, mit der man direkt `<tttmessage>`-Nachrichten erstellen kann und in der man die ganzen API-Klassen, die man zur Erstellung des XMLs benötigt, versteckt. Möglichkeiten zur Erstellung von XML-Strings mit API-Klassen führt [www-xmlstring] auf.

Bei den TextAnnotations könnte man die Unterscheidung zwischen den beiden Modi aufheben. Stattdessen wird immer am Ende der Zeichenfläche umbrochen und es sind aber gleichzeitig auch manuelle Umbrüche erlaubt. Auch der Flash-Export von Text könnte verbessert werden. Zur Zeit wird zu langer Text nicht umbrochen, so dass Textteile nicht dargestellt werden, wenn der Text zu lang ist. Vielleicht gibt es doch eine Möglichkeit, den Umbruch in Flash umzusetzen oder man muss doch auf die Java-Berechnung der Umbrüche zurückgreifen.

6.2 Ausblick

Zum Schluß dieser Arbeit werden noch einige Ideen zur Erweiterung des Nachrichtensystems vorgestellt.

6.2.1 andere Geräte als Clients

Eine naheliegender Punkt ist die Entwicklung von Clients für weitere Geräte. Der in Rahmen dieser Arbeit geschriebene Client beschränkt sich auf Geräte, auf denen Java 2 SE 1.5 JVMs verfügbar sind. Aufgrund der Tatsache, dass man das Gerät mit zum Vortrag nimmt, wird dies meist ein Laptop ein.

Interessante Geräte für die Erweiterung wären SmartPhones und PDAs mit integriertem WLan, da diese weniger sperrig wie Laptops sind und manche Hörer diese sowieso schon als Handy mit sich tragen. Übliche Geräte besitzen als Betriebssystem Windows Mobile, Palm OS, Symbian OS oder (beim Apple iPhone) Mac OS X. Auf den meisten Geräten ist Java verfügbar, üblicherweise in Form einer Java ME JVM. Je nach unterstütztem Profil sind diese gegenüber Java SE deutlich eingeschränkt in den vorhanden Klassen. Für das Apple iPhone ist dagegen keine Java-Umgebung verfügbar. Einen Client müsste man demnach mit dem von Apple zur Verfügung gestellten SDK erstellen.

Durch die Verwendung von XML zur Übertragung der Nachrichten sollte sich die Kommunikation jedoch auf den meisten Plattformen umsetzten lassen. Worüber man sich Gedanken machen muss, ob man für diese Geräte vollwertige Clients schreibt, d.h. ob man auch Folien oder Whiteboards annotieren kann, da die Bildschirmgröße doch recht eingeschränkt ist. Sie liegt meist im Rahmen von 320x240 (niedrige Windows Mobile Auflösung) über 480x320 (Apple iPhone) bis 640x480 (höhere Auflösung bei Windows Mobile). Da eine Folie üblicherweise eine Auflösung von 1024x768 hat, müsste man sich überlegen, wie man die Folien darstellt, ob man in die Folie zoomen kann und dann erst Annotationen erstellen kann oder ob man auch in der verkleinerten Ansicht Annotationen erstellen kann.

Zumindest für die Versendung von Textnachrichten und die Teilnahme an Umfragen würde eine recht einfache Oberfläche ausreichen, so dass man zunächst nur Clients mit diesen beiden Funktionaliäten anbieten könnte, um zumindest somit eine breite Auswahl an Geräten zu unterstützen.

6.2.2 Webserver Messaging

Einen etwas anderen Weg der Kommunikation würde die folgende Idee bieten: im Nachrichten-Server läuft ein Webserver, der Webseiten ausliefert, auf denen die Nachrichten erstellt und abgeschickt werden können. Der Vorteil wäre, dass die Hörer kein extra Programm (das TTT mit integriertem Client) installieren müssen, sondern über ihren normalen Webbrowser Nachrichten verschicken können.

Das Versenden von Textnachrichten und die Teilnahme an Umfragen ließen sich recht einfach umsetzen, da dafür Standard-HTML-Formulare ausreichen. Der Webserver würde dann die per HTTP Post oder Get Anfrage übermittelten Informationen verarbeiten.

Anspruchsvoller wäre die Umsetzung der Annotation von Folien oder Whiteboards. Dass sich dies aber über JavaScript und SVG lösen läßt, zeigen verschiedene Online-Zeichenprogramme, wie die recht einfache SVG Demo Application von Amaltas [www-draw1] oder die beiden schon recht komplexen Web-Applikationen *ajaxsketch* von ZoooS [www-draw2] und *Projekt Draw* von Autodesk [www-draw3]. Man müsste dann einen Konverter schreiben, der die Annotation von und in SVG-Elemente umwandelt.

6.2.3 digitale Vorlesungsnotizen

Da durch das Nachrichtensystem die Möglichkeit besteht, Folien des Dozenten an die Hörer zu übertragen und diese dann die Folien um eigene Annotationen erweitern können, wäre die Idee, das die Hörer ihre Vorlesungsnotizen gleich direkt am Laptop machen. Damit würde das Ausdrucken der Folien des Dozenten entfallen und Papier gespart werden. Außerdem könnte der Dozent noch bis Beginn der Vorlesung bearbeiten und müsste sie nicht im Vorfeld der Vorlesung schon zum Drucken zur Verfügung stellen. Allerdings sollten die Hörer dann möglichst auch Rechner mit GraphicTablet-Funktion verwenden, damit das Schreiben der Notizen mit der Freihandwerkzeug gut von der Hand geht.

Verschiedene Punkte, zu denen man sich unter anderem Gedanken machen müsste, sind:

- Wie speichert man die so annotierten Folien lokal auf dem Rechner des Hörers (das gleiche XML-Format wie bei der Übertragung oder doch platzsparender in einem Binärformat)?

