COMO

Ein Applet-basierter Prototyp für computerunterstützte Moderation

Universität Hamburg - Fachbereich Informatik

Arbeitsbereich Softwaretechnik

Studienarbeit

Betreuer: Julian Mack

Juli 1999

Henrik Ortlepp
Matr.Nr.: 4823035

Wolfgang Riese
Matr.Nr.:4627477

Inhalt:

1. Einleitung

(von Henrik Ortlepp)

Mit der Verbreitung des Internets haben sich für die Arbeit in überregionalen Gruppen ganz neue Möglichkeiten ergeben. Es ist nun beispielsweise möglich, moderierte Sitzungen nicht mehr traditionell so durchzuführen, daß alle Teilnehmer in einem Raum zusammenkommen, sondern es gibt die Möglichkeit in virtuellen Gruppen – also in Gruppen, in denen die Teilnehmer physisch voneinander getrennt und eben nur durch das Internet miteinander verbunden sind – zu diskutieren.

Es liegt nahe, neben den durch die Vernetzung entstandenen Methoden wie Chat, Newsgroups und Foren, auch traditionelle Moderationsmethoden zu benutzen, die auch in realen Gruppen eingesetzt werden. Unser Ziel ist es nun, eine Plattform zur Verfügung zu stellen, die sowohl die Vorteile des Internets nutzt als auch die konventionellen Moderationsmethoden unterstützt. Auf dieser Plattform soll es möglich sein, an einer Moderation einer virtuellen Gruppen teilzunehmen und zwar ohne den Einsatz zusätzlicher kommerzieller Software, sondern lediglich mit Hilfe eines Internet-Browsers. Unter Moderation verstehen wir hier eine Diskussion, die durch einen Moderator nach bewährten Konzepten (den Moderationsmethoden) geleitet wird (s.a. Kap. 2).

Die vorliegende Arbeit hat dabei die Entwicklung eines Applet-basierten Prototypen für diese Plattform – ***CoMo*** – zum Ziel. Dadurch, daß ***CoMo*** Applet-basiert ist, ist es möglich, das Programm (bzw. den Prototyp, der zur Aufgabe hat die generelle Funktionsweise darzustellen) mit einem Browser zu starten. Im Namen ***CoMo*** finden sich dann Hinweise auf die Bereiche, die es zu integrieren gilt: ***Co***mputerunterstützt und ***Mo***deration.

Im Laufe der Bearbeitung dieses Themas hat sich gezeigt, daß sich die verschiedenen Moderationsmethoden tatsächlich auf eine einheitliche Plattform abbilden lassen. Auch die Nutzung des Internets, um ***CoMo*** tatsächlich in virtuellen Gruppen nutzbar zu machen, ist erfolgreich verlaufen. Wir haben jedoch festgestellt, daß der Aufwand zur Erstellung eines funktionstüchtigen Programmes trotz Nutzung vorhandener Teillösungen beträchtlich ist.

Nach der Vorstellung der Moderationsmethoden und einem Marktüberblick über vorhandene Lösungen in Kapitel 2 beschäftigt sich der Rest der Arbeit mit dem Entwurf, der Konstruktion

und dem Ausblick auf spätere Versionen von ***CoMo***. Hierbei wird deutlich getrennt zwischen dem fachlichen Entwurf (Kapitel 3), der die vorhandene Situation bzw. die Vision der verschiedenen Arbeitsweisen des Prototypen aus Sicht des Anwenders beleuchtet, und dem technischen Entwurf (Kapitel 4), der das programmiertechnische Vorgehen und die „Innereien“ des Programms beschreibt.

In Kapitel 5 schauen wir einmal zurück auf die gewählte Arbeitsweise und was man daran eventuell verbessern könnte und in Kapitel 6 wird ein Ausblick auf spätere Versionen von ***CoMo*** gegeben, der beschreiben soll, was noch zu tun ist, um aus dem vorliegenden Prototypen ein funktionsfähiges Programm zu machen.

Das Literaturverzeichnis und Anhänge Installationshinweisen, Begriffserklärungen, Verzeichnis der Abbildungen und eine CD-ROM mit Quelltext und generierter Dokumentation runden die Arbeit ab.

2. Moderationsmethoden (von Henrik Ortlepp)

In diesem Kapitel sollen die in der Einleitung angesprochenen Methoden für Moderationen vorgestellt werden – zunächst die Metaplan-Methode Kapitel 2.1 und darauffolgend die Mind Maps in Kapitel 2.3. Da es bereits einige kommerzielle Produkte gibt, die sich mit der computerbasierten Erstellung von Mind Maps, teilweise sogar in virtuellen Gruppen, beschäftigen, enthält das Kapitel 2.4 auch eine kurze Übersicht über die auf dem Markt verfügbaren Programme und Kapitel 2.2 geht kurz auf die Situation im Bereich Metaplan ein.

Die Moderationsmethoden sind eine Sammlung von Methodenvorschlägen, um in verschiedenen Diskussions- und Vortragssituationen möglichst effizient und demokratisch zu Lösungen und Vorschlägen zu kommen. Deshalb hier noch eine kurze Übersicht darüber, welche Methoden es gibt und wofür sich welche am besten eignet. Dabei bedeuten die großen Kreise in der folgenden Tabelle (Abbildung 1), daß sich die entsprechende Methode besonders gut für das jeweilige Aufgabenziel eignet und kleinere, daß sich die Methode zwar auch für dieses Ziel eignet, jedoch nicht so optimal wie im Fall mit dem großen Kreis.

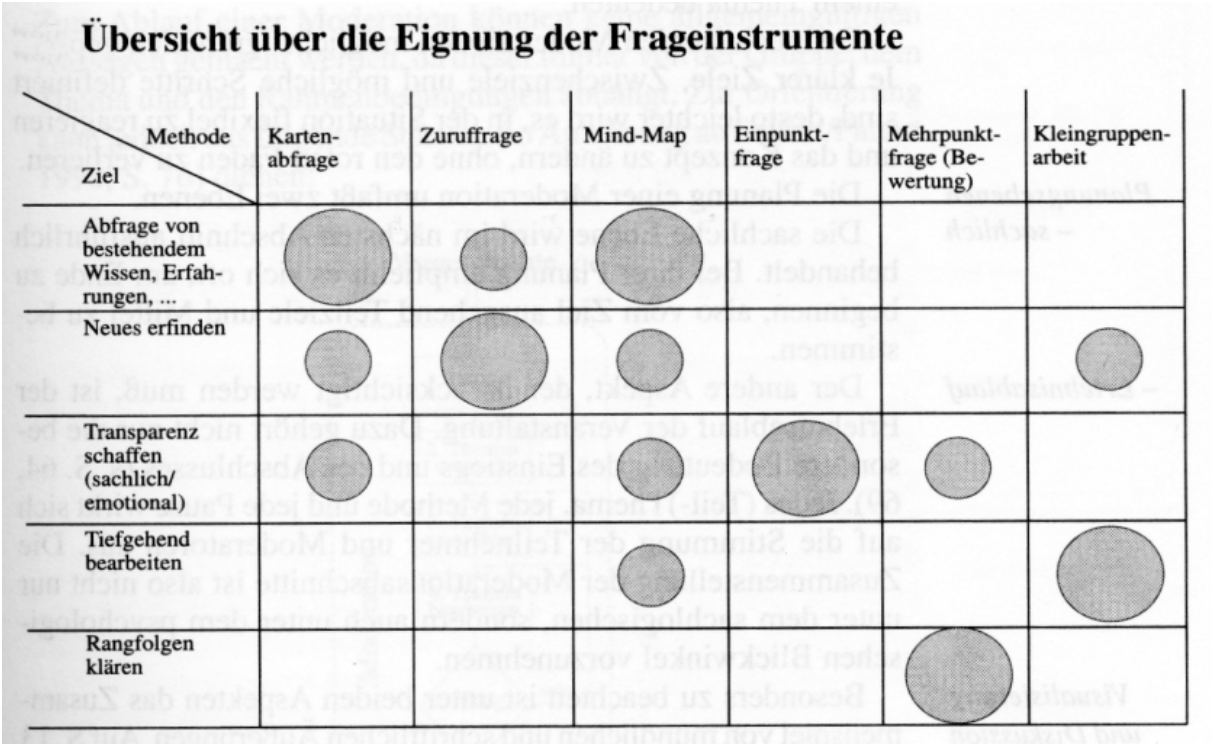


Abbildung 1 - über die Eignung der Frageinstrumente [Dauscher 1998] S.61

2.1. Die Metaplan-Methode (von Wolfgang Riese)

Hier nun ein kleiner geschichtlicher Überblick über die Entstehung der Metaplan-Methode, wie er im Buch Moderationsmethode und Zukunftswerkstatt [Dauscher 1998] beschrieben wird.

Die Metaplan-Methode entstand in den späten 60'er und frühen 70'er Jahren des 20Jh und entstand zum einen aus den Erfahrungen, die in den Versuchen der Reformierung des Hochschulwesens gewonnen wurden und zum anderen aus den Erfahrungen der in dieser Zeit entstehenden Unternehmensberatungen. In beiden Fällen hatten die Betroffenen erkennen müssen, daß wirklich geeignete Instrumente zur Koordinierung von Diskussionen und Ergebnissen entweder völlig fehlten, oder die vorhandenen nicht effizient genug waren.

So entwickelte Eberhard Schnelle, Mitglied einer Unternehmensberatung ("Quickborner Team") das "Entscheidertraining". In diesem wurde gelehrt, wie eine Planung und Entscheidungsfindung im Team unter Einbeziehung von Betroffenen aussehen könnte. Später kamen dann noch die damalige Dozentin für Sozialphilosophie und Sozialpsychologie an der Uni-Münster, Frau Dr. Karin Klebert und der Soziologe Dr. Einhard Schrader zum "Quickborner Team".
Anfang der 70'er Jahre entwickelten sie dann die Grundzüge der Moderationsmethode, die sie "Metaplan-Methode" nannten. Dabei wurde aus dem "Quickborner Team" die "Metaplan GmbH".
Seitdem wurden die Methoden beständig weiterentwickelt und verfeinert.

2.1.1. Was ist die Metaplan-Methode ? (von Wolfgang Riese)

Die Metaplan-Methode ist eine Sammlung von Vorschlägen und Werkzeugen, um moderierte und visuell unterstützte Vorträge und Diskusionsrunden abzuhalten.

Zu deren Unterstützung werden mit Hilfe von Pinwänden Kartenabfragen, Zurufabfragen, Einpunktfragen und Mehrpunktfragen durchgeführt, sowie Listen, Netze, Tabellen, Charts und Mindmaps erstellt. Eine genauere Erläuterung der entsprechenden Methoden erfolgt in den Kapiteln 2, 2.1, 2.3, 3.1 und 3.2.

In der Metaplan-Methode nimmt der Moderator, der diese Methoden koordiniert, eine zentrale Stellung ein. Er wählt die jeweils passende Methode aus.

Außerdem soll er dafür sorgen, daß alle Teilnehmenden im gleichen Rahmen zu Wort kommen und nicht durch Vielredner gehemmt werden. Des weiteren muß er dafür sorgen, daß keine persönlichen Fehden zwischen einzelnen ausgetragen werden. Zu diesem Zweck muß er unter anderem darauf achten, daß in den Aussagen der einzelnen nicht verallgemeinert und sich nicht hinter Phrasen versteckt wird.

Dabei ist es das Ziel, eine weitestgehend aufgelockerte Atmosphäre zu schaffen. In dieser soll es für alle Teilnehmer möglich sein, daß alle Themen mit der nötigen Offenheit und Spontaneität behandelt werden können. Dabei soll in vielen Situationen vor allem auch die innere Zensur des einzelnen über seine eigenen Ideen abgeschaltet werden.

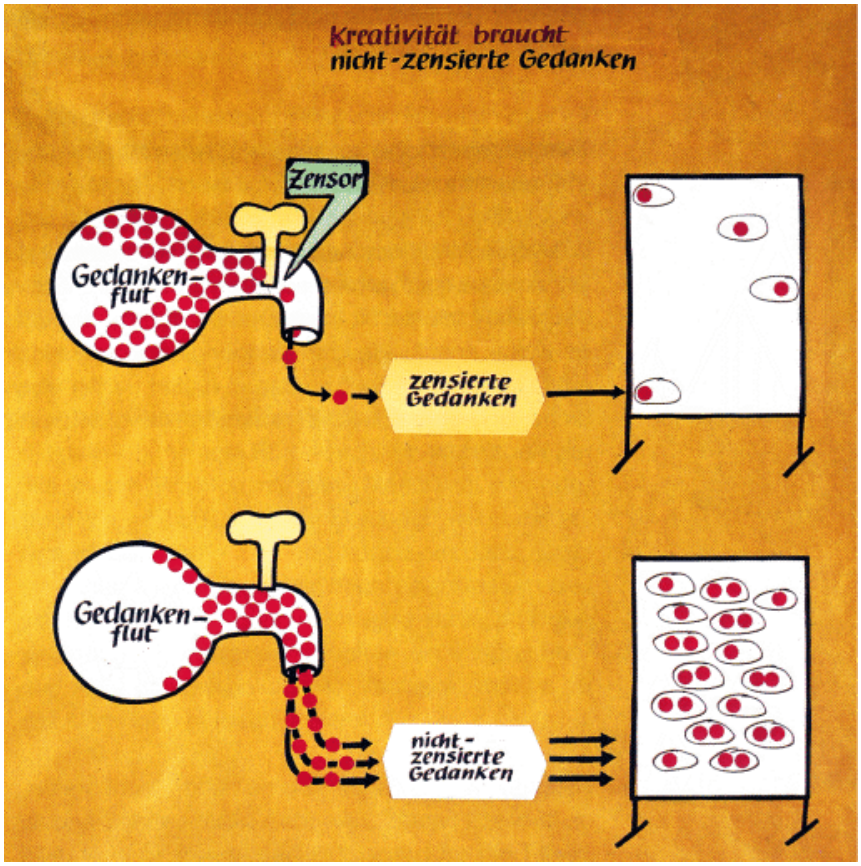


Abbildung 2 - Steigerung der Kreativität durch Abschalten des inneren Zensors [Schnelle-Cölln & Schnelle 1997] S.25

2.1.2. Elemente der Visualisierung in Metaplan (von Wolfgang Riese)

- Pinwände ca. 125 x 150 cm mit braunem Packpapier bespannt

- Filzstifte in verschiedenen Dicken und Farben (z.B. Edding 800 und Nr. 1)

- Karteikarten ca. 10 x 20 cm in weiß, grün, orange und gelb

- Ovale Karten ca. 11 x 19 cm in weiß, grün, orange und gelb

- Kreisrunde Karten mit ca. ø10 cm, ø15 cm und ø20 cm in weiß, grün, orange und gelb

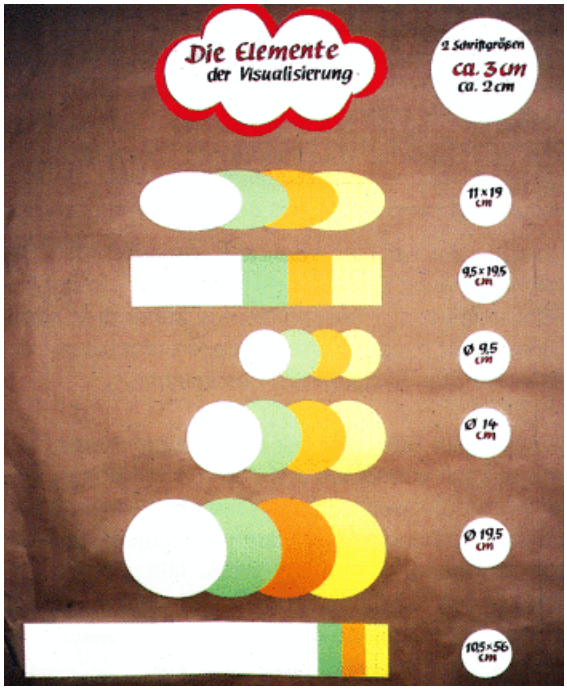


Abbildung 3 -Metaplankarten [Schnelle-Cölln & Schnelle 1997] S.58

- Papierstreifen ca. 10 x 56 cm in weiß, grün, orange und gelb

- Wolke - weißes Papier mit rotem Rand

- Beschriftungen in zwei Schriftgrößen

- Linien zum Verbinden

- Klebepunkte zum Bewerten

2.1.3. Vor- und Nachteile der computerunterstützten Metaplan-Moderation

Der Hauptvorteil der computerunterstützten Metaplan-Moderation ist, daß räumlich getrennt lebenden Menschen die Möglichkeit gegeben wird, mit Hilfe des Computers über das Internet eine Moderation gemeinsam abzuhalten. Ein weiterer Vorteil ist es, daß es einfacher ist, jedem Teilnehmer anschließend eine Kopie der in der Moderation entstandenen Dokumente zukommen zu lassen. Jeder Teilnehmer kann sich jederzeit während einer Moderation eine Kopie des aktuellen Ergebnisses sichern, was von einer Pinwand nicht so einfach möglich ist. Karten und graphische Elemente lassen sich auf dem Computer leichter verschieben und

anpassen.
Sowohl vor- als auch nachteilig kann sich die stärkere Anonymität des einzelnen vor seinem Computer auswirken. Zwar kann die stärkere Anonymität helfen, offener über Themen zu diskutieren, dafür fehlt aber der soziale Kontakt unter den einzelnen Teilnehmern, und das direkte Eingehen auf Beiträge anderer fällt schwerer. Auch die Spontaneität des Einzelnen wird durch den Computer gehemmt, da man nicht einfach etwas dazwischen rufen kann. Außerdem wird die Aufmerksamkeit des einzelnen zusätzlich durch die Bedienung des Computers von der eigentlichen Moderation abgelenkt.

2.2. Marktanalyse Metaplan (von Wolfgang Riese)

In diesem Unterkapitel war eine Marktanalyse anhand von vorhandenen Softwarelösungen für Metaplan Moderationen vorgesehen. Leider ist es uns trotz intensiver Internetrecherche nicht möglich gewesen, existierende kommerzielle oder nicht kommerzielle Softwarelösungen für computerunterstützte Metaplan-Moderationen im Internet zu finden.

Auch wenn dies die Vermutung nahelegt, daß es kein Interesse an einem Produkt wie dem von uns entwickeltem gibt, haben wir uns dennoch entschlossen, dieses, unserer Meinung nach nützliche und vielseitige Tool, zu entwickeln.

2.3 Mind Maps (von Henrik Ortlepp)

In diesem Kapitel soll eine Einführung in die Methode der Mind Maps gegeben werden. Zunächst werden in Kapitel 2.3.1 die zugrundeliegenden Gedanken beschrieben, die für den Einsatz von Mind Maps sprechen und die Anwendungsweisen und Regeln werden kurz skizziert (für eine genauere Betrachtung zur Herstellung von Mind Maps, siehe Kapitel 3.2). Es wird weiterhin ein kurzer Überblick über die Vor- und Nachteile von computerbasiertem Mind Mapping gegeben.
Das daran anschließende Kapitel 2.4 beschreibt vier bestehende Computerprogramme, die sich mit der Erstellung von Mind Maps (2.4.1 und 2.4.4) oder nah verwandten Konzepten (2.4.2 und 2.4.3) beschäftigen.

Abbildung 4 - von Hand erstellte Mind Map [Reimann 1998]

2.3.1. Mind Map Methode (von Henrik Ortlepp)

Mind Maps (kognitive Karten) sollen die Strukturierung und Visualisierung von Themenkomplexen und Problemstellungen erleichtern. Dies geschieht durch das schrittweise Anlegen einer hierarchischen Struktur von Zeichnungen und knappen Kurzformeln, bei der sich die Wurzel im Zentrum befindet. Um diesen Anfangsknoten, der das Thema umreißt, werden die dazugehörigen Teilthemen an linienförmigen sogenannten Ästen und Zweigen rekursiv nach außen entwickelt (für eine genauere Beschreibung siehe Abb. 4 und vergleiche auch Kap. 3.2). Dabei gibt es einige Regeln, welche die fertige Mind Map brauchbarer machen. Hilfreich sind folgende Punkte:

- In der Blattmitte starten
- Schlüsselworte statt viel Text
- Groß-/Kleinschreibung
- Mindestens 3 verschiedene Farben
- Abbildungen, Symbole, Kodes in verschiedenen Größen
- Jedes Wort und jede Abbildung möglichst allein auf eine eigene Linie
- Linien sind miteinander verbunden
- Linien im Zentrum der Map sind dicker und werden zum Rand hin dünner
- Länge der Linie ist genauso lang wie das Wort/die Abbildung
- Punkte hervorheben und mit Assoziationen arbeiten

Das den Mind Maps zugrundeliegende Denkmodell geht davon aus, daß das Gehirn aus einer analytischen linken und einer bildhaft assoziativ arbeitenden rechten Gehirnhälfte besteht. Dadurch, daß in Mind Maps eine visuelle Struktur, eventuell sogar durch Bilder, Skizzen und Zeichnungen angereichert, gewählt wird, sollen beide Gehirnhälften aktiviert und damit die Produktivität gesteigert werden. Obwohl es stark umstritten ist, ob sich eine derart strikte Trennung zwischen den Hirnhälften überhaupt treffen läßt (Untersuchungen an hirnverletzten Patienten deuten darauf hin, daß dies nicht möglich ist [Jungbluth 1998A]), scheint es offensichtlich, daß sich viele Informationen durch Bilder und strukturierte Darstellungen, wie sie beim Mind Mapping gegeben sind, schneller aufnehmen lassen, als ein Text mit vergleichbarem Informationsgehalt. Als anschauliches Beispiel läßt sich ein Familienstammbaum nennen, der, ebenso wie Mind Maps, in einer Baumstruktur vorliegt und beispielsweise die Information, ob eine Person Enkel einer anderen ist, sehr viel schneller zugänglich macht als ein einfacher Text [Reimann 1998].

2.4 Marktanalyse Mind Maps (von Henrik Ortlepp)

In diesem Kapitel sollen nun einige computerbasierte Mind-Mapping-Tools vorgestellt werden. Die Marktanalyse stützt sich dabei im Wesentlichen auf [Jungbluth 1998B] und eine eigene Evaluation der Produkte.

2.4.1 Mind Manager (von Henrik Ortlepp)

Mit dem Mind Manager steht ein vollwertiges und ausgereiftes Programm zur Erstellung von Mind Maps zur Verfügung, das eine Vielzahl von Funktionen realisiert, die in diesem

Rahmen unmöglich alle vorgestellt werden können. Mit intuitiver, an MS Office 97 angelehnter Bedienung lassen sich im Handumdrehen komplexe und sogar mehrdimensionale Mind Maps erstellen, die man wahlweise schon während der Eingabe oder später mit Farben, Grafiken und Textnotizen (Text der bei Anklicken eines Knotens im Zusatzfenster erscheint) bereichern kann. Zum Einfügen von einer der fast 450 vorgegebenen Grafiken stehen verschiedene Symbolgalerien zur Verfügung, die das Grafikangebot in sinnvolle kleinere Sammlungen zerlegen, um die Übersichtlichkeit zu erhöhen. Grafisch ansprechend sind auch die ‚organischen' Äste und Zweige, mit denen die einzelnen Knoten verbunden werden. Diese können auch später noch verändert, verschoben und skaliert werden.

Zusätzlich zu den von einem Mind Mapping-Tool erwarteten Funktionen und intelligent angeordneten Bedienungselementen, bietet der MindManager noch einige Spezialitäten. Dazu gehört beispielsweise, daß jeder Knoten einen Hyperlink darstellen kann, der auf eine beliebige Internet-Seite, ein lokales Dokument oder sogar eine weitere Mind Map verweist. Nützlich ist auch die Möglichkeit, die Mind Map als Bild, strukturiertes Html-Dokument oder anklickbare Imagemap zu exportieren. Und in Abbildung 6 ist unten links zu sehen, wie eine Übersicht angezeigt werden kann, die bei großen Mind Maps das Scrollen vereinfacht. Weitere Funktionen, die aus Textverarbeitungsprogrammen entliehen sind (Rechtschreibprüfung, Suchen/Ersetzen, Tutorials...) runden die Vielfalt ab.

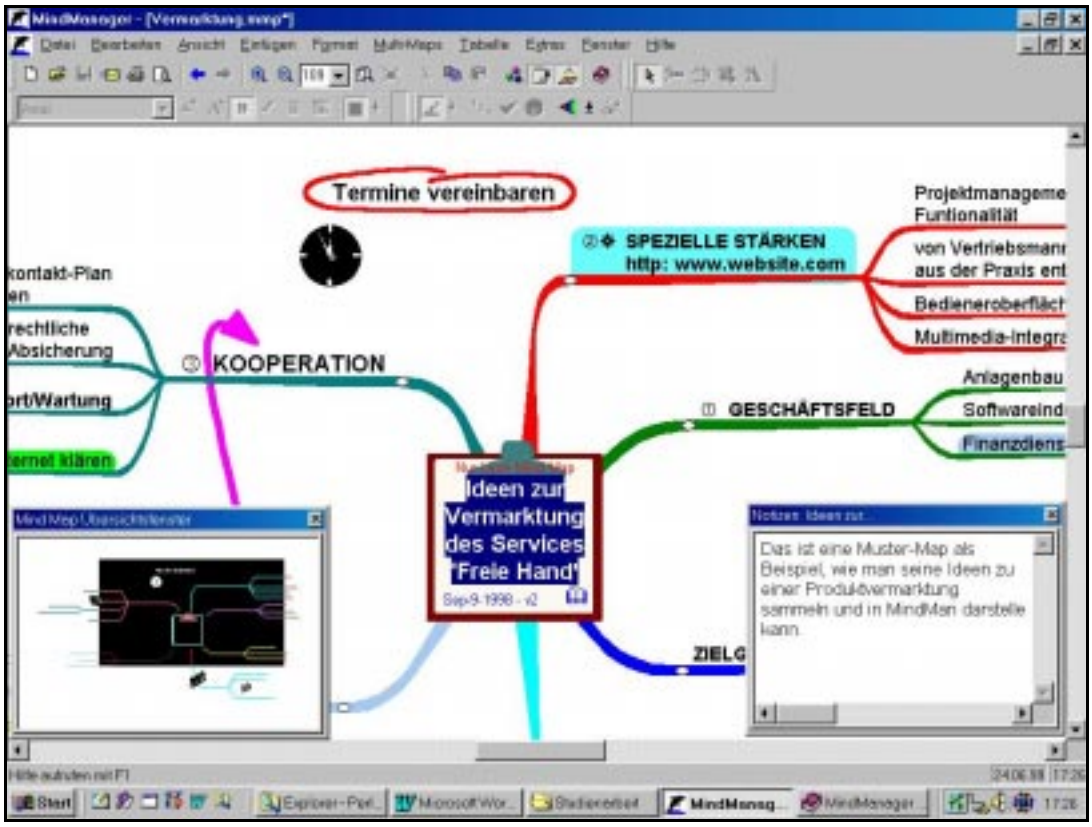


Abbildung 5 – Mind Map aus MindManager [MindManger 1998]

2.4.2. Inspiration (von Henrik Ortlepp)

Bei Inspiration wird besonderes Gewicht auf die Möglichkeit, Ideen schnell eingeben zu können, gelegt. Dazu steht die „Rapid Fire“-Technologie zur Verfügung. Dabei kann der Anwender einfach eintippen, was ihm gerade in den Sinn kommt. Zwischen jeweils zwei durch die Eingabe-Taste abgegrenzten Gedanken, ist im Programm ein roter Blitz zu sehen, der die einzelnen Punkte trennt, den Anwender aber nicht durch aufklappende Fenster oder sonst irgend etwas in seinem Ideenfluß unterbricht. Setzt in der Eingabe eine Pause ein, so ordnet das Programm die Eingaben selbständig den entsprechenden Ebenen zu, zeichnet die passenden Rechtecke und verbindet diese durch unidirektionale Pfeile.

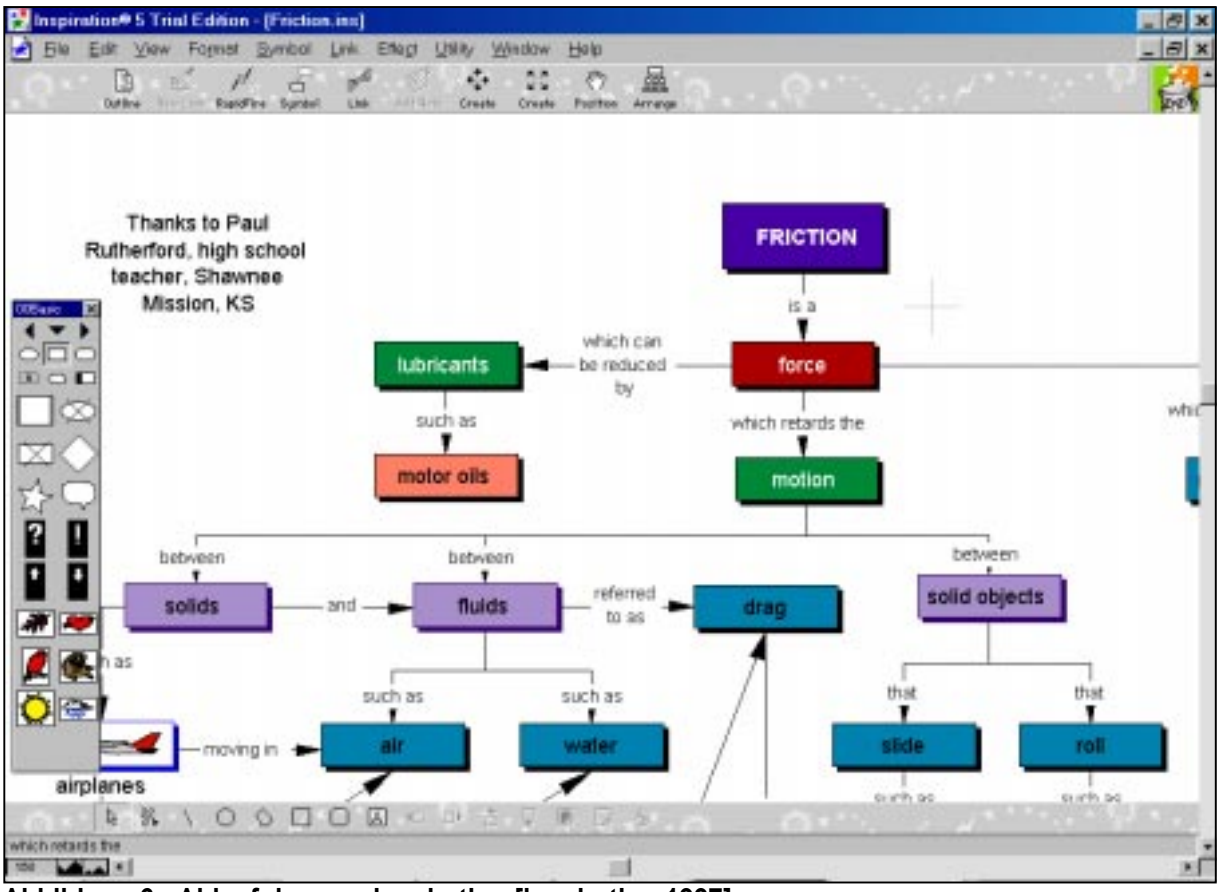


Abbildung 6 - Ablaufplan aus Inspiration [Inspiration 1997]

Zur Nachbearbeitung stehen dann viele Wege offen. So kann das entstandene Schaubild durch Hinzufügen von Grafiken (aus einer gut geordneten Symbolbibliothek von etwa 500 Bildern), Farben und Freitext zu einer Mind Map werden, es ist aber auch möglich, die streng hierarchische Struktur einer Mind Map zu durchbrechen. Damit eignet sich Inspiration beispielsweise besonders zur Erstellung von Ablaufplänen oder komplexeren

Zusammenhängen zwischen den einzelnen Punkten. Möchte man nun aber eine Mind Map erstellen, so fehlen die Beschränkungen, die sich in echten Mind Map – Tools wie Mind Manager (Kap. 2.4.1) oder VisiMap (Kap. 2.4.4) finden.

Wiederum finden sich auch bei Inspiration Export-Möglichkeiten zu Html, und als Grafik. Hyperlinks stehen allerdings nicht zur Verfügung. Besonders angenehm ist, daß die automatische Anordnung der Knotenpunkte durch einfaches Ziehen der Knoten angepaßt werden kann. Die Verbindungen zu anderen Knoten passen sich dabei automatisch an. So läßt sich relativ schnell und intuitiv eine übersichtliche angepaßte Darstellung kreieren.

2.4.3. Decision Explorer (von Henrik Ortlepp)

Concept Maps, wie sie der Decision Explorer von Banxia Software verwendet, unterscheiden sich von Mind Maps vor allem dadurch, daß auch multidirektionale Verbindungen erlaubt sind. Dadurch entsteht im Gegensatz zu den streng hierarchischen Strukturen von Mind Maps ein Netzwerk von Knoten und Verbindungslinien.

Die Kernidee beim Decision Explorer liegt darin, daß mehrere Personen, die an dem selben Problem arbeiten, das Programm als eine Strukturierungs- und Kommunikationshilfe für ihre Gedanken benutzen sollen. Durch die im Bild unten erkennbaren Registerkarten, lassen sich zum gleichen Thema sehr schnell auch die Ideen anderer Mitarbeiter ansehen und mit den eigenen verknüpfen. Dabei steht weniger die grafische Anschaulichkeit (Grafiken kann man nicht einfügen) und intuitive Bedienbarkeit als vielmehr die logische Verknüpfung der Inhalte im Vordergrund. So sind unidirektionale Verbindungen beispielsweise nicht nur reine Grafiken, sondern haben auch für die Programmlogik eine Semantik. Über verschiedene Funktionen lassen sich die Beziehungen der einzelnen Punkte zueinander analysieren und entsprechend die verwandten Probleme zu einem Thema anzeigen. Diese Anzeige läßt sich dann auf vielfache Weise strukturieren und die entscheidenden Punkte können zur Erhöhung der Übersichtlichkeit hervorgehoben werden.

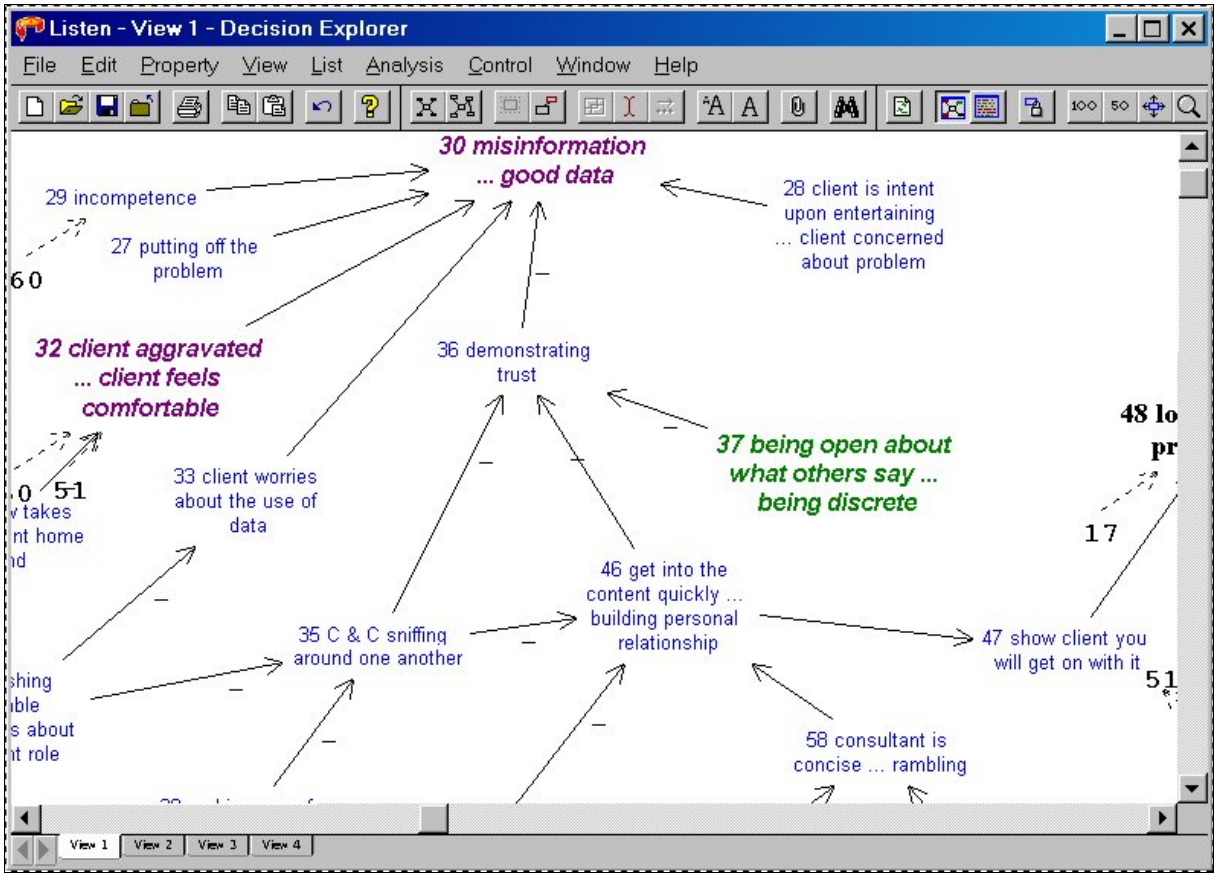


Abbildung 7 - Concept Map aus Decision Explorer [DecisionExplorer 1998]

2.4.4. VisiMap (von Henrik Ortlepp)

Ein weiteres Programm zur Erstellung von Mind Maps ist VisiMap. Bei VisiMap heißen Mind Maps Visual Maps, die sich aber von Mind Maps ansonsten nicht unterscheiden. Bei VisiMap wird besonders auf die schnellstmögliche Eingabeart Wert gelegt. Bei der Eingabe soll der Benutzer sich nicht um das Programm kümmern und damit von dem eigentlichen Thema abgelenkt werden. Der Anwender kann einfach das, was auf dem Ast erscheinen soll, eintippen und das Programm fügt einen Unterpunkt vom aktuell markierten Punkt ein. Angenehm dabei ist, daß bei einem eingetippten Buchstaben automatisch sofort ein Fenster angezeigt wird, in dem der eingetippte Text zu sehen ist und zusätzliche Gestaltungsmöglichkeiten (ein Symbol oder die Art des Zweiges) optional zur Verfügung stehen. Die Methoden zur manuellen Nachbearbeitung und Reorganisation der Zweige lassen allerdings ein wenig zu wünschen übrig. Die Symbole, die den Textknoten hinzugefügt werden können stammen entweder aus der integrierten Bibliothek mit etwa 100 Bildern oder können aus verschiedensten Dateitypen (wie .exe, .dll usw.) extrahiert werden.

Auch Visimap kann die Visual Maps zu anklickbaren Imagemaps für das Internet konvertieren.

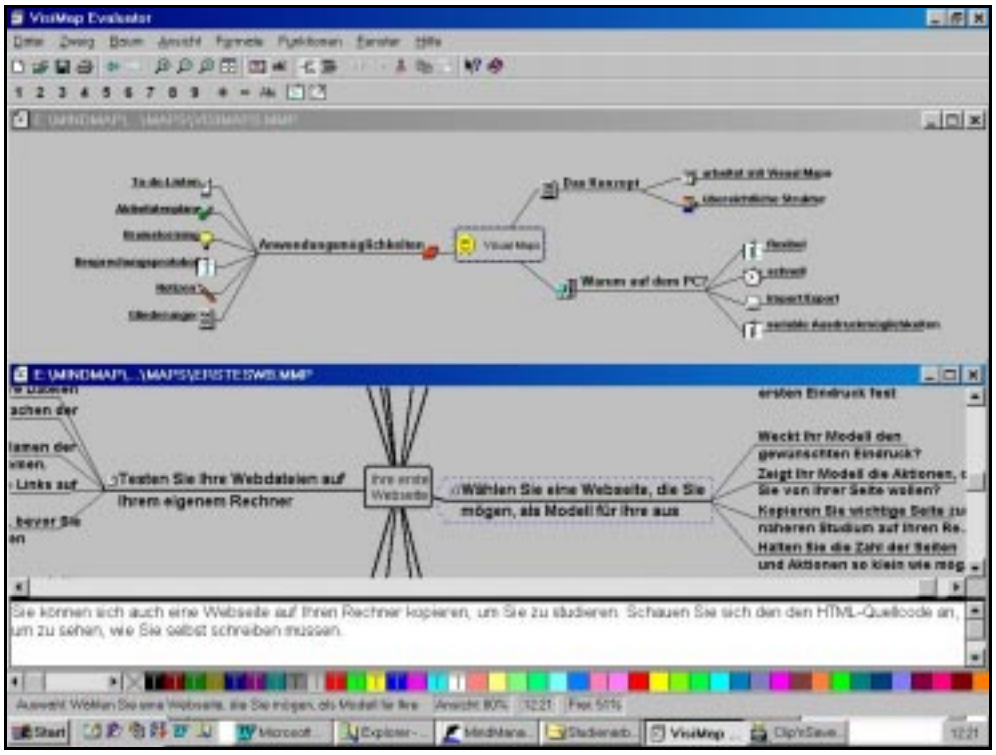


Abbildung 8 – Mind Map aus VisiMap [VisiMap 1998]

2.4.5. Bewertung von computerbasierten Mind Map Tools

Werden Mind Maps mit computerbasierten Tools angefertigt (s.a. Kap. 2.4.1 und 2.4.4), ergeben sich automatisch einige Vorteile und Nachteile:
Ein Vorteil ist mit Sicherheit, daß einmal Geschriebenes nicht feststeht. Bietet das verwendete Tool dies an, so kann die Mind Map, ohne daß alles neu gezeichnet werden muß, umstrukturiert werden, um beispielsweise einen Ast an einer Stelle einzufügen, an der vorher kein Platz mehr war und einfache Verbesserungen können ohne Radiergummi durchgeführt werden. Darüber hinaus lassen sich komplexe Mind Maps durch geeignete Zoom-Funktionen entweder im Überblick oder in Detailansicht darstellen oder können sogar in kleinere Mind Maps der Teilbereiche zerlegt werden. Soll die Mind Map dann aber doch auf Papier gebracht werden, so kann man sie, eventuell mehrfach, in verschiedener Qualität und individuell angepaßt, ausdrucken.

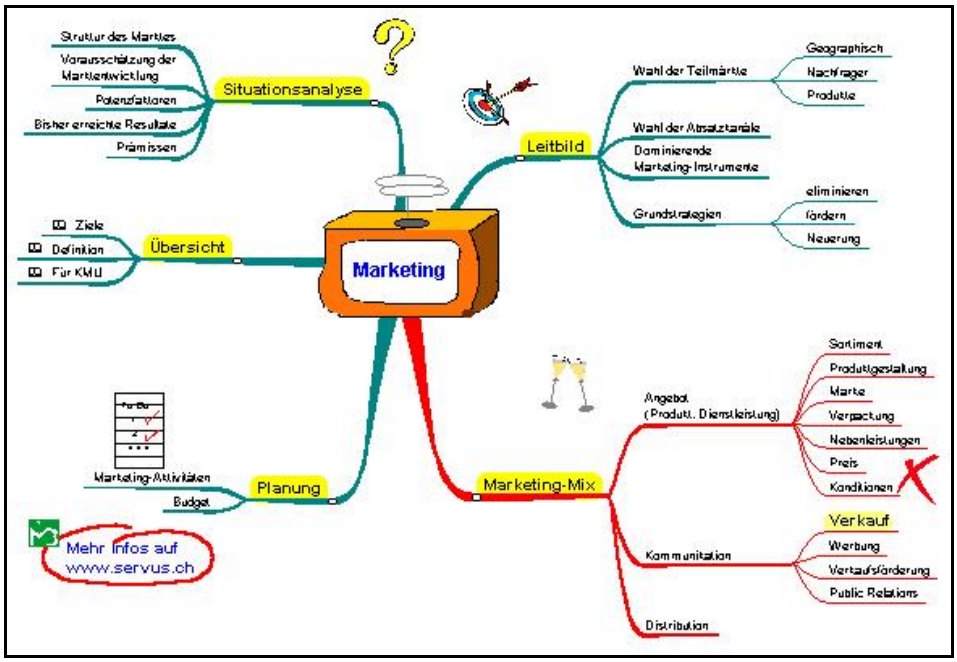


Abbildung 9 - Mind Map aus MindManager [MindManger 1998]

Als nachteilig muß als allererstes gesehen werden, daß das Erstellen einer Mind Map an den Computer gebunden ist. Egal wie gut das verwendete Tool auch sein mag, es wird immer Einschränkungen in der Design-Freiheit geben, die beim Malen auf Papier nicht da wären. Dies kann im schlimmsten Fall dazu führen, daß ein großer Teil der Aufmerksamkeit darauf gerichtet sein muß, wie das Tool zu benutzen ist und würde damit vom eigentlichen Thema stark ablenken. Ein zweiter wesentlicher Nachteil ist die Einbindung von Grafiken und Skizzen. Während man auf Papier einfach „drauflos skizziert“, ist dies am Computer nicht

oder nur mit erheblichem Extraaufwand (Grafiktablett...) möglich. Stattdessen bieten Mind Mapping Tools Grafikbibliotheken, die aber eben nur eine sehr begrenzte Auswahl von Grafiken zur Verfügung stellen können, wenn die Übersichtlichkeit erhalten bleiben soll.

3. Fachlicher Entwurf (von Wolfgang Riese)

Entsprechend dem im WAM-Ansatz [Züllighoven 1998] vorgeschlagenen evolutionären und dokumenten-basierten Vorgehen beim Entwurf von Softwaresystemen haben wir für die Analyse des fachlichen Modells unserer Anwendung jeweils zuerst Szenarios der Ist-Situation für die beiden Teilaspekte, Metaplan und Mind Map, entworfen. Diese haben wir in Form von Handlungsstudien, wie sie im WAM-Buch definiert werden, erstellt.

> Handlungsstudie:
> „Handlungsstudien beschreiben einzelne Schritte zur Erledigung einer Aufgabe im Detail. Oft sind dies Handlungen, wie etwa das Ausfüllen eines bestimmten Formulars oder ein Prüfvorgang, die selbst wieder im Kontext unterschiedlicher Aufgaben auftreten. Typisch für eine Handlungsstudie ist z.B., wenn beim Ausfüllen eines Formulars bestimmte Details zu beachten sind, oder wenn eine bestimmte Prüfung oder Berechnung mehrfach an verschiedenen Stellen durchgeführt wird. Handlungen sind in diesem Zusammenhang konkrete körperliche Bewegungen, geistige Aktivitäten oder maschinelle Aktionen, die sich an den Arbeitsgegenständen und –mitteln orientieren. Sie sind die detaillierteste Form der Unterteilung von Arbeit im Anwendungsbereich. Aus der Beschreibung von Handlungen ist ablesbar, wie etwas genau und unter Verwendung welcher Hilfsmittel geschieht.“
> [Züllighoven 1998], S.612

Wir haben unsere Anforderungsanalysen dabei jeweils durch Literaturrecherche und, soweit möglich, eigene Anwendung der Methoden erarbeitet. Dabei haben wir folgende Literatur verwendet : [Buzan 1974], [Dauscher 1998], c’t 20/98 [Jungbluth 1998], c’t 20/98 [Reimann 1998] und [Schnelle-Cölln & Schnelle 1997].

Anhand dieser Szenarios haben wir dann einen Glossar mit den Fachbegriffen unseres Anwendungsgebietes erstellt.

Anschließend haben wir, auf diesen Dokumenten basierend, unsere Visionen zu einer möglichen computerunterstützten Moderation entworfen und uns Gedanken darüber gemacht, wie eine gemeinsame Software vom „Look & Feel“ her schließlich aussehen sollte. Unsere Visionen haben wir dabei in der Form einer Handhabungsvision verfaßt.

Handhabungsvision :

„ Handhabungsvisionen beschreiben das Aussehen und die Verwendung des künftigen Anwendungssystems. Sie haben also einen statischen und einen dynamischen Aspekt. Die Kernfrage ist: Wie werden die bisherigen und die neuen Aufgaben vom System unterstützt? Die Antwort auf diese Frage sollte immer eine Beschreibung sein, in der eine Aufgabe mit ihren einzelnen Tätigkeiten erledigt wird, wozu dann Werkzeuge, Automaten und Materialien benutzt werden. Ausgangspunkt der Beschreibung ist das Benutzungsmodell, d.h. die Aktionen des Benutzers, die verwendeten sichtbaren Komponenten und die darauf folgende Reaktion des Systems. ...

In einer Handhabungsvision stehen zunächst diejenigen Aufgaben im Mittelpunkt, die in Szenarios beschrieben worden sind und die auch zukünftig durch das System unterstützt werden sollen. Denn diese Aufgaben sind zum einen gut beschrieben, zum anderen sind die Anwender mit ihnen vertraut und werden in diesem Bereich das neue System besonders kritisch unter die Lupe nehmen. Als Ergänzung zu fachlichen und technischen Ablaufvisionen betonen Handhabungsvisionen die Außenansicht eines Werkzeugs. ...“ [Züllighoven 1998], S. 630

Die in den einzelnen Unterkapiteln folgenden Dokumente stellen das Ergebnis unseres evolutionären und dokumenten-basierten Vorgehens da. Dabei handelt es sich um die vorerst abschließenden Dokumente. Diese sind durch die Autor-Kritiker-Zyklen, wie in [Züllighoven 1998] beschrieben, entstanden.

Autor-Kritiker-Zyklen:
„Der evolutionäre Prozeß spielt sich in unseren Projekten in der Form sog. Autor-Kritiker-Zyklen ab. Die Entwickler sind meist die Autoren. Ihre Arbeitsgegenstände sind die bereits genannten Dokumenttypen und Prototypen. Als Kritiker treten die Beteiligten auf, die jeweils das notwendige Fachwissen besitzen. Im Sinne der Anwendungsorientierung sind dies meist die Anwender. Der Autor-Kritiker-Zyklus ist also ein Wechsel zwischen Analysieren, Modellieren und Bewerten. ...“
[Züllighoven 1998], S.9

Evolutionares und dokumentenbasiertes Vorgehen:
„- Jedes Softwareprojekt sollte eine zyklische Abfolge der generellen Aktivitäten Analysieren, Modellieren und Bewerten sein. Dies gilt für die Entwicklung und das Management. Auf die Auswahl geeigneter Autoren und Kritiker ist zu achten.

- Alle für ein Projekt ausgewählten Dokumenttypen sollten prinzipiell im gesamten Projektverlauf bearbeitbar sein. Eine vordefinierte Bearbeitungsreihenfolge existiert nicht. Jeder Dokumenttyp sollte mit Blick auf seinen Zweck und seine Verständlichkeit für Autoren und Kritiker ausgewählt werden."
[Züllighoven 1998], S.545

„- Dokumentenbasierte Modellierung: Der Entwicklungsprozeß findet mit und an Dokumenten statt. Dies scheint auf den ersten Blick nicht neu. Genauer betrachtet heißt dies aber, daß die Beteiligten nur solche Dokumente erstellen und bewerten, die insgesamt das Anwendungssystem ausmachen. Jede Aktivität sollte also in diesem Sinne zielgerichtet sein, und jedes Dokument sollte einen Beitrag zum Anwendungssystem liefern. ...

- Ist- und Sollabgleich: ...

- Permanente Rückkoppelung: ...

"
[Züllighoven 1998], S.546

3.1. Szenarios Metaplan (von Wolfgang Riese)

- Kartenabfrage:

Bei der Kartenabfrage werden vom Moderator an einer Pinwand Fragen visualisiert aufbereitet. Dabei hat der Moderator die Aufgabe den Moderationsteilnehmern deutlich zu machen, worum es genau geht (z.B. Probleme, Erwartungen, Ideen, ...) und wieviel Zeit ihnen für die Beantwortung zur Verfügung steht. Hierfür erhalten die Moderationsteilnehmer Kärtchen, auf die sie dann ihre Antworten zu den Fragen schreiben können.

Bei den Antworten gilt es zu beachten, daß pro Karte nur ein Stichpunkt als Halbsatz mit max. 7 Wörter in 3 Zeilen von den Teilnehmern geschrieben werden sollte.

Wie bei eigentlich allen Methoden der Metaplan-Methode ist es auch bei der Kartenabfrage das Ziel, möglichst ungehemmte, spontane, kreative und unzensierte Gedanken der einzelnen Teilnehmer zu erhalten.

Dabei ist es möglich, die Kartenabfrage anonym, nicht anonym und in Kleingruppen abzuhalten.