- Wie kann man die gespeicherten Folien wieder aufrufen, darin navigieren und ggf. ausdrucken?
- Müssen die Studenten die Folien selbst holen oder werden diese automatisch bei Folienwechsel gesendet?
- Falls automatische Übertragung, was passiert, wenn eine neue Folie beim Benutzer ankommt, dieser aber noch bei der vorherigen Folie Notizen schreibt? Puffern der neuen Folien im Client?

Literaturverzeichnis

[www-draw1] : *amaltas - SVG Demo Application.* – URL `http://www.amaltas.org/svgapp/`

[apch-cdc] : *Apache Commons Codec.* – URL `http://commons.apache.org/codec/`

[www-draw3] : *Autodesk - Project Draw.* – URL `http://labs.autodesk.com/technologies/draw/`

[sun-base64] : *Bug ID: 4235519 Make sun.misc.BASE64{De,En}coder classes public.* – URL `http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4235519`

[www-eclipse] : *Eclipse.org*

[sun-rmi-tut] : *Getting Started Using Java RMI.* – URL `http://java.sun.com/j2se/1.5.0/docs/guide/rmi/hello/hello-world.html`

[www-ziewer] : *Homepage - Dr. Peter Ziewer.* – URL `http://www2.in.tum.de/~ziewer`

[www-lecturnity] : *imc - Lecturnity.* – URL `http://www.lecturnity.de/de/produkte/lecturnity/`

[sun-rmiop] : *J2ME RMI Optional Package.* – URL `http://java.sun.com/products/rmiop/`

[www-jabber] : *Jabber, open instant messaging and presence.* – URL `http://www.jabber.org`

[sun-api-dlg] : *Java 2 Platform SE 5.0 API - JOptionPane.* – URL `http://java.sun.com/j2se/1.5.0/docs/api/`

[sun-javaapi] : *Java 2 Platform Standard Edition 5.0 API Specification.* – URL `http://java.sun.com/j2se/1.5.0/docs/api/`

[sun-tut-line] : *The Java Tutorials - Drawing Multiple Lines of Text.* – URL `http://java.sun.com/docs/books/tutorial/2d/text/drawmulstring.html`

[sun-tut-sckt] : *The Java Tutorials - Lesson: All about Sockets.* – URL `http://java.sun.com/docs/books/tutorial/networking/sockets/index.html`

[sun-tut-font] : *The Java Tutorials - Measuring Text.* – URL `http://java.sun.com/docs/books/tutorial/2d/text/measuringtext.html`

[www-javaswf2] : *JavaSWF - How to create text using the object model.* – URL `http://www.anotherbigidea.com/javaswf/javaswfdoc.html`

[www-javaswf1] : *JavaSWF - How to load a JPEG using the object model.* – URL `http://www.anotherbigidea.com/javaswf/samples/ImportJPEG.java`

[www-xml] : *JavaZoom Tutorial: XML generation.* – URL `http://javazoom.net/services/newsletter/xmlgeneration.html`

[sun-rmi] : *Remote Method Invocation Home.* – URL `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp`

[w3c-soap] : *SOAP Version 1.2 Part1: Messaging Framework (Second Edition).* – URL `http://www.w3.org/TR/soap12-part1/`

[www-camtasia] : *TechSmith - Camtasia.* – URL `http://www.techsmith.de/camtasia.asp`

[ttt-tum] : *TeleTeaching Homepage an der TU München.* – URL `http://ttt.in.tum.de`

[wiki-soap] : *Wikipedia - SOAP.* – URL `http://de.wikipedia.org/wiki/SOAP`

[wiki-convertible] : *Wikipedia - Tablet PC - Convertible.* – URL `http://de.wikipedia.org/wiki/Tablet_PC#Convertible`

[wiki-uuencode] : *Wikipedia - UUencode.* – URL `http://de.wikipedia.org/wiki/UUencode`

[www-xmpp] : *XMPP Standards Foundation.* – URL `http://www.xmpp.org`

[www-draw2] : *Zooos - ajaxsketch.* – URL `http://www.ajaxsketch.com`

[Campbell 2007] Campbell, Chris: The Perils of Image.getScaledInstance(). In: *java.net* (2007). – URL `http://today.java.net/pub/a/today/2007/04/03/perils-of-image-getscaledinstance.html`

[Erbsland und Nitsch] Erbsland, Tobias ; Nitsch, Andreas: *Diplomarbeit mit LaTeX.* – URL `http://drzoom.ch/project/dml/`

[Goossens u. a. 2003] Goossens ; Mittelbach ; Samarin: *The LaTeX Companion*. Addison-Wesley, September 2003. – ISBN 0-201-54199-8

[James] James, Mark: *Silk Icons*. – URL `http://www.famfamfam.com/lab/icons/silk/`

[Josefsson] Josefsson, S.: *RFC 4648 - The Base16, Base32, and Base64 Data Encodings*

[Knudsen] Knudsen, Jonathan: *Parsing XML in J2ME*. – URL `http://developers.sun.com/mobility/midp/articles/parsingxml/`

[Lämmle] Lämmle, Stefanie: *Übersicht Autorentools*. – URL `http://portal.mytum.de/iuk/electum/lernplattform/uebersicht_autorentools/#Aufzeichnung`

[Menneisyys] Menneisyys: *Comparison table of different mobile JVMs*. – URL `http://www.winmobiletech.com/092007MidletBible/CompatibilityAndMain.html`. – // published in „The (Java) MIDlet Bible“, // `http://forum.xda-developers.com/showthread.php?t=339577`

[Muller und Walrath 2000] Muller, Hans ; Walrath, Kathy: *Threads and Swing*. 2000. – URL `http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html`