1. Die anonyme K. gestaltet sich dabei als die zeitaufwendigste der drei möglichen K..
   Bei ihr werden jedoch die spontansten und am weitesten gestreuten Ideen, Vorschläge und Kritiken erhalten, da durch die Anonymität keiner Angst vor Anfeindungen und Repressalien haben muß.
   Nach dem also alle Teilnehmer mit dem Schreiben ihrer Kärtchen fertig sind bzw. die Zeit um ist, sammelt der Moderator die beschriebenen Kärtchen wieder ein und mischt diese danach. Das sich anschließende Ordnen der Karten dauert nun bei der anonymen K. länger als bei den anderen K.. Denn nun muß durch die Gruppe über die Aussage der einzelnen Karten diskutiert werden, bevor diese durch den Moderator an der Pinwand angeordnet werden können.

2. Die nicht anonyme K. hat den Nachteil, daß der Einzelne länger überlegt, was er auf seine Karten schreibt und so Spontanes eher unterdrückt, da er evtl. in Hierarchien gefangen ist oder schlicht Angst hat, für eine zu abwegige Idee einstehen zu müssen.
   Der Moderator sammelt auch bei der nicht anonymen K. die Kärtchen der Teilnehmer ein, sobald diese fertig sind bzw. die Zeit abgelaufen ist. Er kann sich aber das anschließende Mischen sparen. Nun präsentiert der Moderator ein Kärtchen nach dem anderen, so daß der jeweilige Schreiber falls nötig sein Kärtchen erläutern kann. Dadurch kann ein zügiges Ordnen und Anpinnen der Karten erfolgen.

3. Bei der K. in Kleingruppen besteht der Vorteil, daß die Karten nicht nur von einem Schreiber kommen und so für den Einzelnen eine gewisse Anonymität gewahrt bleibt, was allerdings um den Nachteil einer Vorauswahl durch die Kleingruppe erkauft wird. Das bedeutet, daß bereits in der Kleingruppe über die Vorschläge der jeweiligen Gruppenmitglieder diskutiert wird. Auch hier werden die Kärtchen, nachdem die Kleingruppen mit dem Schreiben fertig sind bzw. die Zeit abgelaufen

ist, durch den Moderator eingesammelt, wobei wiederum kein Mischen nötig ist. Das Anordnen der Karten beschleunigt sich bei der K. in Kleingruppen noch einmal, da nur noch den Moderationsteilnehmern aus den anderen Kleingruppen die Karte vorgestellt werden muß.

Allen Kartenabfragen ist gemein, daß abschließend nach dem Sammeln und ersten Anordnen der Kärtchen noch eine Phase folgen kann, in der noch Oberpunkte gebildet werden. Außerdem kann noch ein gewichtetes Clustern erfolgen.
Wenn auch dies abgeschlossen ist, besteht die Möglichkeit, dazu passende Aussage- und Fragesätze zu bilden, die dann in einer Liste gesammelt werden.

- Zuruffrage:

Bei der Zuruffrage werden ähnliche Ziele verfolgt wie bei der Kartenabfrage.

Die wesentlichen Unterschiede der beiden Werkzeuge sind sowohl die gesteigerte Spontaneität der Zuruffrage als auch die bei ihr nicht vorhandene Anonymität des einzelnen.

Auch in der Zuruffrage hat der Moderator wieder die Aufgabe, zuerst die Frage an einer Pinwand zu Visualisieren und den Moderationsteilnehmern zu erläutern, sowie auf die Einhaltung eines zeitlichen Rahmens zu achten. Nachdem also vom Moderator eine Einleitung ins Thema erfolgt ist, fällt ihm die Aufgabe zu, die nun von den Moderationsteilnehmern zugerufenen Beiträge an der Pinwand zu notieren.

Deshalb eignet sich die Zuruffrage weniger für Situationen, in denen auch Kritik an bestehendem geäußert werden soll. Dafür können die Teilnehmer aufeinander eingehen und sich ergänzen.

Daraus ergeben sich zwei Schwerpunkte für den Einsatz der Zuruffrage:
Zum einen die schnelle Sammlung von bereits bekanntem und zum anderen als Brainstorming, also die spontane und möglichst offene Sammlung von neuen und originellen Ideen.

- Einpunktfragen:

  Die Einpunktfrage dient der kurzfristigen Meinungsbildung durch Aufzeigen der gegenwärtigen Meinungen, Stimmungen oder Erwartungen.

  Dies geschieht, in dem durch den Moderator an einer Pinwand eine klar formulierte Frage gestellt und erläutert wird. Jeder Teilnehmer erhält dann einen Klebepunkt, den er an der Pinwand in ein bewertetes Koordinatenkreuz klebt.

  Daran sollte sich noch eine Phase anschließen, in der versucht wird, das Ergebnis zu interpretieren.

- Mehrpunktfragen:

  Mehrpunktfragen werden eingesetzt, um Prioritäten in einer Menge von Vorschlägen zu setzen.

  Der Moderator hat auch hier die Aufgabe, die Fragen vorher den Moderationsteilnehemern zu erklären und an der Pinwand zu visualisieren.

  Dabei gibt es zwei Vorgehensweisen:

  Entweder wird eine einfache Rangfolge der einzelnen Vorschläge angestrebt, dann werden die Themen in einer Liste aufgeschrieben und jeder Teilnehmer erhält ca. so viele Punkte, wie die Anzahl der Themen durch 2, die er dann frei vergeben kann. Abschließend müssen nur noch die Punkte pro Vorschlag zusammengezählt und das Ergebnis mit der Gruppe diskutiert werden.

  Die andere Möglichkeit ist, daß die Vorschläge ebenfalls in einer Liste aufgeschrieben werden und dann aber jeweils mit einer Skala von 0 bis X versehen werden. Danach erhält jeder Teilnehmer für jeden Vorschlag einen Punkt. Wenn alle ihre Punkte geklebt haben, wird für jeden Vorschlag die Summe gebildet.

Aus den so errechneten Summen ergibt sich jetzt die Prioritäten-Reihenfolge der Themen, die abschließend noch einmal mit der Gruppe diskutiert werden kann.

Die so erhaltenen Ergebnisse bilden die Grundlage für das weitere Vorgehen.

3.2. Szenario Mind Map (von Henrik Ortlepp)

Auf ein weißes Blatt wird in die Mitte das Problem oder das Thema, mit dem es sich zu beschäftigen gilt, geschrieben und mit einem Rahmen (meist eine Wolke oder ein dickerer Rahmen als sonst üblich, um diesen Punkt besonders hervorzuheben) versehen. Davon ausgehend werden Äste in verschiedene Richtungen gemalt, die das Thema in Unterbereiche gliedern oder Assoziationen zum Thema darstellen. Diese Äste können dann rekursiv mit Zweigen versehen werden, in denen weitere Details niedergeschrieben oder gemalt werden. So entsteht nach und nach eine hierarchische Struktur, die die einzelnen Unterbereiche veranschaulicht. Wenn immer möglich, sollen Skizzen oder Bilder gemalt werden, anstatt Text zu verwenden. Dabei können, so weit es sinnvoll erscheint, auch verschiedene Farben eingesetzt werden, um bestimmte Bereiche besonders hervorzuheben oder inhaltliche Zusammenhänge durch Bedeutungszuweisung der Farben herauszustellen. Der Einsatz von Stiften verschiedener Farbe und Dicke darf aber den Gedankenfluß nicht hemmen und nicht vom eigentlichen Thema ablenken. Man sollte sich also von vornherein nur eine begrenzte Anzahl von Stiften bereitlegen (drei Farben sollten reichen), damit man sich nicht zu lange mit der Auswahl beschäftigen muß.

Der zweite Punkt, der beim anfertigen einer Mind Map beachtet werden muß, ist, daß die Übersichtlichkeit erhalten bleibt. Dazu hat es sich als sinnvoll herausgestellt keine vollständigen Sätze, sondern ein bis drei kurze, prägnante Stichworte oder Kurzformeln zu verwenden (besser sogar gar kein Text, sondern Bilder). Darüber hinaus sollte jeder Ast nicht mehr als sieben Verzweigungen aufweisen.

3.3. Die Handhabungsvisionen (von Wolfgang Riese)

In den nun folgenden Unterkapiteln beschreiben wir sowohl die geplanten Gemeinsamkeiten unseres Entwurfes als auch die Unterschiede in den Metaplan und Mind Map teilen.

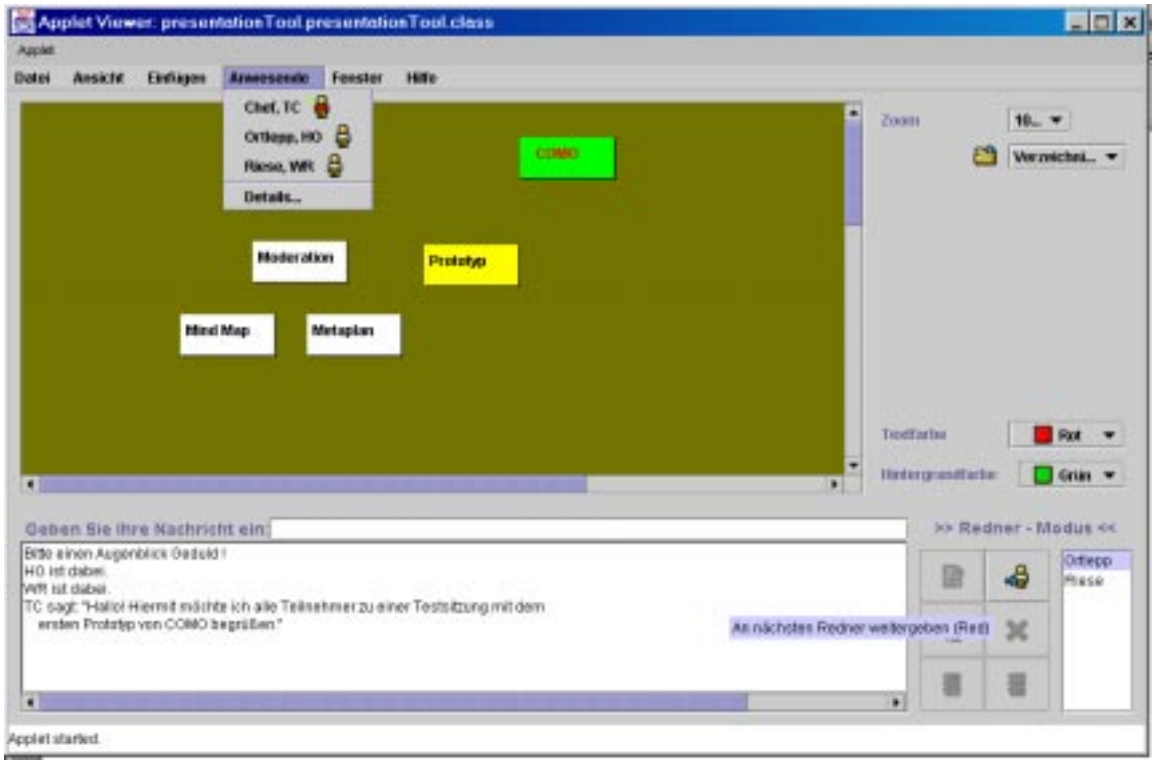


Abbildung 10 - Ansicht des COMO-Prototyps

3.3.1. Vorbereitung einer Moderation (von Wolfgang Riese)

Als erstes startet der Moderator das COMO-Applet auf seinem Rechner. Nun wird er vom Applet in einem Dialogfenster dazu aufgefordert auszuwählen, ob er als 'Moderator' oder 'Teilnehmer' einer Moderation in Erscheinung treten möchte. Nach dem er sich für 'Moderator' entschieden hat, den entsprechende Radiobutton ausgewählt und 'weiter' gedrückt hat, wird er jetzt gefragt, ob er 'offline' oder 'online' am Server arbeiten möchte. Entsprechend seiner Wahl wird er im nächsten Fenster aufgefordert, entweder die IP-Adresse des Servers einzugeben und einen Namen und ein Paßwort für die Moderation festzulegen, oder es wird direkt statt erst im nächsten Schritt gefragt, ob er eine MP- oder eine MM-Moderation leiten möchte. Im anschließenden Schritt kann er die Größe der zu verwendenden Arbeitsfläche (Pinwand) festlegen. Dabei kann er wenn er online arbeitet sehen, ob sich noch jemand am Server für die aktuelle Moderation angemeldet hat. Da sich der Moderator aber erst mit der Planung einer späteren Moderation beschäftigt, wird er vermutlich alleine arbeiten und gleich weiter zur eigentlichen Arbeitsumgebung wechseln. Dort kann er die Arbeitsfläche (Pinwand) für die spätere Moderation mit den Teilnehmern vorbereiten. Die fertig vorbereitete Arbeitsfläche (Pinwand) kann dann von ihm lokal oder auf dem Server abgespeichert werden und eine weitere in Angriff genommen werden.

Die grafische Benutzeroberfläche (GUI) des Metaplan (MP)- und des MindMap (MM)-Teils sind weitestgehend identisch, lediglich die spezifischen Werkzeuge und Materialien sind jeweils ausgetauscht.

In der Arbeitsumgebung stehen zusätzlich ein Chat und eine Rednerliste zur Verfügung.

Jetzt kann der Moderator aus einer vorgefertigten HTML-Seite eine Startseite für die Teilnehmer der Moderation machen. Standardmäßig enthält diese Seite einen Link auf das Applet, der entsprechend angepaßt werden kann, so daß dem Applet beim Start die IP-Adresse des Servers gleich mitgegeben wird und es als Teilnehmer Applet gestartet wird. Zusätzlich sollte auf ihr noch etwas zum Thema und dem Zeitpunkt der Moderation vermerkt werden.

Wenn alle diese Vorbereitungen getroffen sind und die Seite im Web veröffentlicht ist, kann eine Einladung an alle Teilnehmer per e-mail geschickt werden, in welcher der Tag, die Uhrzeit, das Thema, der Name und das Paßwort der Moderation sowie die dazugehörige Webseite bekanntgegeben werden.

3.3.2. Die Moderation (von Wolfgang Riese)

Der Moderator startet die online Moderation durch sein Applet, wie in Kapitel 3.3.1 'Vorbereitung der Moderation' beschrieben. Nach dem Auswählen der Größe der Arbeitsfläche wartet der Moderator jetzt jedoch darauf, daß sich alle Teilnehmer angemeldet haben. Während er wartet, wird er auf dem Bildschirm darüber informiert, wer sich schon alles angemeldet hat.

Die Teilnehmer der Moderation starten das Applet über die Webseite. Falls die Webseite nicht entsprechend angepaßt wurde, müssen sie jetzt noch auswählen als 'Teilnehmer' an der Moderation teilzunehmen und die IP-Adresse des Servers angeben. Im nächsten Schritt oder sonst direkt erhalten sie eine Übersicht über alle auf dem Server laufenden Moderationen. Jetzt muß der Teilnehmer nur noch die entsprechende Moderation auswählen und das zugehörige Paßwort eingeben. Der Teilnehmer befindet sich jetzt in der Arbeitsumgebung, kann aber noch nicht ins Geschehen eingreifen. Wenn sich alle angemeldet haben oder der Moderator nicht länger warten will, wechselt auch er in die eigentliche Arbeitsumgebung.

Jetzt in der Arbeitsumgebung angekommen, lädt der Moderator die erste seiner vorbereiteten Arbeitsflächen (Pinwände).

Über den Chat kann der Moderator jetzt noch etwas zum Thema der Moderation sagen, die Teilnehmer Nachfragen stellen, oder es kann eine Diskussion stattfinden.

Um an der Arbeitsfläche (Pinwand) arbeiten zu können, muß der Moderator die Bearbeiterliste mit dem 'Freigeben'-Button frei geben. Danach können sich alle über den 'Eintragen'-Button in die Liste eintragen. Der erste auf der Liste kann die Arbeitsfläche (Pinwand) bearbeiten, sobald der Moderator, der zu Beginn die Kontrolle hat, auf den 'Nächster'-Button drückt. Jetzt kann der nun aktive Teilnehmer die Arbeitsfläche (Pinwand) bearbeiten. Sobald er fertig ist, drückt er den 'Nächster'-Button. Nur der aktive Teilnehmer und der Moderator können den 'Nächster'-Button drücken. Zusätzlich zu dem 'Eintragen'- und

dem 'Nächster'-Button hat der Moderator die Möglichkeit, die Liste mit dem 'Leeren'-Button zu leeren, sie mit dem 'Schließen'-Button zu schließen, oder zwischenzeitlich selbst durch Drücken des 'Unterbrechen'-Buttons die Kontrolle zu übernehmen. Außerdem steht an der Rednerliste extra die Anzahl der derzeit eingetragenen Redner.

Jeder Teilnehmer kann sich nach belieben eine lokale Kopie der aktuellen Arbeitsfläche (Pinwand), über einen entsprechenden Menüeintrag, speichern, um diese später weiter zu bearbeiten. Zusätzlich kann durch den Moderator die aktuelle Version der Arbeitsfläche mit einer Versionsnummer unter Zuhilfenahme eines entsprechenden Verwaltungswerkzeugs auf dem Server gespeichert werden.

Die gesamte Moderation kann nur vom Moderator durch Auswahl des entsprechenden Menüeintrags beendet werden.

Wenn der Moderator irgendwann einmal keine Zeit mehr haben sollte, kann er aber auch einen Teilnehmer als neuen Moderator bestimmen. Dazu erhält er eine Liste der aktuell in der Moderation angemeldeten Teilnehmer. Aus dieser kann er dann denjenigen auswählen, der sein Nachfolger werden soll. Bei diesem erscheint dann ein Popup-Window, in dem gefragt wird, ob er bereit ist, die Moderation als Moderator fortzuführen.

3.3.3. Die Metaplan-Sitzung (von Wolfgang Riese)

Die MP-Umgebung soll es ermöglichen, über ein Netzwerk zu mehreren eine visuell unterstützte nicht anonyme Kartenabfrage durchzuführen. In einer späteren Version könnten noch Komponenten für eine anonyme Kartenabfrage, Ein- und Mehrpunktfragen hinzugefügt werden.

In der MP-Umgebung hat der jeweils aktive Teilnehmer die Möglichkeit, für die Materialien die aktuelle Farbe zu bestimmen (weiß, grün, orange oder gelb). Anschließend werden die danach zur Arbeitsfläche (Pinwand) hinzugefügten Karten in der entsprechenden Farbe dargestellt. Lediglich die Wolke gibt es immer nur in weiß mit rotem Rand. Sobald der Bearbeiter eines der Materialien ausgewählt hat, erscheint ein kleiner Texteditor, in dem er das Material entsprechend den MP-Regeln beschriften kann. Danach erscheint die Karte oder Wolke dann auf der Arbeitsfläche (Pinwand), wo sie mit der Maus frei positioniert werden kann. Zusätzlich gibt es noch ein Textwerkzeug, mit dem man die Arbeitsfläche (Pinwand)

beschriften kann, ein Werkzeug um Linien zu ziehen, mit denen Karten verbunden werden können, eines um freihändig Linien zu ziehen, um Karten zu gruppieren, eines um Raster für Tabellen zu erstellen und eines um Blitze an Karten zu setzen.

3.3.4. Die Mind Map-Sitzung (von Henrik Ortlepp)

Über das Menü „Datei“ erstellt der Moderator eine neue Mind Map bzw. lädt eine bereits vorhandene. Soll eine neue Mind Map erstellt werden, so muß der Moderator den Startknoten in der Mitte des Zeichenbereiches festlegen und benennen, sowie die in Kapitel 3.3.1. beschriebenen Texte schreiben, um andere Teilnehmer zur Mind Map-Session einzuladen. Über den Chat wird dann der Start der Session bekannt gegeben. Alle Teilnehmer, die die Webseite ansehen und sich bei dem Moderator registriert haben (s.o.), sollten jetzt in ihrem Browser-Fenster das COMO-Applet sehen, indem sich die von dem Moderator vorbereitete Mind Map befindet.

Oben rechts steht ein Zoom-Tool zur Verfügung, mit dem zwischen Übersichtsansicht und Detailansicht gewechselt werden kann. Darunter ist die Grafikauswahl zu sehen. Über die Ordnerauswahl kann ein Grafikordner gewählt werden, der einige Grafiken zu einem Thema enthält und diese dann bei Auswahl darunter zur Verfügung stellt. In einer späteren Version sollte es möglich sein, sich im passiven Modus eigene Grafikordner zusammenzustellen, damit die Bearbeitung der Mind Map im aktiven Modus so vorbereitet werden kann. Im Chat unten links können Kommentare zur Mind Map oder andere Nachrichten an den Moderator oder andere Teilnehmer geschrieben werden. Alle im Chat eingetippten Nachrichten gehen an alle Teilnehmer.

Sobald der eigene Name der oberste in der Rednerliste ist, wird man vom System dazu aufgefordert aktiv zu werden und sollte dies auch tun, da alle anderen warten müssen. Man hat jetzt die Möglichkeit, die Mind Map (für alle sichtbar) zu verändern. Durch einfachen Mausklick auf einen bestehenden Knoten in der Mind Map wird dieser Knoten ausgewählt. Über das Menü „Einfügen“ oder idealer Weise über Drag-and-Drop läßt sich dann eine Grafik aus der Grafikauswahl rechts in die Mind Map als Unterpunkt einfügen. Der verbindende Ast/Zweig wird vom System automatisch gezeichnet und der Teilnehmer wird aufgefordert, einen optionalen Text zum Bild zu schreiben. Ist kein Knoten ausgewählt, so kann eine Grafik per Drag-and-Drop frei in der Mind Map positioniert werden. Es wird dann kein Zweig gezeichnet. Mit einem Doppelklick auf einen Knoten kann dieser editiert (d.h. der Text

editiert oder das Bild ausgetauscht) werden. Über die Farbauswahlen rechts kann das jeweils ausgewählte Element mit einer Farbe und einer Hintergrundfarbe versehen werden.

Hat man die aktive Bearbeitung an der Mind Map abgeschlossen, übergibt man, wie im Kapitel 3.3.2. beschrieben, mit dem Knopf „Nächster" die Aktivität an den nächsten auf der Rednerliste aufgeführten Teilnehmer.

Der Moderator überwacht die Session und kann auch selbst als Teilnehmer aktiv werden. Zusätzlich besitzt sie/er das Recht, jederzeit die Aktivität zu übernehmen (Knopf "Unterbrechen") oder die Rednerliste zu löschen (Knopf "Liste Löschen"), um z.B. die Session zu beenden.

3.4. Zusammenfassung (von Wolfgang Riese)

In diesem Kapitel haben wir unsere Handlungsstudien bzw. Aufgaben-Szenarios und Handhabungs-Visionen in ihrer jetzigen Version vorgestellt. Die Szenarios stellen dabei das Ergebnis unserer Recherche bezüglich des klassischen Einsatzes der Metaplan und Mind Map Methoden ohne Computer dar, wohingegen die Visionen unsere Auffassung eines möglichen computerunterstützten Moderationssystems als Ergebnis widerspiegeln. Diese Dokumente stellen für das weitere Vorgehen im nächsten Kapitel und bei der Programmierung die zentralen Dokumente dar, anhand derer wir das fachliche Modell des Anwendungsgebietes ableiten können.

4. Technischer Entwurf (von Wolfgang Riese)

In diesem Kapitel beschäftigen wir uns damit, auf welche Weise wir unseren fachlichen Entwurf in ein lauffähiges Programm überführt haben.

Im besonderen gehen wir dabei auf die verschiedenen technischen Konzepte ein.

Außerdem erklären wir, wie weit wir uns am WAM Ansatz [Züllighoven 1998] orientiert haben und warum.

Danach gehen wir kurz darauf ein, warum wir uns für die Programmiersprache JAVA bei der Umsetzung unseres Projektes entschieden haben.

Im anschließenden Unterkapitel 4.2 gehen wir auf verschiedene Entwurfsentscheidungen ein und erläutern unsere Entscheidungen.

In diesem Zusammenhang ist noch zu erwähnen, daß wir in den folgenden Kapiteln Klassennamen, Methodennamen und Attribute in `Courier` gesetzt haben, wobei sich an Methodennamen immer auch noch ein Paar Klammern anschließen und private Attribute mit einem _ anfangen.

Bsp.:

- Klassenname : `ListOfSpeakersTool`
- Methodenname : `notifyObservers()/notifyObservers(object,...)`
- Attribute : `_currentBoard`

Schließlich zeigen wir unsere Klassenhierarchie und gehen näher auf unsere Material-, Werkzeug- und Interaktions- Klassen ein.

Zum Abschluß des Kapitels zeigen wir noch kurz exemplarisch ein leicht vergröbertes Interaktions-Diagramm.

4.1. Entwurfskonzept (von Wolfgang Riese)

Zu Beginn unserer Planung über das weitere Vorgehen hatten wir uns nach längeren Überlegungen darauf geeinigt, uns am WAM Ansatz des Arbeitsbereiches-Softwaretechnik des Fachbereichs Informatik der Uni-Hamburg zu orientieren.

Dies haben wir vor allem deshalb gewählt, da wir ein evolutionäres und dokumentenbasiertes Vorgehen, wie es im WAM-Buch [Züllighoven 1998] vorgestellt wird, für die Realisierung unserer Studienarbeit als sehr sinnvoll erachten.
Wir haben uns jedoch entschlossen, nicht das vorhandene JWAM-Framework zu verwenden, da uns der Einarbeitungsaufwand in diesem Zusammenhang zu groß erschien.

Mit der hier vorliegenden Studienarbeit haben wir nun einen ersten Demonstrationsprototypen erstellt. Sollten wir diese Arbeit mit einer Diplomarbeit fortsetzen, dann werden wir aus unseren Erfahrungen mit unserem Prototypen und den in dieser Studienarbeit enthaltenen Dokumenten ein Pilotsystem erstellen.

> Demonstrationsprototypen:
> „Ein Demonstrationsprototyp unterstützt die Projektinitiierung: Er soll dem Auftraggeber zeigen, wie ein Anwendungssystem prinzipiell aussehen kann. Er soll Entwicklern und Anwendern eine erste Vorstellung von der Handhabung und dem Einsatzkontext geben. Demonstrationsprototypen werden meist vom Management bewertet.“ [Züllighoven 1998], S.638
>
> Pilotsystem:
> „Ein Pilotsystem wird im Anwendungsbereich eingesetzt und bewertet. Es ist ein technisch ´gereifter´ Prototyp. Ein erstes Pilotsystem entspricht oft dem Kernsystem der zukünftigen Anwendung. Es wird evolutionär durch Ausbaustufen ergänzt. Ein Pilotsystem bietet eine komfortable und sichere Bedienbarkeit und ein Mindestmaß an Benutzungsdokumentation.“ [Züllighoven 1998], S.638

Bei unserem technischen Entwurf haben wir uns daran gehalten, eine Trennung von Funktion und Interaktion zu erhalten, wie sie im WAM-Buch [Züllighoven 1998] S.234 vorgeschlagen wird.

Für die nachstehenden Ausführungen erfolgt die Definition von Funktionskomponente und Interaktionskomponente:

> „Die Funktionskomponente (FK) ist der bewirkende und sondierende Teil des Werkzeugs. In ihr wird die fachliche Funktionalität des Werkzeugs festgelegt. Die FK bearbeitet das Material und kennt den Arbeitszusammenhang, der durch das Werkzeug unterstützt wird...“ [Züllighoven 1998], S.236

> „Die Interaktionskomponente (IAK) legt die Benutzungsschnittstelle des Werkzeugs fest. Dazu nimmt sie Ereignisse entgegen, ruft die FK und steuert die Präsentation an der Oberfläche...“ [Züllighoven 1998], S.236

Ferner haben wir eine Unterscheidung von Werkzeug und Material vorgenommen, wie dies im WAM-Buch [Züllighoven 1998] beschrieben ist.

Definition von Werkzeug und Material:

> „Ein Werkzeug unterstützt wiederkehrende Arbeitsabläufe und -handlungen. Es ist bei unterschiedlichen Aufgaben und Zielsetzungen nützlich. Ein Werkzeug wird von seinem Benutzer je nach den Erfordernissen einer Situation gehandhabt oder wieder zur Seite gelegt. Es schreibt keine festen Arbeitsabläufe vor. Als Softwarewerkzeug ermöglicht es den interaktiven Umgang mit den Arbeitsgegenständen.“ [Züllighoven 1998], S.6

> „Materialien sind die Arbeitsgegenstände, die schließlich zum Arbeitsergebnis werden. Materialien werden mit Werkzeugen entsprechend bearbeitet. Softwarematerialien verkörpern ‚reine‘ anwendungsfachliche Funktionalität. Sie werden niemals direkt benutzt und stellen sich auch nicht selbst dar. Ein Softwarematerial ist durch sein Verhalten, nicht durch seine Struktur charakterisiert.“ [Züllighoven 1998], S.6

Die verschiedenen Dokumente (Szenarios, Glossar, Visionen...), die wir in diesem Zusammenhang entwickelt haben, finden sich alle in den entsprechenden Kapiteln dieses Dokuments wieder.

Hier noch einmal ein kurzer Überblick über unser generelles Vorgehen:

1. Ist-Zustand über Literatur-Recherche und eigenes Vorwissen ermitteln.

2. Die Szenarios und das Glossar erstellen.

3. Vorlage der Szenarios und des Glossars zur ersten Rezension.

4. Interne Überarbeitung der Dokumente anhand der Anmerkungen aus Schritt 3.
   Vollendung des 1. Autor-Kritiker-Zyklus (Def. Autor-Kritiker-Zyklus siehe WAM-Buch [Züllighoven 1998]) über Schritte 2-4.

5. Erste Vision eines möglichen Prototypen und seiner möglichen Benutzung erstellen.

6. Vorlage der Visionen zur ersten Rezension.

7. Interne Überarbeitung der Dokumente anhand der Anmerkungen aus Schritt 6.
   Vollendung des 1. Autor-Kritiker-Zyklus über Schritte 5-7.

8. Erstellung einer Klassen-/Benutzthierarchie der möglichen Materialien und Werkzeuge anhand der Szenarios, Visionen, des Glossars und CRC Cards.

9. Implementierung des Prototypen.

10. Erstellung des abschließenden Dokuments.

4.1.1. JAVA, warum? (von Wolfgang Riese)

Zu Beginn unseres Projektes stellte sich uns die Frage, in welcher Art von Programmiersprache wir unser System realisieren wollten.

Wir haben uns dabei für JAVA entschieden, da uns diese für die vorliegende Problemstellung, der Entwicklung eines Multiusersystems in einer Intra- oder Internet Umgebung, am geeignetsten erschien.
Hinzu kommt, daß wir ein verteiltes System entwerfen wollten, welches möglichst ohne großen Aufwand auf verschiedenen Systemen laufen soll. Hierfür erschien uns Java mit seinen bereits vorhandenen Konzepten prädestiniert. Insbesondere die Möglichkeiten, die sich aus dem Einsatz von Applets ergeben, gefallen uns sehr gut.

Dazu ist anzumerken, daß ein Applet ein kleines Programm ist, welches nicht allein lauffähig ist, sondern für den Einsatz in Intra- und Internet gedacht ist. Um ein Applet ausführen zu können benötigt man entweder einen Appletviewer der Firma SUN oder einen Web Browser wie z.B. den Netscape Comunicator oder den MS Explorer und eine HTML-Seite, in der das Applet referenziert wird. Für den Anwender ergibt sich daraus unter anderem der Vorteil einer erhöhten Sicherheit gegenüber einem normalen Programm: Da er durch den Browser bzw. den Appletviewer dem Applet den Zugriff auf lokale Ressourcen wie z.B. Festplatte oder Drucker verhindern kann. Ebenso kann er verhindern, daß das Applet Daten an beliebige Empfänger im Netzwerk verschickt. Ein weiterer Vorteil ist der, daß für fast jedes verfügbare System heutzutage Java fähige Web Browser zur Verfügung stehen, und ein Applet so ohne Änderung auf sehr vielen verschiedenen Plattformen einsetzbar ist. Durch die Applet-Technologie erhält man außerdem eine interessante Möglichkeit des Softwarevertriebs, da man es so einrichten kann, daß nur der Betreiber des Servers eine Lizenz braucht. Die einzelnen Teilnehmer hingegen müssen nur das beim Veranstalter vorliegende Applet adressieren und können dann ohne selber eine Lizenz zu benötigen an einer Moderation teilnehmen.

4.2. Klassenhierarchie (von Henrik Ortlepp)

Zunächst soll im Kapitel 4.2.1 eine Übersicht über die gesamte Klassenstruktur gegeben werden. Alle Klassen im Package `presentationTool` werden in Vererbungs- und Benutzt-Beziehungen dargestellt. Es läßt sich außerdem klar die Aufteilung in die drei Komponenten *Interaktionskomponente* (auch „Graphical User Interface“ - GUI oder „View“ nach Java-Terminologie), *Funktionskomponente* (auch „Controller“ nach Java-Terminologie) und *Material* (auch „Model“ nach Java-Terminologie) erkennen. In der vorliegenden Version werden noch nicht alle diese Klassen verwendet. Durch das vorliegende Entwurfskonzept soll vielmehr die weitere Arbeit vorgedacht und konzeptionell erschlossen werden. Die folgenden Kapitel 4.2.2 bis 4.2.4 beschäftigen sich nacheinander mit diesen drei Komponenten etwas näher, geben detailliertere Teilübersichten und stellen auch die Aufgaben der einzelnen Klassen kurz vor. Die Klassen werden allerdings auch in den Teilübersichten nicht vollständig mit allen Methoden und Attributen erklärt, sondern es werden nur die für das Verständnis des Entwurfs wichtigen Eigenschaften behandelt.

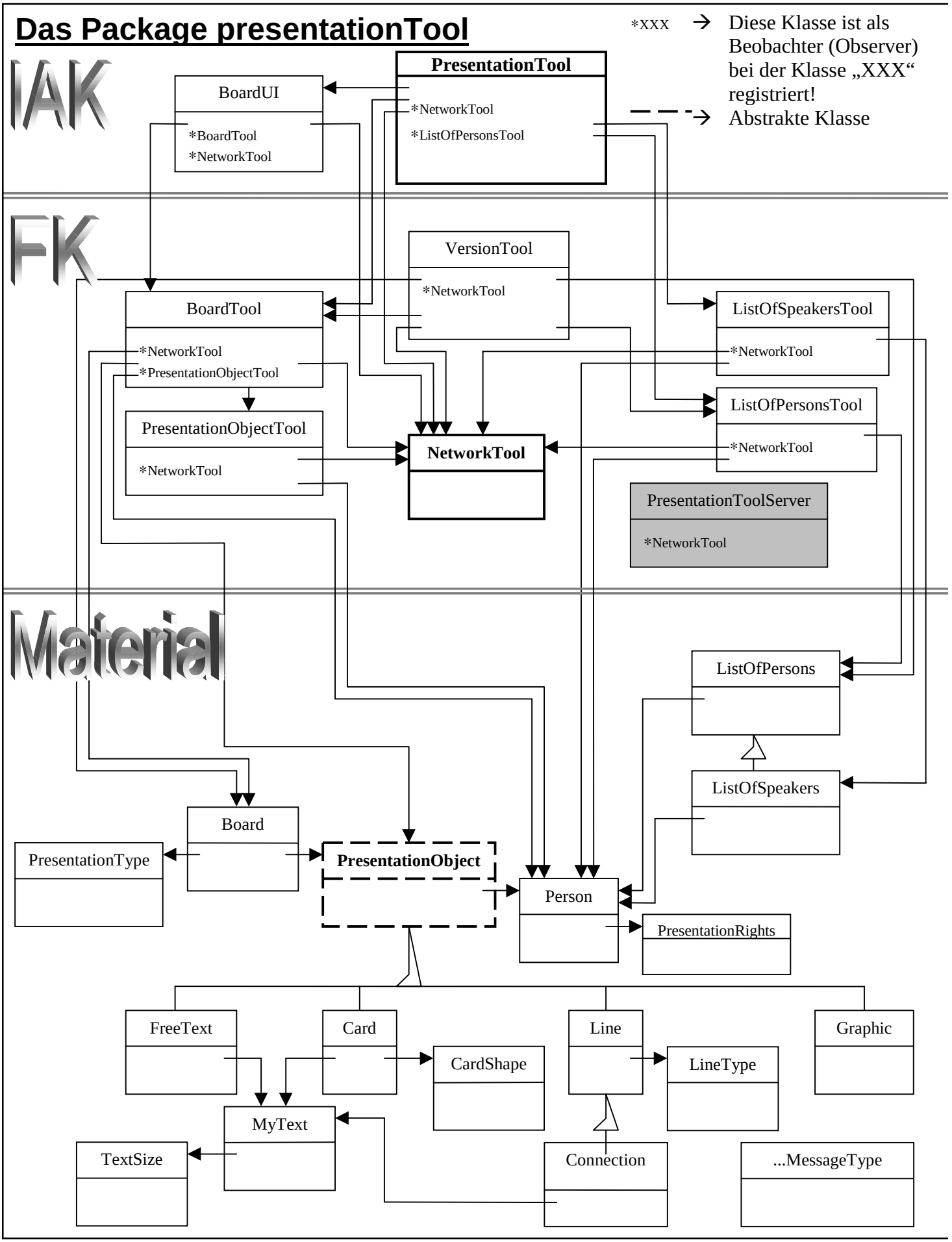
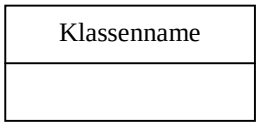


Abbildung 11 – Klassendiagramm CoMo

4.2.1. Überblick (von Henrik Ortlepp)

In der Abbildung auf der vorherigen Seite ist ein Überblick über die Klassen im Package presentationTool zusammengestellt. Darin werden folgende Symbole verwendet:



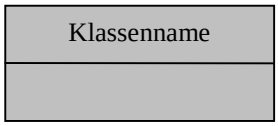
→ Eine Java-Klasse mit dem angegebenen Namen

→ Eine „Benutzt-Beziehung“ zwischen den Klassen

→ Die Klassen dieser Ebene bilden die Interaktionskomponente und werden in Kapitel 4.2.4 näher vorgestellt.

→ Die Klassen dieser Ebene bilden die Funktionskomponente und werden in Kapitel 4.2.3 näher vorgestellt.

→ Die Klassen dieser Ebene bilden die Materialien des Entwurfs und werden in Kapitel 4.2.2 näher vorgestellt.



→ Die grau hinterlegten Klassen benötigen besondere Erklärungen und werden im folgenden ausführlicher behandelt.

Die Klasse `PresentationToolServer` stellt die Serverfunktionalität für die Applets zur Verfügung. Sie ist eine Erweiterung der Klasse `Multiserver` des Packages fict von Jeff Breidenbach (Copyright May 1996) welcher wiederum auf Beispiel 9-3 aus [Flanagan 1998] beruht.

Nachrichten (Messages), die über die Methode `send()` der Klasse `NetworkTool` verschickt werden, erreichen bei korrekter Konfiguration den PresentationToolServer. Dieser speichert die Nachricht lokal und leitet sie ggf. an alle Teilnehmer-Applets (Clients) weiter. Möchte ein neuer Teilnehmer an einer laufenden Moderation teilnehmen, so erhält er in der aktuellen Version nach erfolgreicher Anmeldung vom Server alle Nachrichten, die dieser im Laufe der Moderation verschickt hat. Dieses Verfahren soll als erste Arbeitslösung gedacht sein, um zu gewährleisten, daß auch der neue Teilnehmer konsistente Daten zur Verfügung bekommt. In späteren Versionen sollte dieses Verfahren, bei dem ja beispielsweise hinzugefügte und später wieder gelöschte Objekte doppelt übertragen werden, überarbeitet und verbessert werden (s.a. Kap. 6.)

Zu jeder Client `Tool` Klasse gehört außerdem eine `MessageType` Klasse, in der die Nachrichtentypen der jeweiligen Tool Klasse definiert werden. Dies gibt dem Empfänger einer Nachricht das Wissen, worum es in der betreffenden Nachricht geht.

4.2.2. Materialien, Behälter & Fachwerte (von Henrik Ortlepp & Wolfgang Riese)

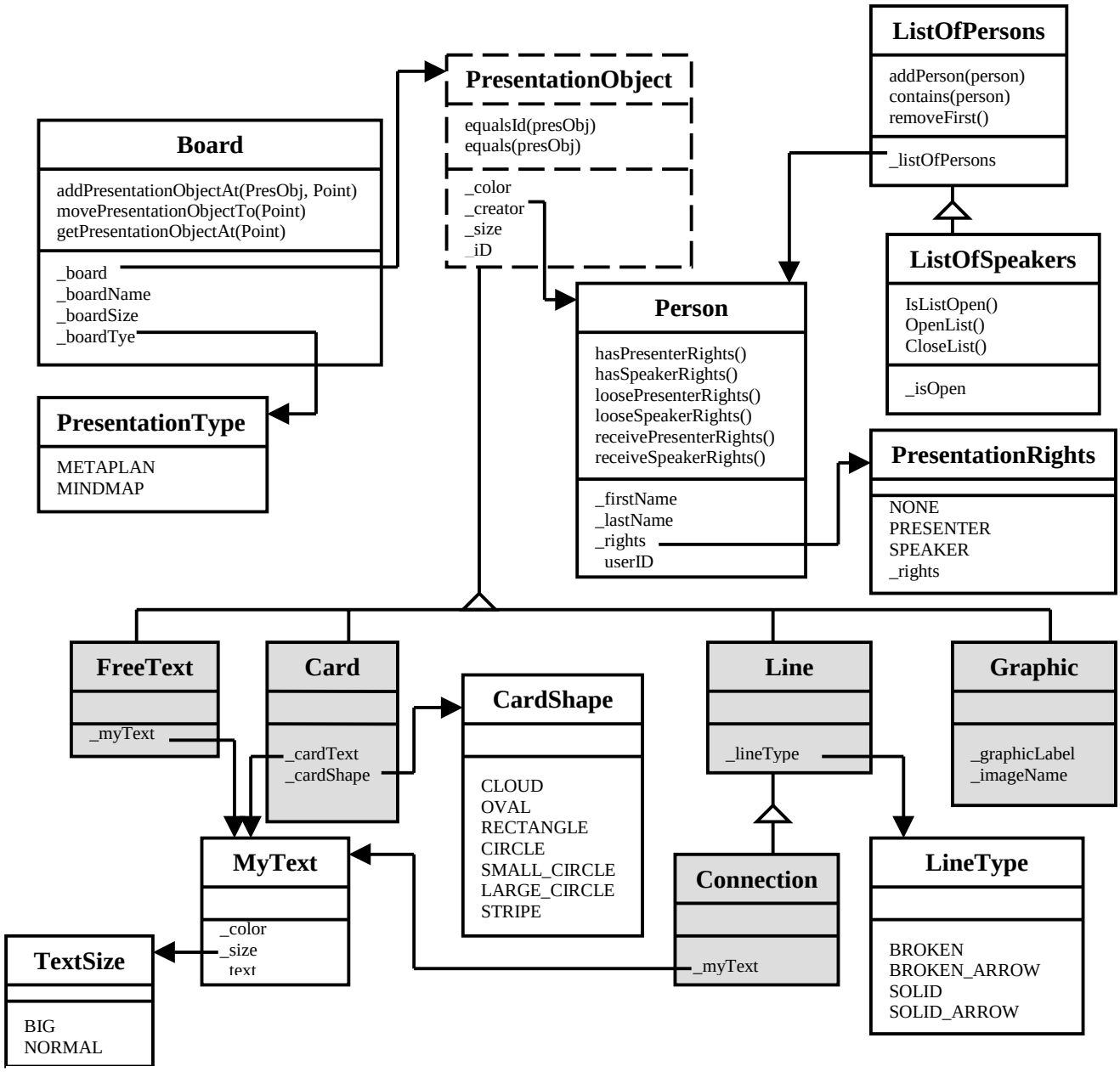


Abbildung 12 – Klassendiagramm Material

In dieser Übersicht sind jetzt alle wichtigen Basis Materialien, ihre Behälter und fünf der modellierten Fachwerte zu sehen. Wobei noch anzumerken wäre, daß Behälter im entsprechenden Kontext ihrer Werkzeuge auch als Materialien zu betrachten sind.

Zentrale Klassen in der Gruppe der Materialien sind zum einen die Klasse `Board`, die den Behälter Arbeitsfläche implementiert. Zugegriffen wird auf die Methoden dieser Klasse durch das zuständige Werkzeug, das in der Klasse `BoardTool` (s.Kap.4.2.3) realisiert ist und dessen IAK sich in der Klasse `BoardUI` (s.Kap.4.2.4) wiederfindet. Herausgestellt sind in der Grafik oben die Methoden, die sich mit der Manipulation von den Objekten auf der Arbeitsfläche (`PresentationObject`) beschäftigen. Es läßt sich außerdem erkennen, daß eine Arbeitsfläche die in der die Werte kapselnden Klasse `PresentationType` festgelegten Arbeitsflächen-Typen annehmen kann.

Die Klasse `Board` speichert neben den Eigenschaften der Fläche selbst auch Referenzen auf die Objeke, die sich auf der Arbeitsfläche befinden und ihre Position. Diese Objekte sind Instanzen einer Subklasse der abstrakten Klasse `PresentationObject`. In der vorliegenden Version stehen dafür die grau hinterlegten Klassen `FreeText` – für frei plazierbaren Text – `Card` – für Karten, wie sie bei Metaplan-Moderationen verwendet werden – `Line` – für Linien und Pfeile zur Strukturierung der Arbeitsfläche – `Connection` – als Verbindung zwischen zwei Objekten, wie es bei Mind Maps gewünscht ist – und `Graphic` – für Grafiken in Mind Maps – zur Verfügung. Besonders interessant dabei ist die Klasse `Connection,` die das objektbasierte Strukturmuster Composite [Gamma et al. 1996] implementiert. Dies bedeutet hier, daß ein `Connection`-Objekt immer zwei Objekte vom Typ `PresentationObject` beinhaltet. Dadurch wird es möglich, die für Mind Map nötigen Baumstrukturen zu bilden. Zusätzlich können die `Connection`-Objekte auch noch beschriftet werden.

Die möglichen Werte von einigen dieser Eigenschaften der Präsentationsobjekte werden wiederum gekapselt in eigenen Klassen bereitgestellt: `MyText` und `TextSize` bieten nur die Operationen auf dem Text der Präsentationsobjekte an, die für eine Moderation sinnvoll sind und den entsprechenden Regeln der Präsentationsformen Rechnung tragen. `CardShape` bietet genau die in Metaplan-Moderationen verwendeten Kartenformen an und `LineType` beschränkt sich auf wenige Formen von Linien, die sich auch in Mind Maps und Metaplan-Moderationen wiederfinden. Werden dem Programm weitere Präsentationsformen hinzugefügt, können sehr einfach auch neue Subklassen der Klasse `PresentationObject` neue Objekte für die Arbeitsfläche bilden.

Schließlich gibt es noch ein weiteres Material: Die Teilnehmer, die in der Klasse `Person` implementiert sind. Sie haben als Eigenschaften ihre Vor- und Nachnamen, eine Benutzerkennung (die im Chat und bei jedem neu eingefügten Objekt benutzt wird) und die Präsentationsrechte, die wiederum als Werte in der Klasse `PresentationRights` gekapselt sind. Diese Rechte lassen sich durch die gezeigten Methoden abfragen. Dabei werden folgende drei Rechte unterschieden:

- keine extra Rechte (`PresentationRights.PARTICIPANT`)
  → Der Teilnehmer darf nur passiv an der Moderation teilnehmen oder sich am Chat beteiligen

- Redner-Rechte (`PresentationRights.SPEAKER`)
  → Der Teilnehmer darf die Arbeitsfläche bearbeiten

- Moderator-Rechte (`PresentationRights.PRESENTER`)
  → Der Moderator kann jederzeit den Redner unterbrechen, neue Arbeitsflächen erstellen, die Rednerliste unterbrechen oder löschen und Moderationen eröffnen.

Alle Teilnehmer einer Moderation (bzw. ihre Person-Objekte) werden in der `ListOfPersons` geführt. Diese wird in der aktuellen Version im Menü Anwesende angezeigt. Die `ListOfSpeakers`, stellt die Rednerliste dar.

Zu den Materialien ist abschließend noch zu sagen, daß alle Funktionskomponenten *niemals* direkt ihre Materialien an die Interaktionskomponenten weitergeben, sondern immer nur *Kopien*. Genauso akzeptieren die Funktionskomponenten *niemals* geänderte Materialien von der Interaktionskomponente. Objekte der Klasse PresentationObject werden z.B. an der Funktionskomponente durch ihre in der gesamten Moderation eindeutige presObjID ausgewählt und dann von der entsprechenden Funktionskomponente bearbeitet. Bei anderen Objekten wird der Funktionskomponente zwar zur Identifikation ein entsprechendes Objekt von der Interaktionskomponente übergeben, dies dient aber nur dazu, in den passenden Behältern nach einem entspechenden Objekt zu suchen, um dann mit diesem weiter zu arbeiten.

4.2.3. Werkzeuge (von Henrik Ortlepp)

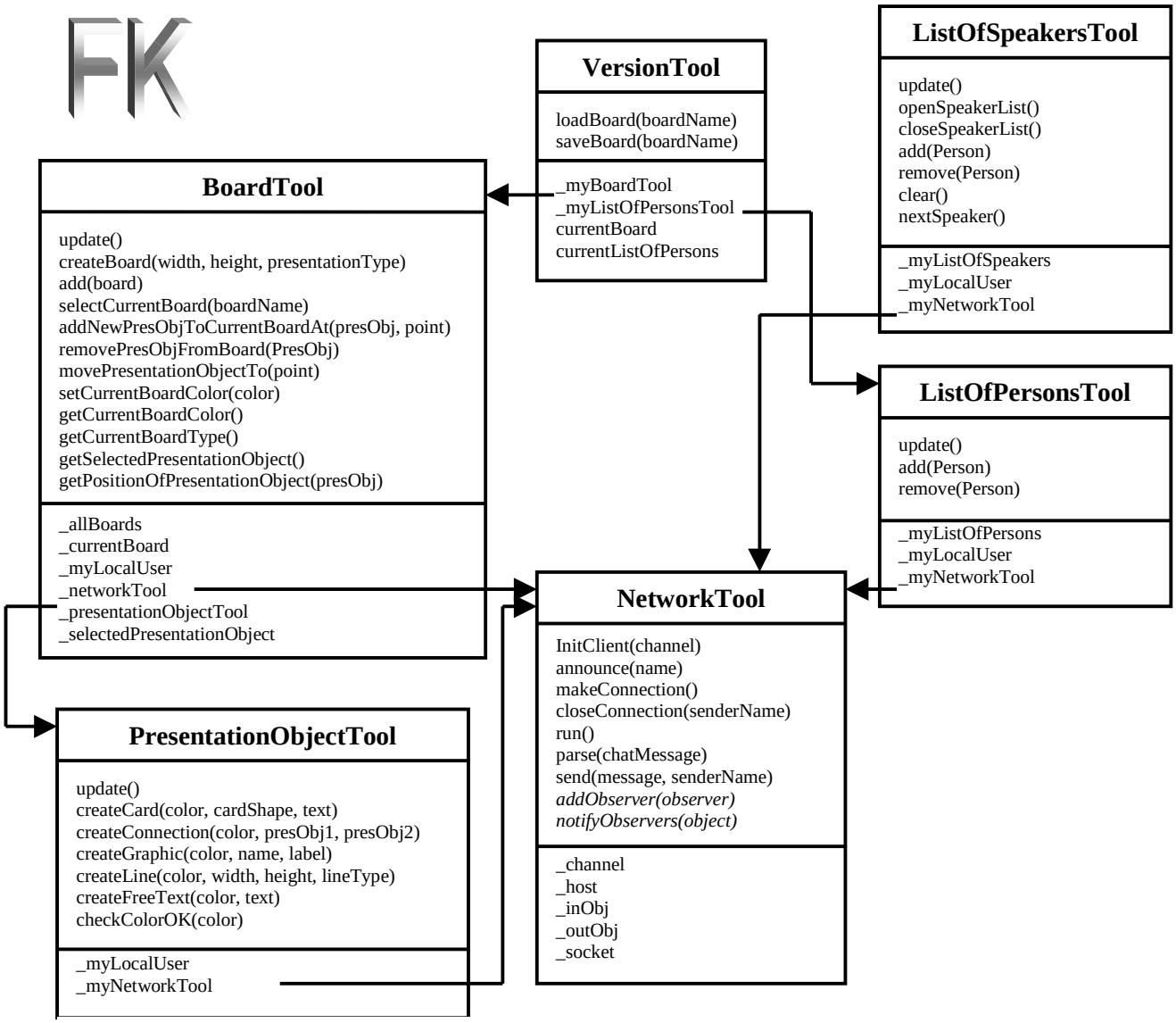


Abbildung 13 – Klassendiagramm

Zunächst fällt auf, daß alle Tools eine Methode `update()` aus dem Interface Observer implementieren. Über diese Methode werden Änderungen der Klasse, bei der sich die Klasse mit der update-Methode als Observer (Beobachter) angemeldet hat, übertragen, wenn diese die Änderungen mit der Methode `notifyObservers(object)` bekannt macht. Alle diese Tools haben sich bei der Klasse NetworkTool als Observer registriert, um über Nachrichten aus dem Netz informiert zu werden.

Die Klasse `NetworkTool` spielt eine zentrale Rolle, denn sie wird von allen anderen Tools benutzt. Dies dient vor allem zur oben angesprochenen Registrierung als Observer. Sie ist für jegliche Kommunikation über den Server zuständig. Neben den aus der Superklasse geerbten Methoden `addObserver()` und `notifyObservers()` stehen zum einen Methoden,

um die Verbindung zum Server zu verwalten (`initClient(), makeConnection(), announce(), closeConnection()`) und zum zweiten Methoden zum Versenden und Empfangen der Messages (`send(), runparse()`) zur Verfügung.

Das `ListOfPersonsTool` verwaltet die Liste aller Teilnehmer, die sich an einer Moderation beteiligen, während sich das `ListOfSpeakersTool` mit der Liste der Redner, die im Arbeitsbereich unten rechts zu sehen ist, beschäftigt. Beide Werkzeuge haben Methoden zum Hinzufügen und Löschen von Personen (`add(), remove()`) sowie Referenzen auf die jeweilige Liste (`_myListOfPersons` bzw. `_myListOfSpeakers`) und eine Refenrenz auf den lokalen Teilnehmer (`myLocalUser`) also die Person, die diesen Arbeitsbereich bedient. Zusätzlich gibt es im `ListOfSpeakersTool` noch Methoden zur Verwaltung der Rednerliste, die durch die entsprechenden Knöpfe der Interaktionskomponenten ausgelöst werden (`clear(), nextSpeaker(), openSpeakerList(), closeSpeakerList()`).

Die Klasse `BoardTool` ist für die eigentliche Arbeit des Teilnehmers auf der Arbeitsfläche sowie für die Verwaltung der verschiedenen Arbeitsflächen selbst (`createBoard(), selectCurrentBoard(), addBoard()`) zuständig. Zur Arbeit mit den Materialien auf der Arbeitsfläche stehen die Methoden Hinzufügen (`addNewPresObjToCurrentBoardAt()`), Entfernen (`removePresObjFromBoard()`) und Bewegen (`movePresentationObjectTo()`) zur Verfügung. Das `BoardTool` verwaltet dabei auch die Position der Objekte auf der Arbeitsfläche und speichert diese zusammen mit der Referenz auf die Objekte in dem aktuellen Board (`_currentBoard`). Das jeweils markierte Objekt der Arbeitsfläche (`_selectedPresentationObject`) kann dann über das ebenfalls referenzierte `PresentationObjectTool` (`_presentationObjectTool `) weiter bearbeitet werden.

`PresentationObjectTool` regelt das Erstellen und Bearbeiten von PresentationObjects. Es stehen Methoden zur Verfügung, um die Materialien, die sich auf der Arbeitsfläche plazieren lassen, zu erzeugen und diese zu bearbeiten (in späteren Versionen).

Die Klasse `VersionTool` wird in der vorliegenden Version noch nicht benutzt. Sie soll in späteren Versionen das Öffnen und Schließen mehrerer `Boards` verwalten und enthält dazu

Referenzen auf die Klassen, die die zu speichernden Daten verwalten (`BoardTool` und `ListOfPersonsTool`). (s.a. Kap. 6.)

4.2.4. Benutzungsschnittstellen

(von Henrik Ortlepp)

Rechts sind die Klassen der Klassenübersicht gezeigt, die für die Anzeige des Applets, also die Benutzungsschnittstelle oder IAK, verantwortlich sind.

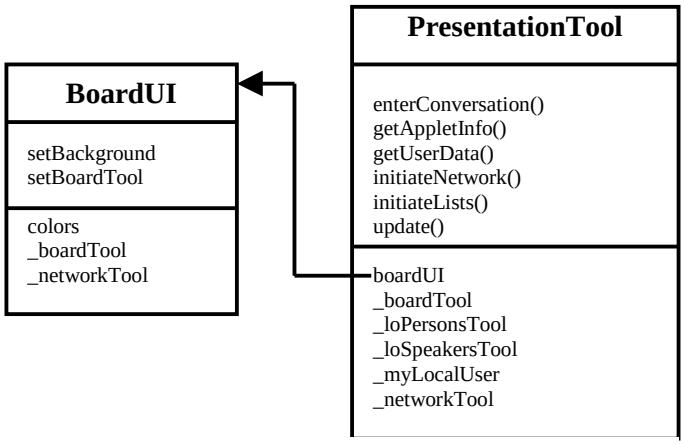


Abbildung 14 – Klassendiagramm

Den Rahmen für das Applet bildet die aus JApplet abgeleitete Klasse PresentationTool. In dieser Klasse befindet sich die Implementation der gesamten Benutzungsschnittstelle bis auf die Arbeitsfläche (Board), die in der Klasse BoardUI realisiert wird. BoardUI wiederum steht zu PresentationTool in Benutzt-Beziehung. Ebenso werden auch das BoardTool, das ListOfPersonsTool, das ListOfSpeakersTool und das NetworkTool von PresentationTool benutzt. So werden Änderungen, die der Benutzer im Applet vornimmt, an die entsprechenden Werkzeug-Klassen in der FK-Ebene weitergeleitet oder direkt durch Aufruf der Methode `send()` im NetworkTool an den Server verschickt. PresentationTool registriert sich auch als Beobachter (`java.util.Observer`) bei den Klassen NetworkTool und ListOfPersonsTool. Dadurch wird bei einer Änderung, die diese Klassen betrifft die Methode `update()` aufgerufen, die wiederum für die Ansicht relevante Änderungen abfängt und die Änderungen veranlaßt.

Zur Vereinfachung beinhaltet die PresentationTool allerdings nicht ausschließlich Interaktionskomponenten, sondern realisiert auch eine gewisse Funktionalität – den Chat; denn ein ChatTool gibt es bisher nicht (s.a. Kap. 6.). Dies ergab sich dadurch, daß die Chatfunktionalität auf den *Free Internet Conferencing Tools* (*F.I.C.T.*) von Jeff Breidenbach (s. [Breidenbach 1997]) aufsetzt und die dort gewählte Struktur übernommen wurde. Der Server im *F.I.C.T.*-Package ist wiederum eine Erweiterung von Beispiel 9-3 in [Flannagan 1998] (s.a. Kap.4.2.1).

Die Klasse BoardUI ist für die Anzeige der Arbeitsfläche zuständig und für das Abfangen von Benutzereingaben hierauf. Eingaben und Änderungen des Benutzers werden an das

NetworkTool weitergeleitet und Änderungen im Netz durch Beobachten des NetworkTools übernommen.

4.3. Interaktionsdiagramm (von Wolfgang Riese)

In diesem Kapitel zeigen wir nun einen kleinen beispielhaften Programmablauf als Interaktionsdiagramm.

In [Gamma et al. 1996] S.404 wird ein Interaktionsdiagramm wie folgt definiert :

> „Ein Interaktionsdiagramm zeigt die Reihenfolge auf, in welcher Anfragen zwischen Objekten ausgeführt werden. ... In einem Interaktionsdiagramm verläuft die Zeit von oben nach unten.“

Im unten abgebildeten Diagramm wird in der oberen Zeile angezeigt, wo die darunter stehenden Objekte existieren, und zwar ob auf dem Client oder dem Server. Wobei Client und Server räumlich getrennt existieren und nur durch ein Netzwerk verbunden sind.
Die senkrechten durchgezogenen Rechtecke sind Objekte, die während der ganzen beobachteten Zeit aktiv existieren. Bei dem Rechteck mit den unterbrochenen Linien handelt es sich um ein Objekt, das zwar während der ganzen Zeit existiert aber in den Zeiten mit unterbrochener Linie passiv ist.

In diesem Beispiel kann man erkennen, wie verändernde Operationen in unserem jetzigen Ansatz über das Netz übertragen werden, um dann erst beim Serverecho lokal endgültig ausgeführt zu werden. Dadurch kann der Bearbeitende erkennen und sicher sein, daß mindestens seine eigene Netzwerkverbindung noch steht und seine Daten zum Server übertragen wurden, wenn bei ihm lokal seine Aktion ausgeführt wurde.

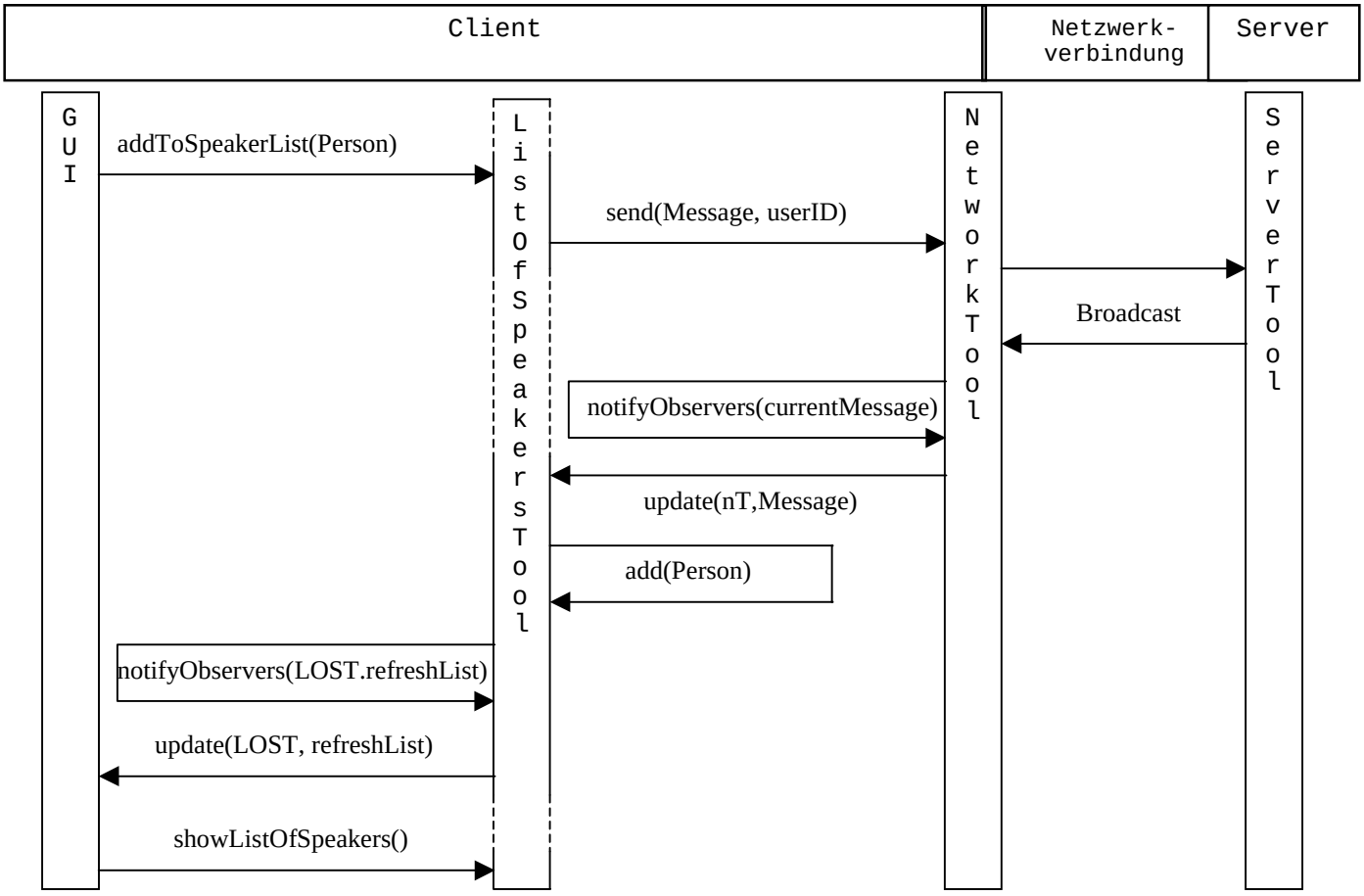


Abbildung 15 - Interaktionsdiagramm zu unserem Client/Server Ansatz

4.4. Umsetzung (von Wolfgang Riese)

Das Ergebnis unserer Umsetzung liegt in Form eines Applets und eines eigenständigen Server-Programms vor. Das Applet stellt dabei die Benutzerschnittstelle und das eigentlich verarbeitende System, während das Server-Programm in seiner jetzigen Form nur als Relais und Verlaufsspeicher für die im Netz befindlichen Applets in einer laufenden Moderation auftritt.

Unser Server-Programm besteht dabei nur aus Funktionskomponenten für die Datenzwischenspeicherung und die Datenübermittlung übers Netz, denn jede Interaktion mit den Benutzern erfolgt durch das Applet, genau wie die Bearbeitung der Materialien in unserem System ebenfalls nur durch die Werkzeugkomponenten des Applets stattfindet.

In unserem Entwurf wird die Trennung von Werkzeug und Material deutlich durch die Einteilung in Tool-Klassen (Werkzeuge) und deren Materialien, wie z.B. `ListOfSpeakersTool` (Werkzeug) und `ListOfSpeakers` (Material).

Hierzu ist noch anzumerken, daß `ListOfSpeakers` nicht nur ein Material, sondern zusätzlich ein fachlicher Behälter ist. Gleiches gilt auch für die `ListOfPersons,` von der die `ListOfSpeakers` erbt.

Bei unserem Entwurf haben wir an vielen Stellen Gebrauch vom Observer Entwurfsmuster ([Gamma et al. 1996] S.257) gemacht.
Dieses Entwurfsmuster beschreibt, wie voneinander abhängige Komponenten Informationen über die Änderung von internen Zuständen miteinander austauschen können, ohne daß eine zu große Abhängigkeit zwischen den einzelnen Komponenten aufgebaut werden muß. Einer späteren Wiederverwendung des Beobachteten stehen so keine zu großen Abhängigkeiten mit seinem Beobachter im Wege. So kann man beispielsweise leichter eine Benutzeroberfläche (Interaktionskomponente) gegen eine andere austauschen, ohne die darunterliegenden Funktionskomponenten mühsam an die neue Oberfläche anpassen zu müssen.

Hierbei haben wir uns zunutze gemacht, daß dieses Entwurfsmuster bereits in Java implementiert ist. So konnten wir ohne größere Probleme unsere Interaktionskomponenten über eine lose Koppelung mit den Werkzeugen (Funktionskomponenten) verbinden.

Dabei haben wir uns dafür entschieden, daß in Java bereits implementierte Push-Modell des Observermusters voll auszunutzen. Wir haben dabei keine zu großen Abhängigkeiten zwischen den einzelnen Komponenten gebildet und die Funktionskomponenten mußten auch nicht zuviel über die Interaktionskomponenten wissen.

In unserem Entwurf beobachten jedoch nicht nur die Interaktionskomponenten die einzelnen Tools, um durch diese auf fachliche Veränderungen in ihren Materialien hingewiesen zu werden, sondern auch alle Tools überwachen das `NetworkTool,` um auf Veränderungen ihrer Materialien auf anderen Clients im Netz reagieren zu können.

An mehreren Stellen haben wir auch Fachwerte nachgebildet.
Definition von Fachwert nach [Züllighoven 1998] S.62 :

> „Ein Fachwert ist ein benutzerdefinierter Wert. Er repräsentiert Werte im Anwendungsbereich.
> Ein Fachwert ist ein Werttyp mit definierter Wertmenge und festgelegten Operationen. Seine innere Repräsentation ist verborgen.
> Fachwerte werden in objektorientierten Sprachen als Klassen definiert. Wichtig ist, daß die Exemplare eines Fachwerts immer Wertsemantik besitzen, d.h. daß ein einmal gesetzter Wert nicht mehr verändert werden kann.“

So haben wir z.B. `PresentationType`, `PresentationRights` und `CardShape` als Fachwerte implementiert, was wir dadurch realisiert haben, daß die Klassen zum einen die Werte als final static Klassen Variablen enthalten und zum anderen keine verändernden Opperationen auf ein erzeugtes Objekt zugelassen sind.

Zur Datenübertragung der einzelnen Komponenten übers Netz und zum Teil auch untereinander haben wir zusätzlich eine Behälter-Klasse, nämlich die `Message` eingeführt. Ein Objekt der Klasse `Message` enthält sowohl einen der von uns eingeführten `MessageType`s (Werte) als auch bis zu zwei weitere Material Objekte. Die Objekte, die diese Nachrichten zur Laufzeit erhalten, können anhand des Nachrichtentyps erkennen, ob diese Nachricht sie interessiert und wenn ja, was mit den dazugehörigen Daten zu passieren hat, oder wo sich ein Material verändert hat und das entsprechende Werkzeug dazu näher zu sondieren ist.

5. Rückblick über den Arbeitsprozeß

In diesem Kapitel gehen wir jetzt noch auf die Probleme ein, die sich durch unser Vorgehen ergaben.

Dabei trennen wir in Probleme technischer Art, die sich auf Probleme mit dem Entwurf allgemein beziehen, und in solche, die durch die Art unserer Zusammenarbeit entstanden.

5.1. Technische Schwierigkeiten und Lösungen

Bei der Integration vorhandener Teillösungen sollte nicht nur frühzeitig geklärt werden, in wie weit der Funktionsumfang zu dem angestrebten Programm paßt, sondern auch, ob sich der Entwurf und die Konstruktionsart in den eigenen Entwurf sinnvoll einfügen lassen. Im vorliegenden Fall eignete sich die benutzte Teillösung (das FICT-Package von Jeff Breidenbach [Breidenbach 1997]) zwar recht gut, es mußte jedoch noch sehr viel abgeändert werden, damit der Programmteil auch in die Struktur von ***COMO*** paßte. Noch immer ist dieser Prozeß nicht vollkommen abgeschlossen (s.a. Kap. 6.1 – Punkt 6).

5.2. Kooperationsschwierigkeiten und Lösungen

Ein wesentliches Problem bei der Erstellung eines Prototyps von zwei Personen ist sicherlich die Kooperation und Kommunikation. Groß angelegte Kooperationstools, die die Versionskontrolle von einzelnen Klassen und das Zusammenführen verschiedener Bearbeitungen derselben Datei ermöglichen, scheinen für die Arbeit zu zweit an einem überschaubaren Projekt wie dem vorliegenden zu aufwendig und überdimensioniert. Trotzdem tauchen diese Probleme auf. Gerade bei einem Kleinprojekt, wie hier, in dem es nur ein Package gibt, muß ständig abgestimmt werden, wer wann an welcher Datei welche Änderungen vornimmt, um sich zeitaufwendiges „Merging“ zu sparen.

Auch die Herangehensweise an das Programmieren selbst muß abgesprochen werden. Zwischen verschiedenen Arbeitsmethoden (beispielsweise orthogonales Ausprogrammieren und gleichzeitiges Testen einer Komponente contra horizontaler Implementation aller Grundfunktionen) muß ein für alle Beteiligten akzeptabler und sinnvoller Weg gefunden werden.

Für das Schreiben des Quellcodes selbst, sollten frühzeitig genaue Styleguides festgelegt werden, damit eine Nachbearbeitung entfällt. Auch auf die Art und Weise Kommentare zu schreiben sollte sich geeinigt werden, damit diese später flüssig zu lesen sind.

6. Ausblick auf weitere Arbeiten

Nach Abschluß der Studienarbeit stellt sich nun natürlich die Frage, was soll mit den entstandenen Dokumenten und dem Programmcode passieren, und was fehlt noch, damit mit dem Programm wirklich gearbeitet werden kann.

Daher beschäftigen wir uns in den beiden folgenden Unterkapiteln nun mit den „offenen Enden“ und den nächsten Schritten, welche, falls diese Projekt in einer oder mehreren Diplom-Arbeiten fortgeführt werden sollte, zu unternehmen wären.

6.1. „Offene Enden“

Um den Rahmen einer Studienarbeit nicht zu sprengen, ist es uns bis zum jetzigen Zeitpunkt leider nicht mehr möglich gewesen, folgende Punkte zu realisieren:

1. Als erstes fehlt bisher die Möglichkeit, eingefügte Objekte zu selektieren und nachträglich bearbeiten zu können. Die im Moment realisierte Anzeige von rudimentären Karten soll lediglich eine der programmierten Funktionen veranschaulichen und ist keinesfalls als vollständig anzusehen.

2. Außerdem wäre die Interaktionskomponente zu überarbeiten und zu erweitern und die bereits ansatzweise vorhandene MindMap Funktionalität in die Oberfläche zu integrieren.

3. Des weiteren müßte der Server entsprechend Kapitel 6.2 Punkt 4 + 5 überarbeitet werden.

4. Dementsprechend wäre ein VersionsTool zu überlegen und zu implementieren. Dies müßte sowohl auf der Server als auch auf der Client Seite erfolgen.

5. Dann könnte noch eine Erweiterung der implementierten Metaplan-Methoden erfolgen, und zwar beispielsweise um die Methoden der Einpunktabfrage, Mehrpunktabfrage und der anonymen Kartenabfrage.

6. Abschließend könnte man noch Funktionalität hinsichtlich einer offline Vorbereitung von Moderationen implementieren.

6.2. Nächste Schritte

Als nächste mögliche Schritte, um dieses Projekt in einer Diplom-Arbeit weiter zu führen, stellen wir uns folgende Vorgehensweise vor:

1. Die MindMap Funktionalitäten der Basisklassen in die Oberfläche integrieren.

2. Die Oberfläche weiter überarbeiten und fine-tuning daran vorzunehmen (optimalere Abstimmung der einzelnen Komponenten aufeinander).

3. Eine Versionen-Verwaltung implementieren, um verschiedene Stadien einer Moderation festzuhalten.

4. Den Server so überarbeiten, daß eine effizientere Übermittlung und Protokollierung des Ist-Zustandes der Moderation erfolgt und der Server gezielte Anfragen dazu beantworten kann.

5. Entsprechend den Punkten 3 + 4 für Persistenz der in der Moderation anfallenden Daten zu sorgen und dem Benutzer die Möglichkeit zu geben, die Daten auch lokal zwischen-speichern zu können.

6. Das ChatTool als eigenständiges Tool (Klasse) implementieren und nicht in die Interaktionskomponente einzubinden wie bisher.

Für die Zukunft wäre noch darüber nachzudenken, was für alternative Nutzungs-/Weiterentwicklungsmöglichkeiten und andere Moderationsformen (Vortrag, Ideenschmiede, Ablaufplanung, Projektplaner...) evtl. noch durch unser Tool zu unterstützen möglich und sinnvoll wäre.

Dabei fallen uns z.B. Anwendungen wie Internet Colaboration & Conferencing mit Audio- und Videoeinbindung ein.

7. Literatur

[Breidenbach 1997] Jeff Breidenbach: *Free Internet Conferencing Tools*. 1997. http://ac.mit.edu/company.

[Buzan 1993] T. Buzan: *Kopftraining*. München: Goldmann 1993.

[Dauscher 1998] U. Dauscher: *Moderationsmethode und Zukunftswerkstatt*. Neuwied, Kriftel, Berlin: Luchterhand, 2.,verb. Aufl. 1998.

[DecisionExplorer 1998] Decision Explorer™ 3.0.7. Banxia Software. Glasgow. http://www.banxia.com/demain.html.

[Flanagan 1998] David Flanagan: *Java Examples in a Nutshell: Der Beispielband zu Java in a Nutshell*. Köln: O'Reilly 1998.

[Gamma et al. 1996] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Entwurfsmuster - Elemente wiederverwenbarer objektorientierter Software*. Übersetzung von D. Riehle, Bonn: Addison Wesley, 1996.

[Inspiration 1997] Inspiration 5.0c. Innotech. Schönach. http://www.inspiration.com.

[Jungbluth 1998A] Volker Jungbluth: *Denktechniken. Der Weg zum genialen Einfall*. c't 20/98. S.136-138.

[Jungbluth 1998B] Volker Jungbluth: *Ideenmaschienen, Kreativitätswerkzeuge im Vergleich*. c't 20/98. S.142-147.

[MindManger 1998] MindManager™ 3.5-1a. Visualizing Ideas. Starnberg. http://www.mindmanager.de.

[Reimann 1998] Prof. Peter Reimann: *Frische Brise. Helfen Mind Maps beim Denken?* c't 20/98. S.140-141.

[Schnelle-Cölln & Schnelle 1997] T. Schnelle-Cölln, E. Schnelle: *Visualisieren in der Moderation: Eine praktische Anleitung für Gruppenarbeit und Präsentation.* Hamburg: Windmühle, Verl. Und Vertrieb von Medien, 1997 (Moderation in der Praxis; Bd. 5).

[VisiMap 1998] VisiMap. Team success AG. Selent. http://www.coco.co.uk.

[Züllighoven 98] H. Züllighoven: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- & Material-Ansatz*. Heidelberg: dpunkt, 1998.

Anhang A: Installationshinweise

Das Softwaresystem zur computerunterstützten Moderation ***COMO*** teilt sich in zwei Programmteile auf.

Einmal in den Server als eigenständiges Programm und in den Clienten als Applet.

Systemvoraussetzungen auf Anbieterseite einer Moderation:

Um den Server benutzen zu können, benötigt man einen als Webserver eingerichteten Rechner, auf dem ein Java Interpreter der Version JDK/JRE 1.1.6 oder neuer läuft. Auf der beigefügten CD befindet sich in einem entsprechenden Unterverzeichnis ein Archive mit einem JRE 1.1.x für X86-Win32.

Das Applet selbst muß mit einer entsprechend eingerichteten Webseite auf einem Webserver liegen. Dabei muß es sich um den selben Rechner handeln, auf dem auch das dazugehörige Server Programm läuft.

Systemvoraussetzungen auf Teilnehemerseite:

Ein Teilnehmer der Moderation braucht lediglich einen Inter-/Intranetzugang und einen Java fähigen Webbrowser der neuesten Generation. Hierfür bietet sich der Netscape Communicator 4.61 oder der MS Internet Explorer 5 an. Dabei ist jedoch zu beachten, daß für den IE5 zwingend das Java ActiveX-Plugin von SUN benötigt wird. Auf der CD befindet sich ebenfalls in einem entsprechenden Verzeichnis ein Archive mit dem Java Plugin 1.1.x für X86-Win32.

Die 100%ige Funktionalität des Applets wurde dabei bisher nur für X86-Win32 basierende Systeme untersucht.

Kleinere Aktualisierungsprobleme der Grafik treten leider in den Eingabeboxen des Java Plugins, bedingt durch die 1.1.x Swing Implementation, auf.

Dies läßt sich aber durch den Einsatz des Netscape Comunicator 4.61 ohne Plugin umgehen. Leider dauert dabei jedoch das Starten des Applet erheblich länger.

Installation Anbieterseite:

Das Serverprogramm befindet sich auf der CD im Unterverzeichnis
`D:\Server_Java_x\presentationTool\`
Wobei D durch den entsprechenden CD-Laufwerksbuchstaben zu ersetzen ist.

Auf dem Rechner, auf dem der Server installiert werden soll, ist zuerst ein Verzeichnis mit dem Namen presentationTool zu erstellen. In dieses Verzeichnis müssen dann alle Dateien aus dem Verzeichnis `d:\Server_Java_x\presentationTool\` auf der CD kopiert werden.

Der Server wird aus dem Oberverzeichnis von ..`\presentationTool` gestartet.

Die dazugehörige Befehlszeile lautet:

```
java presentationTool/presentationToolServer [-v] [PortNo]
```

Die zusätzlichen Parameter `–v` und `PortNo` sind dabei optional.

`-v` bedeutet dabei, daß alle Aktivitäten in der Shell mit protokolliert werden.

`PortNo` ist die Nummer des Ports, auf dem der Server neue Clients empfängt. Per Default ist der Port 8411 eingestellt.

!! Achtung !! sollte hier eine andere Port Nummer als die Default Einstellung gewählt werden, so muß in den Webseiten, mit denen das Applet gestartet wird, entsprechend der Parameter `PORT` im Applet-Tag angepaßt werden.

Für das Client Applet ist eine Webseite auf dem auf diesem Rechner laufenden Webserver einzurichten. Zwei Templates dafür befinden sich im Verzeichnis `d:\Client_Java_x\` .

Diese sind mit dem Unterverzeichnis `presentationTool\` in das Ziel-Verzeichnis und ein entsprechendes Unterverzeichnis auf dem Webserver zu kopieren. In diesem Unterverzeichnis befinden sich dabei alle Dateien, die für das Applet benötigt werden.

Bei den beiden Templates handelt es sich jeweils um eins für die in die Browser integrierten Javaumgebungen und eins für das Java Plugin.

In die Templates sollten noch kleine Einleitungstexte hinsichtlich einer geplanten Moderation durch die Moderationsleitung eingefügt werden.

Außerdem muß noch der Parameter Channel auf einen für die geplante Moderation eindeutigen Namen gesetzt werden.

Damit wären alle technischen Vorbereitungen von der Anbieterseite erledigt.

Anhang B: Fachliche Begriffe

Anstelle eines Glossars enthält diese Arbeit zwei Anhänge: Zunächst hier eine Auflistung und Kurzerklärung der fachlichen Begriffe, die sich im Wesentlichen aus den Szenarios in Kapitel 3.1 und 3.2 ergeben haben. Bei den Einträgen handelt es sich also vor allem um Begriffe aus den bestehenden, computerunabhängigen Moderationsmethoden. Im folgenden Anhang C folgt eine Auflistung der technischen Begriffe.

Aktiver Teilnehmer - s. Redner

Arbeitsfläche - Die Fläche auf der die einzelnen Methoden der Moderation angewendet werden. Dies bedeutet das z.B. Karten geklebt oder Mind Maps auf diese Fläche gemalt werden.

Ast - Ein Zweig, der von der Wurzel ausgeht

Fragetypen:

- Kartenabfrage - siehe Kapitel 3.1 Szenarios Metaplan
- Zuruffrage - siehe Kapitel 3.1 Szenarios Metaplan
- Einpunktfrage - siehe Kapitel 3.1 Szenarios Metaplan
- Mehrpunktfrage - siehe Kapitel 3.1 Szenarios Metaplan

Grafik - Ein Bild, das bei Mind Maps ergänzend zum Text eingesetzt werden sollte

Karte - Karten in verschiedenen Größen, Formen und Farben (MP).

Kartenabfrage:

- anonyme K. - siehe Kapitel 3.1 Szenarios Metaplan
- nicht anonyme K. - siehe Kapitel 3.1 Szenarios Metaplan
- K. in Kleingruppen - siehe Kapitel 3.1 Szenarios Metaplan

Knoten - Ein Bild mit Text (oder eines von beiden allein) auf der Arbeitsfläche repräsentiert in der Mind Map einen Knoten. Ein Knoten ist immer über andere Knoten und Linien mit der Wurzel verbunden. Knoten werden untereinander durch Linien verbunden.

Moderations Materialien:

- Arbeitsfläche / Pinwand (MP & MM)
- Grafiken (MM)
- Karten (MP)
- Linien (MP)
- Tabellen (MP)
- Rednerliste (MP & MM)
- Verbindungen, d.h. Äste & Zweige (MM)
- Wolke (MP)

Metaplan - Sammlung von Methoden und Werkzeugen, um eine Moderation effektiv zu führen.

Mind Map - Hierarchische Struktur aus Knoten, Ästen und Zweigen

MM - siehe Mind Map

Moderator - Leiter einer Moderation und Kenner der verschiedenen Methoden (MP u. MM).

Moderation	- Anwendungssituation der verschiedenen Methoden (MP u. MM) in einer Gruppe.
Moderationsformen	- Mind Map oder Metaplan
MP	- siehe Metaplan
Redner	- Teilnehmer einer Moderation, dem das Wort erteilt worden ist.
Rednerliste	- In der R. werden die nächsten Redner aufgelistet und durch den Moderator verwaltet.
Pinwand	- Tafel, auf die mit Nadeln Moderations Materialien geheftet werden. s. a. Arbeitsfläche
Teilnehmer	- Jemand, der an einer Moderation teilnimmt.
Verbindungen	- Oberbegriff für Äste und Zweige (MM)
Wolke	- Normalerweise die Form der themenbeschreibenden ersten Karte bei der Metaplan-Moderation. Wird auch häufig für die Wurzel in Mind Maps benutzt.
Wurzel	- Der erste Knoten einer Mind Map
Zweig	- Eine Linie, die zwei Knoten verbindet

Anhang C: Technische Begriffe

Wie im vorherigen Anhang B erläutert, finden sich hier die technischen Begriffe der Arbeit zusammen mit ihrem Kurzerklärungen. Aufgelistet sind also vor allem Begriffe, die aus der technischen Umsetzung in ein computerbasiertes Werkzeug entstanden sind.

Anmeldewerkzeug - Das A. leitet Moderator und Teilnehmer nach dem Start des Applets durch die nötigen Bildschirmmasken.

Arbeitsumgebung - Die Arbeitsumgebung besteht aus der Arbeitsfläche und, darum herum angeordnet, den Werkzeugen Chat, Rednerlistenverwaltung, Zeichenwerkzeug, Formatierwerkzeug und je nach Anwendung Grafikgallerie (MM) oder Kartengallerie (MP)

Chat - Kommunikationseinrichtung, über die ein Teilnehmer Nachrichten an alle anderen Teilnehmer schicken kann.

Grafikgallerie - Werkzeug zum Auswählen einer Grafik (MM)

Kartengallerie - Werkzeug zum Auswählen eines Kartentyps (MP)

Rednerlistenverwaltung - Das Werkzeug unseres Systems, das die nötigen Funktionen einer Verwaltung der Teilnehmer von Moderationen zur Verfügung stellt.

Allgemeine Funktionen:

Eintragen - Hiermit kann ein Teilnehmer seinen Namen an das Ende der Rednerliste hängen.

Redner Funktionen:

Nächster - Ein Redner kann hiermit das Recht die Arbeitsfläche zu bearbeiten an den als nächstes auf der Rednerliste eingetragenen Teilnehmer übergeben.

Moderator Funktionen:

Freigeben - Nach dem Freigeben der Rednerliste können sich Teilnehmer in diese eintragen.

Unterbrechen - Durch U. wird der Redner seines Rechtes die Arbeitsfläche zu bearbeiten beraubt, und der Moderator wird zum Redner.

Löschen - Damit wird die Rednerliste gelöscht.

Schließen - Nach dem Schließen der Rednerliste können sich keine Teilnehmer mehr in diese eintragen.

Texteditor - Mit dem T. können die Texte für neue Knoten / Karten eingegeben und für bereits vorhandene editiert werden.

Versionskontrolle - Über die V. kann der Moderator die momentane Arbeitsfläche speichern und zuvor gespeicherte laden. Darüberhinaus bietet die V. eine Verwaltungshilfe für verschiedene Versionen (Stadien) einer Moderation und der damit verbundenen Arbeitsfläche.

Versionen-Verwaltung - s Versionskontrolle

Werkzeuge - Komponenten, mit denen man das Programmverhalten oder den Status der Moderation aktiv beeinflussen kann. Dazu gehören:

- Anmeldewerkzeug
- Chat
- Grafikgallerie
- Kartengallerie
- Rednerlistenverwaltung
- Texteditor
- Versionskontrolle
- Zeichenwerkzeug
- Zoom

Zeichenwerkzeug - Tabellen, Punkte, Linien, Blitze, Tabellen (MP-spezifisch)

Zoom - Mit diesem Werkzeug läßt sich zwischen Detail-Ansicht und Übersicht auf die Arbeitsfläche wechseln.

Anhang D: Abbildungsverzeichnis

Anhang E: Quellcode Dokumentation

Dieser Anhang enthält die Kommentare des Programmcodes, die im Javadoc-Format (Kommentare im Java-code zwischen den Begrenzern `/**` und `*/`) angegeben sind.

Package como

Class Index

Class como.Board

```
java.lang.Object
   |
   +----como.Board
```

---

public class **Board**
extends Object
implements Serializable Die Klasse `Board` stellt die Datenstruktur (also das Material "Arbeitsfläche") der Arbeitsfläche dar.
Die Objekte (Instanzen der Klasse `PresentationObject`), die sich auf der Arbeitsfläche positionieren lassen, erden mit ihrer Position in der Klasse `Board` gespeichert.

See Also:

Message, PresentationObject, BoardTool

Variables

_board

```
private Hashtable _board
```

_iDs

```
private Hashtable _iDs
```

_boardSize

```
private Dimension _boardSize
```

_boardName

```
private String _boardName
```

_boardColor

```
private Color _boardColor
```

_boardType

```
private PresentationType _boardType
```

_creator

```
private Person _creator
```

Constructors

Board

```
public Board(int width,
             int height,
             String boardName,
             PresentationType presType,
             Person creator)
```

`Constructor` der Klasse Board. Das Board hat neben der fachlichen Bedeutung (Metaplan/MindMap) die Funktion eines Behälters für die `PresentationObjects`.

Parameters:

width - die Breite in Pixel als `int`.
heigth - die Höhe in Pixel als `int`.
boardName - der in der Presentation eindeutige Name des Boards als `String`.
presType - der Typ der Presentation (MP/MM) als `PresentationType`.

creator - der Erzeuger des Boards (meist. Moderator der Presentation). vom Typ `Person`.

Methods

addPresentationObject

```
public void addPresentationObject(PresentationObject presObj)
```

Das angegebene Objekt wird der Datenstruktur `Board` hinzugefügt. Dabei wird als Einfügeposition standardmä&suml;ig die Mitte der Arbeitsfläche benutzt.

Parameters:

presObj - das `PresentationObject` das hinzugefügt werden soll.

addPresentationObjectAt

```
public void addPresentationObjectAt(PresentationObject presObj,
                                    Point location)
```

Das angegebene Objekt wird mit der angegebenen Position der Datenstruktur `Board` hinzugefügt.

Parameters:

presObj - das `PresentationObject` das hinzugefügt werden soll.
location - die Position des neuen Objektes.

boardIsEmpty

```
public boolean boardIsEmpty()
```

clear

```
public void clear()
```

Löscht alle Objekte aus der Datenstruktur Board.

contains

```
public boolean contains(int presObjID)
```

Returns:

true wenn das `PresentationObject` zu angegebenen ID in der Datenstruktur Board enthalten ist.

contains

```
public boolean contains(PresentationObject presObj)
```

Returns:

true wenn das angegebene `PresentationObject` in der Datenstruktur Board enthalten ist.

getAllPresentationObjects

```
public PresentationObject[] getAllPresentationObjects()
```

Returns:

Array von allen `PresentationObjects` die sich auf dem Board befinden.

getBoardColor

```
public Color getBoardColor()
```

Returns:

Hintergrundfarbe des Boards

getCreator

```
public Person getCreator()
```

Returns:

Die Person, die das Board erschaffen hat.

getName

```
public String getName()
```

Returns:

Den Namen des Boards

getPositionOfPresentationObject

```
public Point getPositionOfPresentationObject(int presObjID)
```

Parameters:

presObjID - die ID des `PresentationObjects` nach dessen Position gesucht wird.

Returns:

Die Position des angegebenen Objektes

getPositionOfPresentationObject

```
public Point getPositionOfPresentationObject(PresentationObject presObj)
```

Parameters:

presObj - das `PresentationObject` nach dessen Position gesucht wird.

Returns:

Die Position des angegebenen Objektes

getPresentationObject

```
public PresentationObject getPresentationObject(int presObjID)
```

Returns:

das `PresentationObject` zur angegebenen ID

getPresentationType

```
public PresentationType getPresentationType()
```

Returns:

Den Typ des Boards (bisher `METAPLAN` oder `MINDMAP`)

getSize

```
public Dimension getSize()
```

Returns:

Dimension des Boards als Dimensions-Objekt in Pixeln

movePresentationObjectTo

```
public void movePresentationObjectTo(int presObjID,
                                     Point location)
```

Das angegebene Objekt an die angegebene Position verschoben

Parameters:

presObjID - die ID des `PresentationObjects` das verschoben werden soll.
location - die neue Position des Objekts

movePresentationObjectTo

```
public void movePresentationObjectTo(PresentationObject presObj,
                                     Point location)
```

Das angegebene Objekt an die angegebene Position verschoben

Parameters:

presObj - das `PresentationObject` das verschoben werden soll.
location - die neue Position des Objekts

removePresentationObject

```
public void removePresentationObject(int presObjID)
```

Das angegebene Objekt wird aus der Datenstruktur Board gelöscht.

Parameters:

presObjID - die ID des `PresentationObjects` das gelöscht werden soll.

removePresentationObject

```
public void removePresentationObject(PresentationObject presObj)
```

Das angegebene Objekt wird aus der Datenstruktur Board gelöscht.

Parameters:

presObj - das `PresentationObject` das gelöscht werden soll.

setColor

```
public void setColor(int red,
                     int green,
                     int blue)
```

Methode zum setzen der Hintergrundfarbe des Boards. Diese Methode ist nur bei Pr&aumlsentationen des Typs `PresentationType.MINDMAP` zulässig. Der Aufruf wird bei Pr&aumlsentationen des Typs `PresentationType.METAPLAN` ignoriert.

Parameters:

red - Der Rotanteil der Farbe als `int`.

green - Der Grünanteil der Farbe als `int`.

blue - Der Blauanteil der Farbe als `int`.

setColor

```
public void setColor(Color color)
```

Methode zum setzen der Hintergrundfarbe des Boards. Diese Methode ist nur bei Pr&aumlsentationen des Typs `PresentationType.MINDMAP` zulässig. Der Aufruf wird bei Pr&aumlsentationen des Typs `PresentationType.METAPLAN` ignoriert.

Parameters:

color - Die Farbe als `java.awt.Color`.

setSize

```
protected void setSize(int width,
                       int height)
```

Setzt die Ausma&suml;e des Boards

Parameters:

width - Die Breite des Boards in Pixeln

height - Die Höhe des Boards in Pixeln

setSize

```
protected void setSize(Dimension size)
```

Setzt die Ausma&suml;e des Boards

Parameters:

size - Das Dimensions-Objekt zur Bestimmung der Ausma&suml;e in Pixeln

Class como.BoardMessageType

```
java.lang.Object
   |
   +----como.BoardMessageType
```

---

public class **BoardMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die das Board betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

_ALLTYPES

```
private static final String _ALLTYPES[]
```

Alle gültigen Message-Namen für diese Klasse

ADD_NEW_BOARD_TO_BOARDTOOL

```
public static final BoardMessageType ADD_NEW_BOARD_TO_BOARDTOOL
```

Nachricht, die veranlaßt, eine neue Arbeitsfläche Arbeitsumgebung hinzuzufügen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das hinzuzufügende `Board`-Objekt.

ADD_NEW_PRES_OBJ_TO_BOARD

```
public static final BoardMessageType ADD_NEW_PRES_OBJ_TO_BOARD
```

Nachricht, die veranlaßt, ein neues Objekt zur Arbeitsfläche hinzuzufügen
Bei dieser Nachricht werden zwei Datenobjekte erwartet:
1. das hinzuzufügende `PresentationObject`-Objekt.
2. die Position an der es hinzugefügt werden soll.

MOVE_PRES_OBJ_ON_BOARD

```
public static final BoardMessageType MOVE_PRES_OBJ_ON_BOARD
```

Nachricht, die veranlaßt, ein Objekt auf der Arbeitsfläche zu verschieben
Bei dieser Nachricht werden zwei Datenobjekte erwartet:
1. das hinzuzufügende `PresentationObject`-Objekt.
2. die neue Position des Objekts.

REMOVE_PRES_OBJ_FROM_BOARD

```
public static final BoardMessageType REMOVE_PRES_OBJ_FROM_BOARD
```

Nachricht, die veranlaßt, ein Objekt von der Arbeitsfläche zu löschen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das zu entfernende `PresentationObject`-Objekt.

CLEAR_AND_SET_ALL_BOARDS

```
public static final BoardMessageType CLEAR_AND_SET_ALL_BOARDS
```

Nachricht, die vom VersionTool bei einer neuen Moderation verschickt wird und veranlaßt, daß alle Arbeitsflächen gelöscht werden und eine neue angelegt wird.
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das neue `Board`-Objekt.

NEW_BOARD_SELECTED

```
public static final BoardMessageType NEW_BOARD_SELECTED
```
Nachricht, die veranlaßt, eine andere Arbeitsfläche als aktuelle auszuwählen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das neue `Board`-Objekt.

NEW_DEF_BOARD_NAME_NO

```
public static final BoardMessageType NEW_DEF_BOARD_NAME_NO
```
Nachricht, die veranlaßt, eine neue Nummer der Arbeitsfläche zu kreieren
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar die neue Nummer als `int`

_action

```
private String _action
```
Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, BoardMessageType...

Constructors

BoardMessageType

```
public BoardMessageType(String action)
```

Methods

equals

```
public boolean equals(BoardMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück

See Also:
_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.BoardTool

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----como.BoardTool
```

---

public class **BoardTool**
extends Observable
implements Observer Die Klasse BoardTool bildet die Funktionskomponente (FK) der Arbeitsfläche
Änderungen, die den Client über das `NetworkTool` erreichen und die Arbeistfl&aauml;che betreffen, werden in der Methode `update()` abgefangen und an die IAK (hier die Klasse `BoardUI` über den gleichen Mechanismus mit Hilfe der Methode `notifyObservers()` weitergegeben
See Also:
Observer, Observable

Variables

_defaultName

```
protected static final String _defaultName
```

isInitiated

```
public boolean isInitiated
```

_currentBoard

```
private Board _currentBoard
```

_selectedPresObj

```
private PresentationObject _selectedPresObj
```

_allBoards

```
private Hashtable _allBoards
```

_myLocalUser

```
private Person _myLocalUser
```

_myNT

```
private NetworkTool _myNT
```

_myPOT

```
private PresentationObjectTool _myPOT
```

_defBoardNameNo

```
private int _defBoardNameNo
```

Constructors

BoardTool

```
public BoardTool(NetworkTool nT,
                 PresentationObjectTool pOT,
                 Person myPerson)
```

Erstelle ein neues Werkzeug-Objekt der Klasse `BoardTool`
Parameters:
fkN - referenz auf die lokale Netzwerkkomponente vom Typ `presentationTool.NetworkTool`.
myPerson - der lokale Benutzer des BoardTools vom Typ `presentationtool.Person`.
See Also:
NetworkTool, Person

Methods

add

```
private void add(Board newBoard)
```

Fügt ein Board zum BoardTool hinzu. Wird von update aufgerufen mit den Daten der Netzwerkkomponente.

addNewPresObjToCurBoardAt

```
private void addNewPresObjToCurBoardAt(PresentationObject presObj,
                                       Point location)
```

Diese Methode wird aufgerufen, wenn über die Methode update()vom NetworkTool eine Nachricht eintrifft, die besagt, daß ein neues Objekt auf der Arbeitsfläche erstellt werden soll.
Parameters:
presObj - Das neue Objekt
location - Die Position an der das Objekt der Arbeitsfläche hinzugefügt werden soll.

checkConnectionsWith

```
private void checkConnectionsWith(PresentationObject presObj)
```

entfernt die Connections deren Start- oder Endpunkt das PresentationObject war.

clearAndSetABN

```
private void clearAndSetABN(Hashtable allNewBoards)
```

clearAndSetAllBoardsNew

```
protected void clearAndSetAllBoardsNew(Hashtable allNewBoards)
```

Wird vom VersionTool benutzt wenn eine neue Präsentation geladen wird.

contains

```
public boolean contains(String boardName)
```

überprüft ob ein Board mit Namen `boardName` vorhanden ist.
Parameters:
boardName - ein String mit dem Namen des Boardes auf das geprüft werden soll.
Returns:
boolean

createBoard

```
public void createBoard(int width,
                        int height,
                        PresentationType presType)
```

es wird ein neues Board mit einem Defaultnamen der Präsentation hinzugefügt. Später sollte ein eindeutiger Name generiert werden. Bisher immer der selbe, statische Name!
Parameters:
width - die Breite des Boards in Pixeln als `int`.
height - die Höhe des Boards in Pixeln als `int`.
presType - der Präsentationstyp des Boards innerhalb der Präsentation (Metaplan/MindMap) vom Typ `presentationTool.PresentationTyp`.

createBoard

```
public void createBoard(int width,
                        int height,
                        String boardName,
                        PresentationType presType)
```

es wird ein neues Board der Präsentation hinzugefügt

Parameters:

width - die Breite des Boards in Pixeln als `int`.
height - die Höhe des Boards in Pixeln als `int`.
boardName - ein eindeutiger Name des Boards innerhalb einer Präsentation.
presType - der Präsentationstyp des Boards innerhalb der Präsentation (Metaplan/MindMap) vom Typ `presentationTool.PresentationTyp`.

getAllBoards

```
protected Hashtable getAllBoards()
```

getCurrentBoardColor

```
public Color getCurrentBoardColor()
```

getCurrentBoardCreator

```
public Person getCurrentBoardCreator()
```

getCurrentBoardName

```
public String getCurrentBoardName()
```

getCurrentBoardSize

```
public Dimension getCurrentBoardSize()
```

getCurrentBoardsPresentationObjects

```
public PresentationObject[] getCurrentBoardsPresentationObjects()
```

getCurrentBoardType

```
public PresentationType getCurrentBoardType()
```

getPositionOfPresentationObject

```
public Point getPositionOfPresentationObject(int presObjID)
```

getPositionOfPresentationObject

```
public Point getPositionOfPresentationObject(PresentationObject presObj)
```

gibt die Position des übergebenen PresentationObjects als `java.awt.Point` zurück.

Returns:

liefert wenn vorhanden die Position als Point oder sonst null

getSelectedPresentationObject

```
public PresentationObject getSelectedPresentationObject()
```

hasCurrentBoard

```
public boolean hasCurrentBoard()
```

movePresObjTo

```
private void movePresObjTo(PresentationObject presObj,
                           Point location)
```

Diese Methode wird aufgerufen, wenn über die Methode update() vom NetworkTool eine Nachricht eintrifft, die besagt, daß ein Objekt auf der Arbeitsfläche verschoben werden soll.

Parameters:

presObj - Das Objekt
location - Die Position an die das Objekt der Arbeitsfläche verschoben werden soll.

removePresentationObjectFromCurrentBoard

```
public void removePresentationObjectFromCurrentBoard(int presObjID)
```
Diese Methode wird aufgerufen, wenn über die Methode update() vom NetworkTool eine Nachricht eintrifft, die besagt, daß ein Objekt von der Arbeitsfläche entfernt werden soll.

Parameters:

presObj - Die Identifikationsnummer des Objekts

remPresObjFromCurBoard

```
private void remPresObjFromCurBoard(PresentationObject presObj)
```
Diese Methode wird aufgerufen, wenn über die Methode update() vom NetworkTool eine Nachricht eintrifft, die besagt, daß ein Objekt von der Arbeitsfläche entfernt werden soll.

Parameters:

presObj - Das Objekt

selCurBoard

```
private void selCurBoard(String boardName)
```

selectCurrentBoard

```
public boolean selectCurrentBoard(String boardName)
```

Returns:

true wenn das Board mit dem Namen `boardName` zum aktuellen Board gemacht werden kann

selectPresentationObject

```
public void selectPresentationObject(int presObjID)
```

selectPresentationObject

```
public void selectPresentationObject(PresentationObject presObj)
```

sendAddNewCardMessage

```
protected void sendAddNewCardMessage(Point location,
                                     CardShape cardShape,
                                     Color color,
                                     Color textColor)
```
Fügt dem derzeit ausgewählten Board eine Karte an der bestimmten Position `location` hinzu. Hierzu wird eine Nachricht übers Netz geschickt.

Parameters:

location - die Position an der das PresentationObject hinzugefügt werden soll.

sendMovePresObjToMessage

```
public void sendMovePresObjToMessage(int presObjID,
                                     Point location)
```
Verschiebt das angegebene Objekt im derzeit ausgewählten Board an die angegebene Position. Hierzu wird eine Nachricht übers Netz geschickt.

Parameters:

preObjID - die Identifikationsnummer des zu verschiebenden Objekts.
location - die Position an der das PresentationObject hinzugefügt werden soll.

sendRemovePresObjFromCurrentBoardMessage

```
public void sendRemovePresObjFromCurrentBoardMessage(PresentationObject
presObj)
```
Entfernt das angegebene Objekt vom derzeit ausgewählten Board. Hierzu wird eine Nachricht übers Netz geschickt.

Parameters:

location - die Position an der das PresentationObject hinzugefügt werden soll.

setCurrentBoardColor

```
public void setCurrentBoardColor(Color color)
```

showAllBoardNames

```
public String[] showAllBoardNames()
```

Hilfs-Methode für das Speicherwerkzeug um möglichst bequem alle Boards zu bekommen.

update

```
public void update(Observable obs,
                   Object arg)
```

Wird vom Observable(Netzwerkkomponente) aufgerufen sobald sich an ihr etwas ändert.

Class como.BoardToolMessageType

```
java.lang.Object
   |
   +----como.BoardToolMessageType
```

---

public class **BoardToolMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die das BoardTool betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

LIST_OF_BOARDS_CHANGED

```
public static final BoardToolMessageType LIST_OF_BOARDS_CHANGED
```

Nachricht, die veranlaßt, die Liste der Boards der IAK zu aktualisieren.
Bei dieser Nachricht werden keine Datenobjekte erwartet.

CURRENT_BOARD_CONTENT_CHANGED

```
public static final BoardToolMessageType CURRENT_BOARD_CONTENT_CHANGED
```

Nachricht, die veranlaßt, die Arbeitsfläche mit den aktuellen Daten neu zu zeichnen.
Noch nicht implementiert.

CURRENT_BOARD_CHANGED

```
public static final BoardToolMessageType CURRENT_BOARD_CHANGED
```

Nachricht, die veranlaßt, eine neue Arbeitsfläche auszuwählen.
Noch nicht implementiert.

ERROR_MESSAGE

```
public static final BoardToolMessageType ERROR_MESSAGE
```

Nachricht, die im ChatTeil der IAK angezeigt werden soll!
Bei dieser Nachricht wird der String der angezeigt werden soll als Datenobjekt erwartet.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, BoardMessageType...

Constructors

BoardToolMessageType

```
public BoardToolMessageType(String action)
```

Methods

equals

```
public boolean equals(BoardToolMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.BoardUI

`como.BoardUI`

public class **BoardUI**
implements Observer Diese Klasse bildet die Interaktionskomponente (IAK) der Arbeitsfläche. Die Arbeitsfläche ist Teil der Arbeitsumgebung, deren IAK in der Klasse `PresentationTool` implementiert ist.
Benutzereingaben werden abgefangen und an das `BoardTool` weitergeleitet.
See Also:
BoardTool, presentationTool

Variables

colors

```
private final Color colors[]
```

color_n

```
private final int color_n
```

textColor

```
public Color textColor
```

backgroundColor

```
public Color backgroundColor
```

drawingArea

```
JPanel drawingArea
```

_boardTool

```
private BoardTool _boardTool
```

_myLocalUser

```
private Person _myLocalUser
```

size

```
private Dimension size
```

objects

```
private Vector objects
```

Constructors

BoardUI

```
public BoardUI(Person localUser)
```

Konstruiert eine neue Arbeitsfläche

Methods

setBackground

```
public void setBackground(Color color)
```

Setzt die Hintergrundfarbe der Arbeitfläche

Parameters:
color - DIe neue Farbe

setBoardTool

```
protected void setBoardTool(BoardTool bT)
```

update

```
public void update(Observable obs,
                   Object arg)
```

This method is called whenever the observed object (BoardTool) is changed. An application calls an Observable object's notifyObservers method to have all the object's observers notified of the change.

Class como.Card

```
java.lang.Object
   |
   +----como.PresentationObject
           |
           +----como.Card
```

---

public class **Card**
extends PresentationObject
implements Serializable Diese Klasse implementiert ein Material für Objekte, die auf der Arbeitsfläche benutzt werden sollen (Instanzen von Subklassen der abstrakten Klasse `PresentationObject`).
Bei diesem `PresentationObject` handelt es sich fachlich gesehen um eine Metaplan-Karte.
Zur Bestimmung der Form der Karte, steht die Klasse `CardShape` zur Verfügung.
Zur Manipulation dieser Materialien steht das Werkzeug `PresentationObjectTool` zur Verfügung. Die Positionierung auf der Arbeitsfläche regelt das Werkzeug `BoardTool`
See Also:
PresentationObjectTool, PresentationObject, BoardTool, MyText, CardShape

---

Variables

_shape

```
private CardShape _shape
```

_cardText

```
private MyText _cardText
```

Constructors

Card

```
public Card(Card card2)
```

Erstelle eine neue Karte auf der Basis der angegebenen Karte
Parameters:
card2 - die zu kopierende Karte

Card

```
public Card(Color cardColor,
            Person creator,
            CardShape shape,
            int objID)
```

Erstelle eine neue Karte
Parameters:
cardColor - Farbe der Karte
creator - `Person`-Objekt des Teilnehmers, der die Karte erstellt
shape - Form der Karte (`CardShape`)
objID - Eine eindeutige Identifikationsnummer der Karte.
See Also:
CardShape

Card

```
public Card(Color cardColor,
            Person creator,
            CardShape shape,
            MyText cardText,
            int objID)
```

Erstelle eine neue Karte

Parameters:

cardColor - Farbe der Karte

creator - `Person`-Objekt des Teilnehmers, der die Karte erstellt

shape - Form der Karte (`CardShape`)

cardText - Der Text, der auf der Karte erscheinen soll.

objID - Eine eindeutige Identifikationsnummer der Karte.

See Also:

CardShape

Methods

equals

```
public boolean equals(Card card2)
```

getShape

```
public CardShape getShape()
```

getText

```
public MyText getText()
```

setShape

```
private void setShape(CardShape shape)
```

setText

```
public void setText(MyText text)
```

Class como.CardShape

```
java.lang.Object
   |
   +----como.CardShape
```

---

public class **CardShape**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Formen der Karten, die auf der Arbeitsfläche dargestellt werden können, zu kapseln.
See Also:
createCard, Card

Variables

_ALLSHAPES

```
private static final String _ALLSHAPES[]
```

Alle Möglichen Formen einer Karte

CLOUD

```
public static final CardShape CLOUD
```

Kartenform: Wolke

RECTANGLE

```
public static final CardShape RECTANGLE
```

Kartenform: Rechteck

CIRCLE

```
public static final CardShape CIRCLE
```

Kartenform: Kreis

SMALL_CIRCLE

```
public static final CardShape SMALL_CIRCLE
```

Kartenform: kleiner Kreis

LARGE_CIRCLE

```
public static final CardShape LARGE_CIRCLE
```

Kartenform: großer Kreis

OVAL

```
public static final CardShape OVAL
```

Kartenform: Oval

STRIPE

```
public static final CardShape STRIPE
```

Kartenform: langes, schmales Rechteck

_shape

```
private String _shape
```

Constructors

CardShape

```
public CardShape(String shape)
```

Methods

equals

```
public boolean equals(CardShape shape2)
```

getCardShape

```
public String getCardShape()
```

Returns:
Gibt die Form der Karte zurück
See Also:
_ALLSHAPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.ChatMessageType

```
java.lang.Object
   |
   +----como.ChatMessageType
```

---

public class **ChatMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die den Chat betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

WELCOME_MESSAGE

```
public static final ChatMessageType WELCOME_MESSAGE
```

Eine Nachricht, die an neue Teilnehmer verschickt wird.
Bei dieser Nachricht wird eine Datenobjekt erwartet und zwar der Willkommenstext.

NAME_ENTERED

```
public static final ChatMessageType NAME_ENTERED
```

Note: NAME_ENTERED is deprecated. *Jetzt werden stattdessen die Anmeldedialoge benutzt!*
Nachricht, die meldet, daß der Name im Chat eingegeben und abgeschickt wurde.
Bei dieser Nachricht wird eine Datenobjekt erwartet und zwar der eingegebene Namen.

TEXT_ENTERED

```
public static final ChatMessageType TEXT_ENTERED
```

Nachricht, die meldet, daß ein Text im Chat eingegeben und abgeschickt wurde.
Bei dieser Nachricht wird eine Datenobjekt erwartet und zwar der eingegebene Text.

ANNOUNCEMENT

```
public static final ChatMessageType ANNOUNCEMENT
```

Eine Nachricht, die generiert wird und Statusmeldungen (wie beispielsweise, daß jemand neues zur Moderation hinzugekommen ist) meldet.
Bei dieser Nachricht wird eine Datenobjekt erwartet und zwar die Meldung Text.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, BoardMessageType...

Constructors

ChatMessageType

```
public ChatMessageType(String action)
```

Methods

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.ClientMessageType

```
java.lang.Object
   |
   +----como.ClientMessageType
```

---

public class **ClientMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die den Client betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

GET_ALL_BOARDS

```
public static final ClientMessageType GET_ALL_BOARDS
```

GET_LIST_OF_PERSONS

```
public static final ClientMessageType GET_LIST_OF_PERSONS
```

GET_LIST_OF_SPEAKERS

```
public static final ClientMessageType GET_LIST_OF_SPEAKERS
```

GET_NEXT_PRESOBJ_NO

```
public static final ClientMessageType GET_NEXT_PRESOBJ_NO
```

USER_LEAVES_SESSION

```
public static final ClientMessageType USER_LEAVES_SESSION
```

Nachricht, die meldet, daß ein Teilnehmer die Moderation verlassen hat.
Bei dieser Nachricht wird eine Datenobjekt erwartet und zwar der verlassende Benutzer als `Person`-Objekt.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, ListOfPersonsMessageType...

Constructors

ClientMessageType

```
public ClientMessageType(String action)
```

Methods

equals

```
public boolean equals(ClientMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:

Gibt den Typ der Nachricht zurück

See Also:

_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:

toString in class Object

Class como.Connection

```
java.lang.Object
   |
   +----como.PresentationObject
           |
           +----como.Line
                   |
                   +----como.Connection
```

---

public class **Connection**
extends Line
implements Serializable Diese Klasse implementiert ein Material für Objekte, die auf der Arbeitsfläche benutzt werden sollen (Instanzen von Subklassen der abstrakten Klasse `PresentationObject`).
Bei diesem `PresentationObject` handelt es sich um eine Linie, die zwei andere Elemente (z.B. `Graphic`-Objekte) verbindet.
Zur Manipulation dieser Materialien steht das Werkzeug `PresentationObjectTool` zur Verfügung. Die Positionierung auf der Arbeitsfläche regelt das Werkzeug `BoardTool`.
See Also:
PresentationObjectTool, PresentationObject, BoardTool

Variables

_startPresObjectID

```
private int _startPresObjectID
```

_endPresObjectID

```
private int _endPresObjectID
```

Constructors

Connection

```
public Connection(Connection connect2)
```

Erstelle eine neue Verbindung auf der Basis der angegebenen Verbindung
Parameters:
connect2 - die zu kopierende Verbindung

Connection

```
public Connection(Color connectColor,
                  Person creator,
                  int startPresObjID,
                  int endPresObjID,
                  int objID)
```

Erstelle eine neue Verbindung zwischen zwei Knoten (`PresentationObjects`)
Parameters:
connectColor - Farbe der Verbindungslinie
creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt
startPresObjID - Identifikationsnummer des ersten Objekts
endPresObjID - Identifikationsnummer des zweiten Objekts
objID - Eine eindeutige Identifikationsnummer der Verbindung.

See Also:
Person

Methods

contains

```
public boolean contains(int presObjID)
```

Überprüft, ob das Objekt mit der angegebenen ID Start- oder Endobjekt der Verbindung ist.

Parameters:
presObjID - die zu suchende ID eine Objektes

Returns:
true, wenn entweder das Start- oder das Endobjekt die angegebene ID besitzt.

equals

```
public boolean equals(Connection connection2)
```

getEndID

```
public int getEndID()
```

getStartID

```
public int getStartID()
```

setEndID

```
protected void setEndID(int endPresObjID)
```

setStartID

```
protected void setStartID(int startPresObjID)
```

Class como.FreeText

```
java.lang.Object
   |
   +----como.PresentationObject
           |
           +----como.FreeText
```

---

public class **FreeText**
extends PresentationObject
implements Serializable Diese Klasse implementiert ein Material für Objekte, die auf der Arbeitsfläche benutzt werden sollen (Instanzen von Subklassen der abstrakten Klasse `PresentationObject`).
Bei diesem `PresentationObject` handelt es sich um einen Text, der frei auf der Arbeitsfläche plaziert werden kann.
Zur Manipulation dieser Materialien steht das Werkzeug `PresentationObjectTool` zur Verfügung. Die Positionierung auf der Arbeitsfläche regelt das Werkzeug `BoardTool`
See Also:
PresentationObjectTool, PresentationObject, BoardTool

Variables

_freeText

```
private MyText _freeText
```

Constructors

FreeText

```
public FreeText(FreeText freeText2)
```

Erstelle einen neuen freien Text auf der Basis des angegebenen freien Texts
Parameters:
freeText2 - der zu kopierende freie Text

FreeText

```
public FreeText(Color backColor,
                Person creator,
                int objID)
```

Erstelle einen neuen frei plazierbaren Text.
Parameters:
backColor - Hintergrundfarbe des Textes
creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt
objID - Eine eindeutige Identifikationsnummer der Verbindung.
See Also:
Person

FreeText

```
public FreeText(Color backColor,
                Person creator,
                MyText freeText,
                int objID)
```

Erstelle einen neuen frei plazierbaren Text.

Parameters:
backColor - Hintergrundfarbe des Textes
creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt
freeText - Der Text.
objID - Eine eindeutige Identifikationsnummer der Verbindung.
See Also:
Person

Methods

getText

```
public MyText getText()
```

setText

```
public void setText(MyText text)
```

Class como.Graphic

```
java.lang.Object
   |
   +----como.PresentationObject
          |
          +----como.Graphic
```

---

public class **Graphic**
extends PresentationObject
implements Serializable Diese Klasse implementiert ein Material für Objekte, die auf der Arbeitsfläche benutzt werden sollen (Instanzen von Subklassen der abstrakten Klasse `PresentationObject`).
Bei diesem `PresentationObject` handelt es sich um eine Grafik, die frei auf der Arbeitsfläche plaziert werden kann.
Zur Manipulation dieser Materialien steht das Werkzeug `PresentationObjectTool` zur Verfügung. Die Positionierung auf der Arbeitsfläche regelt das Werkzeug `BoardTool`
See Also:
PresentationObjectTool, PresentationObject, BoardTool

Variables

_imageName

```
private String _imageName
```

_graphicLabel

```
private String _graphicLabel
```

Constructors

Graphic

```
public Graphic(Graphic graphic2)
```

Erstelle eine neue Grafik auf der Basis der angegebenen Grafik
Parameters:
graphic2 - die zu kopierende Grafik

Graphic

```
public Graphic(Color backColor,
               Person creator,
               String imageName,
               int objID)
```

Erstelle eine neue Grafik.
Parameters:
backColor - Hintergrundfarbe des Textes
creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt
imageName - Der Dateiname der Grafik.
objID - Eine eindeutige Identifikationsnummer der Verbindung.
See Also:
Person

Graphic

```
public Graphic(Color backColor,
               Person creator,
               String imageName,
               String graphicLabel,
               int objID)
```

Erstelle eine neue Grafik.

Parameters:

backColor - Hintergrundfarbe des Textes

creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt

imageName - Der Dateiname der Grafik.

graphicLabel - Der Text der bei der Grafik stehen soll.

objID - Eine eindeutige Identifikationsnummer der Verbindung.

See Also:

Person

- **equals**

```
public boolean equals(Graphic graphic2)
```

- **getImageName**

```
public String getImageName()
```

- **labelGraphic**

```
public void labelGraphic(String graphicLabel)
```

- **setImageName**

```
public void setImageName(String imageName)
```

- **showGraphicLabel**

```
public String showGraphicLabel()
```

Class como.Line

```
java.lang.Object
   |
   +----como.PresentationObject
           |
           +----como.Line
```

---

public class **Line**
extends PresentationObject
implements Serializable Diese Klasse implementiert ein Material für Objekte, die auf der Arbeitsfläche benutzt werden sollen (Instanzen von Subklassen der abstrakten Klasse `PresentationObject`).
Bei diesem `PresentationObject` handelt es sich um einen Linie oder einen Pfeil (s. `LineType`
Zur Manipulation dieser Materialien steht das Werkzeug `PresentationObjectTool` zur Verfügung. Die Positionierung auf der Arbeitsfläche regelt das Werkzeug `BoardTool`.
See Also:
PresentationObjectTool, PresentationObject, BoardTool, LineType

---

Variables

_lineType

```
private LineType _lineType
```

Constructors

Line

```
public Line(Line line2)
```

Erstelle eine neue Linie auf der Basis der angegebenen Linie
Parameters:
line2 - die zu kopierende Linie

Line

```
public Line(Color lineColor,
            Person creator,
            int width,
            int height,
            LineType lineType,
            int objID)
```

Erstelle eine neue Linie.
Parameters:
lineColor - Farbe der Linie
creator - `Person`-Objekt des Teilnehmers, der die Verbindungslinie erstellt
width - Die Breite des die Linie umschließenden Rechtecks
height - Die Höhe des die Linie umschließenden Rechtecks
lineType - Typ der Linie (`LineType`)
objID - Eine eindeutige Identifikationsnummer der Verbindung.
See Also:
Person, LineType

Methods

equals

```
public boolean equals(Line line2)
```

getLineType

```
public LineType getLineType()
```

setLineType

```
public void setLineType(LineType lineType)
```

Class como.LineType

```
java.lang.Object
   |
   +----como.LineType
```

---

public class **LineType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Formen Linien, die auf der Arbeitsfläche dargestellt werden können, zu kapseln.
See Also:
createLine, Line

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

Alle Möglichen Typen einer Linie

BROKEN

```
public static final LineType BROKEN
```

Linientyp: Gestrichelt

SOLID

```
public static final LineType SOLID
```

Linientyp: normale Linie

BROKEN_ARROW

```
public static final LineType BROKEN_ARROW
```

Linientyp: Gestrichelter Pfeil

SOLID_ARROW

```
public static final LineType SOLID_ARROW
```

Linientyp: Pfeil

_type

```
private String _type
```

Constructors

LineType

```
public LineType(String type)
```

Methods

equals

```
public boolean equals(LineType type2)
```

getLineType

```
public String getLineType()
```

Returns:

Gibt den Typ der Linie zurück
See Also:
ALLTYPES
giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.ListOfPersons

```
java.lang.Object
   |
   +----como.ListOfPersons
```

---

public class **ListOfPersons**
extends Object
implements Serializable Diese Klasse repräsentiert die Liste der Teilnehmer einer Moderation. Ein neuer Teilnehmer wird bei seiner Anmeldung automatisch in die ListOfPersons der Moderation eingetragen.
See Also:
ListOfPersonsTool

Variables

_listOfPersons

```
private Vector _listOfPersons
```

Constructors

ListOfPersons

```
public ListOfPersons()
```

Erstelle eine neue Teilnehmerliste

Methods

addPerson

```
public void addPerson(Person person)
```

clear

```
public void clear()
```

contains

```
public boolean contains(Person person)
```

getAll

```
public Person[] getAll()
```

getFirst

```
public Person getFirst()
```

isEmpty

```
public boolean isEmpty()
```

removeFirst

```
public void removeFirst()
```

removePerson

```
public void removePerson(Person person)
```

Class como.ListOfPersonsMessageType

```
java.lang.Object
   |
   +----como.ListOfPersonsMessageType
```

---

public class **ListOfPersonsMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die die ListOfPersons betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

Alle gültigen Message-Namen für diese Klasse

NEW_LIST_OF_PERSONS

```
public static final ListOfPersonsMessageType NEW_LIST_OF_PERSONS
```

Nachricht, die veranlaßt, eine neue Teilnehmerliste zu erstellen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das hinzuzufügende `ListOfPersons`-Objekt.

ADD_PERSON_TO_LIST_OF_PERSONS

```
public static final ListOfPersonsMessageType
ADD_PERSON_TO_LIST_OF_PERSONS
```

Nachricht, die veranlaßt, eine neue Person zur Teilnehmerliste hinzuzufügen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das hinzuzufügende `Person`-Objekt.

REMOVE_PERSON_FROM_LIST_OF_PERSONS

```
public static final ListOfPersonsMessageType
REMOVE_PERSON_FROM_LIST_OF_PERSONS
```

Nachricht, die veranlaßt, eine neue Person aus der Teilnehmerliste zu entfernen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das zu entfernende `Person`-Objekt.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, ListOfPersonsMessageType...

Constructors

ListOfPersonsMessageType

```
public ListOfPersonsMessageType(String action)
```

Methods

equals

```
public boolean equals(ListOfPersonsMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.ListOfPersonsTool

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----como.ListOfPersonsTool
```

---

public class **ListOfPersonsTool**
extends Observable
implements Observer Die Klasse ListOfPersonsTool bildet die Funktionskomponente (FK) der Personenliste (`ListOfPersons`)
Änderungen, die den Client über das `NetworkTool` erreichen und die Personenliste betreffen, werden in der Methode `update()` abgefangen und an die IAK (hier die Klasse `PresentationTool` über den gleichen Mechanismus mit Hilfe der Methode `notifyObservers()` weitergegeben
See Also:
Observer, Observable

Variables

_myListOfPersons

```
ListOfPersons _myListOfPersons
```

_myNT

```
NetworkTool _myNT
```

_myLocalUser

```
Person _myLocalUser
```

Constructors

ListOfPersonsTool

```
public ListOfPersonsTool(NetworkTool nT,
                         Person myPerson)
```

Erstelle ein neues Werkzeug-Objekt der Klasse `ListOfPersonsTool`
Parameters:
fkN - referenz auf die lokale Netzwerkkomponente vom Typ `NetworkTool`.
myPerson - der lokale Benutzer des BoardTools vom Typ `Person`.
See Also:
NetworkTool, Person

Methods

add

```
private void add(Person newPerson)
```

remove

```
private void remove(Person remPerson)
```

removeUser

```
public void removeUser()
```
setNewListOfPersons

```
private void setNewListOfPersons(ListOfPersons newListOfPersons)
```
showAllPersons

```
public Person[] showAllPersons()
```
update

```
public void update(Observable obs,
                   Object arg)
```

Class como.ListOfPersonsToolMessageType

```
java.lang.Object
   |
   +----como.ListOfPersonsToolMessageType
```

---

public class **ListOfPersonsToolMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die das ListOfPersonsTool betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

REFRESH_LIST

```
public static final ListOfPersonsToolMessageType REFRESH_LIST
```

Nachricht, die die Beobachter informiert, daß sich die Teilnehmerliste verändert hat. Empfängt die IAK diese Nachricht, so sollte sie sich die neuen Daten vom `ListOfPersonsTool` besorgen.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, BoardMessageType...

Constructors

ListOfPersonsToolMessageType

```
public ListOfPersonsToolMessageType(String action)
```

Methods

equals

```
public boolean equals(ListOfPersonsToolMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:

toString in class Object

Class como.ListOfSpeakers

```
java.lang.Object
   |
   +----como.ListOfPersons
           |
           +----como.ListOfSpeakers
```

---

public class **ListOfSpeakers**
extends ListOfPersons
implements Serializable Diese Klasse repräsentiert die Rednerliste einer Moderation. Ein neuer Teilnehmer der sich in die Rednerliste eintragen möchte betätigt dazu den *Nächster*-Knopf in seinem Applet.
See Also:
ListOfSpeakersTool

Variables

_listOpen

```
private boolean _listOpen
```

Constructors

ListOfSpeakers

```
public ListOfSpeakers()
```

Methods

addPerson

```
public void addPerson(Person person)
```

Overrides:
addPerson in class ListOfPersons

closeList

```
public void closeList()
```

isListOpen

```
public boolean isListOpen()
```

openList

```
public void openList()
```

Class como.ListOfSpeakersMessageType

```
java.lang.Object
   |
   +----como.ListOfSpeakersMessageType
```

public class **ListOfSpeakersMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die die ListOfSpeakers betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

NEW_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType NEW_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, eine neue Rednerliste zu erstellen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das hinzuzufügende `ListOfSpeakers`-Objekt.

ADD_PERSON_TO_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType
ADD_PERSON_TO_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, eine neue Person zur Rednerliste hinzuzufügen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das hinzuzufügende `Person`-Objekt.

REMOVE_PERSON_FROM_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType
REMOVE_PERSON_FROM_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, eine neue Person aus der Rednerliste zu entfernen
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das zu entfernende `Person`-Objekt.

CLEAR_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType CLEAR_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, die Rednerliste zu löschen
Bei dieser Nachricht wird kein Datenobjekt erwartet.

CLOSE_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType CLOSE_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, die Rednerliste zu schließ. Das heißt, daß sich niemand mehr eintragen darf.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

OPEN_LIST_OF_SPEAKERS

```
public static final ListOfSpeakersMessageType OPEN_LIST_OF_SPEAKERS
```

Nachricht, die veranlaßt, die Rednerliste freizugeben

Bei dieser Nachricht wird kein Datenobjekt erwartet.

NEXT_SPEAKER

```
public static final ListOfSpeakersMessageType NEXT_SPEAKER
```

Nachricht, die veranlaßt, dem Redner sein Redner-Recht zu entziehen und den Teilnehmer, der auf der Rednerliste als nächstes aufgeführt ist, zum aktuellen Redner zu machen.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, ListOfSpeakersMessageType...

Constructors

ListOfSpeakersMessageType

```
public ListOfSpeakersMessageType(String action)
```

Methods

equals

```
public boolean equals(ListOfSpeakersMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.ListOfSpeakersTool

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----como.ListOfSpeakersTool
```

---

public class **ListOfSpeakersTool**
extends Observable
implements Observer Die Klasse ListOfSpeakersTool bildet die Funktionskomponente (FK) der Rednerliste (`ListOfSpeakers`)
Änderungen, die den Client über das `NetworkTool` erreichen und die Rednerliste betreffen, werden in der Methode `update()` abgefangen und an die IAK (hier die Klasse `PresentationTool` über den gleichen Mechanismus mit Hilfe der Methode `notifyObservers()` weitergegeben
See Also:
Observer, Observable

Variables

_myListOfSpeakers

```
private ListOfSpeakers _myListOfSpeakers
```

_myNT

```
private NetworkTool _myNT
```

_myLocalUser

```
private Person _myLocalUser
```

Constructors

ListOfSpeakersTool

```
public ListOfSpeakersTool(NetworkTool nT,
                          Person myPerson)
```

Erstelle ein neues Werkzeug-Objekt der Klasse `ListOfSpeakersTool`
Parameters:
nT - referenz auf die lokale Netzwerkkomponente vom Typ `NetworkTool`.
myPerson - der lokale Benutzer des BoardTools vom Typ `Person`.
See Also:
NetworkTool, Person

Methods

add

```
private void add(Person person)
```

addToSpeakerList

```
public void addToSpeakerList()
```

clear

```
 private void clear()
```
clearSpeakerlist

```
 public void clearSpeakerlist()
```
close

```
 private void close()
```
closeSpeakerList

```
 public void closeSpeakerList()
```
isAllowedToAdd

```
 public boolean isAllowedToAdd()
```
isAllowedToNext

```
 public boolean isAllowedToNext()
```
isAllowedToOpenCloseClear

```
 public boolean isAllowedToOpenCloseClear()
```
isSpeakerListOpen

```
 public boolean isSpeakerListOpen()
```
next

```
 private void next()
```
nextSpeaker

```
 public void nextSpeaker()
```
open

```
 private void open()
```
openSpeakerList

```
 public void openSpeakerList()
```
remove

```
 private void remove(Person person)
```
removeUser

```
 public void removeUser()
```
showListOfSpeakers

```
 public Person[] showListOfSpeakers()
```
update

```
 public void update(Observable obs,
                    Object arg)
```

Class como.ListOfSpeakersToolMessageType

```
java.lang.Object
   |
   +----como.ListOfSpeakersToolMessageType
```

---

public class **ListOfSpeakersToolMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die das ListOfSpeakersTool betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

REFRESH_LIST

```
public static final ListOfSpeakersToolMessageType REFRESH_LIST
```

Nachricht, die die Beobachter informiert, daß sich die Rednerliste verändert hat. Empfängt die IAK diese Nachricht, so sollte sie sich die neuen Daten vom `ListOfSpeakersTool` besorgen.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

LIST_OPENED

```
public static final ListOfSpeakersToolMessageType LIST_OPENED
```

Nachricht, die die Beobachter informiert, daß die Rednerliste freigegeben wurde.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

LIST_CLOSED

```
public static final ListOfSpeakersToolMessageType LIST_CLOSED
```

Nachricht, die die Beobachter informiert, daß die Rednerliste geschlossen wurde.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

_action

```
private String _action
```

Der Name der Message - also die Aktion die auf den Aktualisierungsdaten ausgefuehrt werden soll. Die moeglichen Typen sind in Klassen gekapselt z.B.: VersionMessageType, BoardMessageType...

Constructors

ListOfSpeakersToolMessageType

```
public ListOfSpeakersToolMessageType(String action)
```

Methods

equals

```
public boolean equals(ListOfSpeakersToolMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:

Gibt den Typ der Nachricht zurück

See Also:

_ALLTYPES

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:

toString in class Object

Class como.Message

```
java.lang.Object
   |
   +----como.Message
```

---

public class **Message**
extends Object
implements Serializable Alle Nachrichten, die im Package *presentationTool* über das Netz verschickt werden, sind Instanzen dieser Klasse.
Eine entsprechende Nachricht wird erzeugt, ein neues Objekt, dem im Konstruktor ein `messageType` also ein Typ der Nachricht übergeben wird, angelegt wird. Über den `messageType` erkennt der Empfänger der Nachricht, ob die Nachricht ihn etwas angeht und ob er sie weiterverarbeiten muß. Ist dies der Fall, so findet der Empfänger die zur Nachricht gehörigen Daten in den Feldern `data1` und wenn für diesen Nachrichtentyp noch mehr Daten benötigt werden auch noch im Feld `data2`

Variables

INIT

```
public static final String INIT
```

_messageType

```
private Object _messageType
```

Typ der Message - entscheidet von welchem Tool die Messsage verarbeitet werden soll!

_data1

```
private Object _data1
```

Daten-Objekt der Message - Hier stehen die Aktualisierungsdaten drin!

_data2

```
private Object _data2
```

optionales zweites Daten-OBjekt - Hier kann ein zusaetzliches Objekt mit weiteren Aktualisierungsdaten uebergeben werden wenn noetig! Z.B.: Uebergabe eines PresentationObjects + neue Position beim Verschieben

Constructors

Message

```
public Message(Object messageType)
```

Message

```
public Message(Object messageType,
               Object data1)
```

Message

```
public Message(Object messageType,
               Object data1,
               Object data2)
```

Methods

- **getData1**

```
public Object getData1()
```

- **getData2**

```
public Object getData2()
```

- **getMessageType**

```
public Object getMessageType()
```

Class como.MyText

```
java.lang.Object
   |
   +----como.MyText
```

---

public class **MyText**
extends Object
implements Serializable Diese Klasse bildet die Struktur des Textes, der auf den `PresentationObjects` der Arbeitsfläche dargestellt werden soll.
See Also:
FreeText, Connection, Card

Variables

_text

```
private String _text
```

_color

```
private Color _color
```

_size

```
private TextSize _size
```

Constructors

MyText

```
public MyText(MyText myText)
```

MyText

```
public MyText(String text,
              Color color,
              TextSize size)
```

Methods

equals

```
public boolean equals(MyText myText2)
```

getColor

```
public Color getColor()
```

getText

```
public String getText()
```

getTextSize

```
public TextSize getTextSize()
```

setColor

```
public void setColor(Color color)
```

setText

```
public void setText(String text)
```

setTextSize

```
public void setTextSize(TextSize size)
```

Class como.NConnection

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----como.NConnection
```

class **NConnection**
extends Thread
This class is not public and therefore cannot be used outside this package.

Class como.NetworkTool

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----como.NetworkTool
```

---

public class **NetworkTool**
extends Observable
implements Runnable Die Klasse `NetworkTool` ist für die gesamte Kommunikation über das Netzwerk zuständig. Eine Instanz der Klasse regelt den Verbindungsaufbau, Verbindungsabbau und die Synchroinisation zum Server wä,hrend einer Moderation.
Objekte, die sich bei einer Instanz dieser Klasse als Observer angemeldet haben, werden mit allen Nachrichten,die aus dem Netz empfangen werden versorgt. Dabei sind alle Nachrichten Instanzen der Klasse `Message`.
See Also:
Message

Variables

_s

```
private Socket _s
```

for networking

_host

```
private String _host
```

_inObj

```
private ObjectInputStream _inObj
```

_outObj

```
private ObjectOutputStream _outObj
```

_myID

```
private int _myID
```

Internal state

_channel

```
private String _channel
```

_kicker

```
private Thread _kicker
```

_port

```
private int _port
```

Port on server to connect with.

Constructors

NetworkTool

```
public NetworkTool(int port)
```

Methods

announce

```
public void announce(String msg) throws IOException
```

Announce a message. This method sends an announcement and is used to inform the world that someone has entered or left the conversation.

Parameters:

msg - Message to be announced.

attempt_reconnection

```
private void attempt_reconnection() throws IOException
```

In case of trouble, try to restore the connection. The first attempt is discreet (no need to spread panic) but if that fails, inform the user and keep trying. This falls under the "don't tell them there is a problem" strategy. The worst case scenario: a user types a message, hits "send" and nothing happens. Moments later everything seems to work fine. The user is puzzled, but oblivious to the fact that the server may have just crashed and restarted.

checkAnnounce

```
protected boolean checkAnnounce(String tok)
```

Look for announcements

closeConnection

```
public void closeConnection(String senderName) throws IOException
```

Say goodbye and break all connections

displayMessage

```
public void displayMessage(String fromName,
                           String message)
```

Displays a message to the user. If the message is an announcement of some sort, just put it right on the screen. If the message is from someone, we put in the following (arbitrarily chosen) format. Bob says, "Hi there." Override this method if you want to change the look and feel of how messages are displayed.

Parameters:

fromName - The name of the person who sent the message

message - What that person said

initClient

```
public void initClient(String channel)
```

Weise dieser Netzwerkkomponente den angegebenen Channel zu.

Parameters:

channel - der Channel für dieses Applet.

make_connection

```
private boolean make_connection() throws IOException
```

Make a network connnection to known host

make_connection

```
public boolean make_connection(String host) throws IOException
```

Make a network connnection to the server

parse

```
private void parse(String line)
```

Parses incoming data from the network

run

```
public void run()
```

Listen for messages from the net.

send

```
public void send(Message msg,
                 String senderName)
```

Send a message

Class como.Person

```
java.lang.Object
   |
   +----como.Person
```

---

public class **Person**
extends Object
implements Serializable Diese Klasse enthält die Daten eines Teilnehmers. Zu diesen Daten gehören neben den Namen und der UserID auch die Rechte, die ein Teilnehmer hat. Die Werte dieser Rechte sind in der Klasse `PresentationRights` gekapselt.
See Also:
PresentationRights

Variables

_firstName

```
private String _firstName
```

_lastName

```
private String _lastName
```

_userID

```
private String _userID
```

_rights

```
private PresentationRights _rights
```

Constructors

Person

```
public Person(Person person2)
```

Erstelle einen neuen Teilnehmer auf der Basis der angegebenen Person

Person

```
public Person(String firstName,
              String lastName,
              String userID)
```

Erstelle einen neuen Teilnehmer mit den folgenden Daten:
Parameters:
firstname - Vorname
lastname - Nachname
userID - eine moderationsweit eindeutige Identifikationsnummer

Methods

equals

```
public boolean equals(Person person2)
```

Vergleicht das aktuelle Objekt der Klasse Person mit dem übergebenen Objekt. dabei werden die jeweiligen Rechte ignoriert.

getFirstName

```
public String getFirstName()
```

getLastName

```
public String getLastName()
```

getUserID

```
public String getUserID()
```

hasPresenterRights

```
public boolean hasPresenterRights()
```

Returns:
true - wenn diese Person Moderator ist.

hasSpeakerRights

```
public boolean hasSpeakerRights()
```

Returns:
true - wenn diese Person Redner-Rechte besitzt.

loosePresenterRights

```
public void loosePresenterRights()
```

Entziehe dieser Person die Moderator-Rechte.

looseSpeakerRights

```
public void looseSpeakerRights()
```

Entziehe dieser Person die Redner-Rechte.

receivePresenterRights

```
public void receivePresenterRights()
```

Weise dieser Person Moderator-Rechte zu.

receiveSpeakerRights

```
public void receiveSpeakerRights()
```

Weise dieser Person Redner-Rechte zu.

setUserID

```
public void setUserID(String userID)
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.PresentationObject

```
java.lang.Object
   |
   +----como.PresentationObject
```

public abstract class **PresentationObject**
extends Object
implements Serializable Das `PresentationObject` bildet die Superklasse für die Arbeits-Materialien des Metaplan- und MindMapBoards.
See Also:
FreeText, Card, Line, Graphic, Board, Connection

Variables

_presObjID

```
private int _presObjID
```

_size

```
private Dimension _size
```

_color

```
private Color _color
```

_creator

```
private Person _creator
```

Constructors

PresentationObject

```
public PresentationObject()
```

Methods

equals

```
public boolean equals(PresentationObject presObj2)
```

Vergleich des aktuellen PresentationObjects mit dem übergebenen. In der erbenden Klasse zu Implementieren.

equalsID

```
public boolean equalsID(PresentationObject presObj2)
```

getColor

```
public Color getColor()
```

liefert die Farbe des PresentationObjects.
Returns:
`Color` Es wird ein `java.awt.Color` Object zurück geliefert.

getColorRGB

```
public int[] getColorRGB()
```

liefert die RGB Werte der Farbe des PresentationObjects als integer Array.

Returns:
`int[3]` [0] liefert Rot, [1] liefert Grün, [2] liefert Blau.

getCreator

```
public Person getCreator()
```

liefert die `Person` die das Objekts erstellt hat.

Returns:
liefert ein `Person` Objekt zurück das den Objekt-Ersteller enthält.

getID

```
public int getID()
```

liefert die in der Präsentation eindeutige ID des Presentation Objekts.

Returns:
liefert ein `int`

getSize

```
public Dimension getSize()
```

liefert die Größe des PresentationObject's zurück

Returns:
liefert ein `java.awt.Dimension` Objekt.

init

```
public void init(PresentationObject presObj)
```

init

```
public void init(Color objColor,
                 Person creator,
                 int width,
                 int height,
                 int objID)
```

Initialisierung der Klasse.

Parameters:
objColor - Die Färbung des Objekts als `java.awt.Color` Objekt.
creatorID - Die Benutzerkennung des Benutzers der das Objekt erstellt hat als `Person` Objekt. // * @param topLeft_x Die x-Position der linken, oberen Ecke des // * `PresentationObjects` auf einem // * `Board` Objekt als `int`. // * @param topLeft_y Die y-Position der linken, oberen Ecke des // * `PresentationObjects` auf einem // * `Board` Objekt als `int`.
width - Die Breite des PresentationObjects in Pixeln als `int` wert.
height - Die Höhe des PresentationObjects in Pixeln als `int` wert.
die - in der gesamten Präsentation eindeutige PresentationObjectID als `int` wert.

setColor

```
public void setColor(int red,
                     int green,
                     int blue)
```

die Grundfarbe des Objekts wird neu gesetzt.

Parameters:
red - Der rot Anteil der Farbe als `int`.
green - Der grün Anteil der Farbe als `int`.
blue - Der blau Anteil der Farbe als `int`.

setColor

```
public void setColor(Color color)
```

die Grundfarbe des Objekts wird neu gesetzt.

Parameters:
color - Die zu setzende Farbe als `java.awt.Color` Objekt.

setSize

```
public void setSize(int width,
                    int height)
```

die Breite und Höhe des Objekts wird neu gesetzt.

Parameters:

width - Die Pixelbreite des Objekts als `int`.

height - Die Pixelhöhe des Objekts als `int`.

setSize

```
public void setSize(Dimension size)
```

die Breite und Höhe des Objekts wird neu gesetzt.

Parameters:

size - Die Größe des Objekts als `java.awt.Dimension`.

Class como.PresentationObjectMessageType

```
java.lang.Object
   |
   +----como.PresentationObjectMessageType
```

---

public class **PresentationObjectMessageType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die die PresentationObjects betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

NEXT_PRESOBJ_NO

```
public static final PresentationObjectMessageType NEXT_PRESOBJ_NO
```

Nachricht, die veranlaßt, daß die die Nummer fü das nächste `PresentationObject` übergibt.
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar die neue Nummer.

ADD_NEW_PRESOBJ

```
public static final PresentationObjectMessageType ADD_NEW_PRESOBJ
```

Nachricht, die veranlaßt, daß ein neues Objekt erzeugt wird.
Bei dieser Nachricht wird ein Datenobjekt erwartet und zwar das neue `PresentationObject`.

INC_PRESOBJ_NO

```
public static final PresentationObjectMessageType INC_PRESOBJ_NO
```

Nachricht, die veranlaßt, daß die Nummer fü das nächste `PresentationObject` hochgezält wird.
Bei dieser Nachricht wird kein Datenobjekt erwartet.

_action

```
private String _action
```

Der Name der Message - also der Aktion die auf den Clients ausgefuehrt werden soll.

Constructors

PresentationObjectMessageType

```
public PresentationObjectMessageType(String action)
```

Methods

equals

```
public boolean equals(PresentationObjectMessageType type2)
```

getMessageType

```
public String getMessageType()
```

Returns:
Gibt den Typ der Nachricht zurück
See Also:
ALLTYPES
giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.PresentationObjectTool

```
java.lang.Object
   |
   +----java.util.Observable
           |
           +----como.PresentationObjectTool
```

public class **PresentationObjectTool**
extends Observable
implements Observer Die Klasse PresnetationObjectTool bildet die Funktionskomponente (FK) für die Objekte Auf der Arbeitsfläche (`PresentationObject`)
Änderungen, die den Client über das `NetworkTool` erreichen und die Objekte auf der Arbeitsfläche betreffen, werden in der Methode `update()` abgefangen und an die IAK über den gleichen Mechanismus mit Hilfe der Methode `notifyObservers()` weitergegeben
See Also:
Observer, Observable

Variables

ORANGE

```
public static final Color ORANGE
```

Dies sind die für Metaplankarten zugelassenen Farben : `PresentationTool.ORANGE` `PresentationTool.GRUEN` `PresentationTool.WEISS` `PresentationTool.GELB`

GRUEN

```
public static final Color GRUEN
```

WEISS

```
public static final Color WEISS
```

GELB

```
public static final Color GELB
```

_myLocalUser

```
private Person _myLocalUser
```

_myNT

```
private NetworkTool _myNT
```

_nextPresentationObjectNo

```
private int _nextPresentationObjectNo
```

Constructors

PresentationObjectTool

```
public PresentationObjectTool(NetworkTool nT,
                              Person myPerson)
```

Methods

checkColorOK

```
protected boolean checkColorOK(Color color)
```

createCard

```
public Card createCard(Color color,
                       CardShape cShape,
                       MyText cardText)
```

createConnection

```
public Connection createConnection(Color color,
                                   int startPresObjID,
                                   int endPresObjID)
```

createFreeText

```
public FreeText createFreeText(Color color,
                               MyText freeText)
```

createGraphic

```
public Graphic createGraphic(Color color,
                             String imageName,
                             String graphicLabel)
```

createLine

```
public Line createLine(Color color,
                       int width,
                       int height,
                       LineType lineType)
```

update

```
public void update(Observable obs,
                   Object arg)
```

Class como.PresentationRights

```
java.lang.Object
   |
   +----como.PresentationRights
```

---

public class **PresentationRights**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Rechte, die ein Teilnehmer in einer Moderation hat, zu kapseln. Dabei stehen drei Möglichkeiten zur Verfügung:
`PRESENTER` Der Teilnehmer hat alle Rechte und übernimmt die Rolle des Moderators. Insbesondere sind für ihn die Knöpfe *Unterbrechen, Liste Löschen, Liste freigeben, Liste schlie&suml;en* aktiviert.
`SPEAKER` Der Teilnehmer ist der aktuelle Redner und darf die Arbeitsfläche bearbeiten
`PARTICIPANT` Der Teilnehmer hat momentatn keine sonder Rechte und kann die Moderation lediglich passiv verfolgen, sich in die Rednerliste eintragen oder am Chat beteiligen.
See Also:
Person

Variables

_ALLRIGHTS

```
private static final String _ALLRIGHTS[]
```

Alle m&oumlglichen Moderationsrechte, die ein Teilnehmer haben kann

SPEAKER

```
public static final PresentationRights SPEAKER
```

Moderationsrecht: Redner (darf die Arbeitsfläche bearbeiten)

PRESENTER

```
public static final PresentationRights PRESENTER
```

Moderationsrecht: Moderator

PARTICIPANT

```
public static final PresentationRights PARTICIPANT
```

Moderationsrecht: Moderator (darf nur zugucken und sich in die Rendnerliste eintragen)

_rights

```
private String _rights
```

Constructors

PresentationRights

```
public PresentationRights(String rights)
```

Methods

equals

```
public boolean equals(PresentationRights rights2)
```

getPresentationRights

```
public String getPresentationRights()
```

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.PresentationType

```
java.lang.Object
   |
   +----como.PresentationType
```

---

public class **PresentationType**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Typen der Moderationen, die das Applet verwalten können soll, zu kapseln.
See Also:
createBoard

Variables

_ALLTYPES

```
private static final String _ALLTYPES[]
```

Alle möglichen Moderationsformen - Soll erweitert werden!

METAPLAN

```
public static final PresentationType METAPLAN
```

Moderationsform: Metaplan

MINDMAP

```
public static final PresentationType MINDMAP
```

Moderationsform: Mind Map

_type

```
private String _type
```

Constructors

PresentationType

```
public PresentationType(String type)
```

Methods

equals

```
public boolean equals(PresentationType type2)
```

getPresentationType

```
public String getPresentationType()
```

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.TextSize

```
java.lang.Object
   |
   +----como.TextSize
```

---

public class **TextSize**
extends Object
implements Serializable Diese Klasse dient lediglich dazu, die Grö&suml;e der Texte, die auf den `PresentationObjects` angezeigt werden, zu kapseln.
See Also:
MyText

Variables

_ALLSIZES

```
private static final String _ALLSIZES[]
```

Alle Möglichen Textgrößen

NORMAL

```
public static final TextSize NORMAL
```

Textgröße: Normal

BIG

```
public static final TextSize BIG
```

Textgröße: Groß

_size

```
private String _size
```

Constructors

TextSize

```
public TextSize(String size)
```

Methods

equals

```
public boolean equals(TextSize size2)
```

getTextSize

```
public String getTextSize()
```

giveAll

```
public static String[] giveAll()
```

toString

```
public String toString()
```

Overrides:
toString in class Object

Class como.VersionMessageType

```
java.lang.Object
   |
   +----como.VersionMessageType
```

---

public class **VersionMessageType**
extends Object
implements Serializable
Diese Klasse dient lediglich dazu, die Namen der Nachrichten, die das VersionTool betreffen, zu kapseln. Eine entsprechende Nachricht wird erzeugt, indem die Klasse `Message` mit einem dieser Namen als `messageType` instanziert wird.
See Also:
Message

Constructors

VersionMessageType

```
public VersionMessageType()
```

Class como.VersionTool

```
java.lang.Object
   |
   +----como.VersionTool
```

---

public class **VersionTool**
extends Object

Variables

currentBoard

```
Board currentBoard
```

currentListOfPersons

```
ListOfPersons currentListOfPersons
```

Constructors

VersionTool

```
public VersionTool()
```

Methods

loadBoard

```
public Board loadBoard(String filename)
```

saveBoard

```
public void saveBoard(String filename)
```

Class como.Vulture

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----como.Vulture
```

---

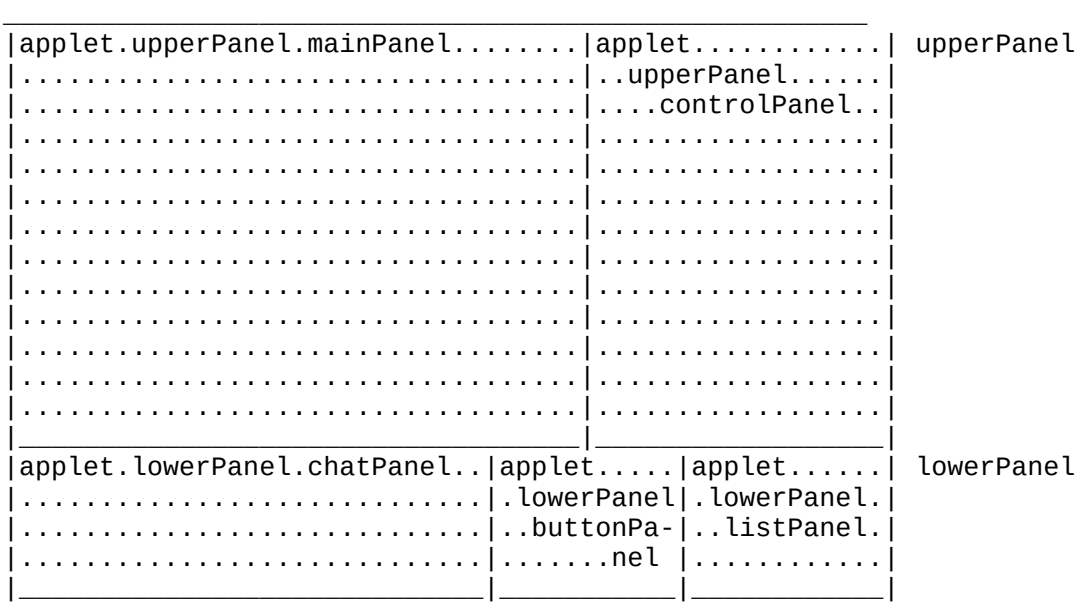
class **Vulture**
extends Thread
This class is not public and therefore cannot be used outside this package.

Class como.presentationTool

`como.presentationTool`

---

public class **presentationTool**
implements ActionListener, Observer Diese Klasse bildet die Interaktionskomponente (IAK) der Arbeitsumgebung.
Die IAK der Arbeitsfläche findet sich in der Klasse BoardUI
PresentationTool basiert auf der Klasse `Annoying Chat` von Jeff Breidenbach, Copyright 1996.
Das Chattool gibt es bisher noch nicht! Bisher alles hier implementiert!
Die GUI-Struktur sieht folgendermassen aus:
`applet`



Variables

_colorStrings

```
private static final String _colorStrings[]
```

_colors

```
private static final Color _colors[]
```

_textColorStrings

```
private static final String _textColorStrings[]
```

_textColors

```
private static final Color _textColors[]
```

isStandalone

```
private static final boolean isStandalone
```

gbLayout

```
private GridBagLayout gbLayout
```

upperPanel

```
private JPanel upperPanel
```
controlPanel

```
private JPanel controlPanel
```
zoomLabel

```
private JLabel zoomLabel
```
zoomComboBox

```
private JComboBox zoomComboBox
```
folderLabel

```
private JLabel folderLabel
```
imageFolderComboBox

```
private JComboBox imageFolderComboBox
```
textColorLabel

```
private JLabel textColorLabel
```
textColorComboBox

```
private JComboBox textColorComboBox
```
backgroundColorLabel

```
private JLabel backgroundColorLabel
```
backgroundColorComboBox

```
private JComboBox backgroundColorComboBox
```
board

```
private BoardUI board
```
lowerPanel

```
private JPanel lowerPanel
```
chatPanel

```
private JPanel chatPanel
```
_inputfield

```
private JTextField _inputfield
```
_outputarea

```
private JTextArea _outputarea
```
_chatScrollOutputArea

```
private JScrollPane _chatScrollOutputArea
```
_chatLabel

```
private JLabel _chatLabel
```
_losButtonPanel

```
private JPanel _losButtonPanel
```
jspSpeakersList

```
private JScrollPane jspSpeakersList
```
_codeBase

```
private URL _codeBase
```
_iconOpenDir

```
private URL _iconOpenDir
```
_iconAdd

```
private URL _iconAdd
```
_iconNext

```
private URL _iconNext
```
_iconInterrupt

```
private URL _iconInterrupt
```
_iconDelete

```
private URL _iconDelete
```
_iconOpenList

```
private URL _iconOpenList
```
_iconCloseList

```
private URL _iconCloseList
```
_iconPresenter

```
private URL _iconPresenter
```
_iconPerson

```
private URL _iconPerson
```
_iconBlue

```
private URL _iconBlue
```
_iconRed

```
private URL _iconRed
```
_iconBrown

```
private URL _iconBrown
```
_iconYellow

```
private URL _iconYellow
```
_iconGreen

```
private URL _iconGreen
```
_iconWhite

```
private URL _iconWhite
```
_iconOrange

```
private URL _iconOrange
```
_iconBlack

```
private URL _iconBlack
```
_losAddButton

```
private JButton _losAddButton
```
_losNextButton

```
private JButton _losNextButton
```
_losInterruptButton

```
private JButton _losInterruptButton
```

_losClearButton

```
private JButton _losClearButton
```

_losOpenButton

```
private JButton _losOpenButton
```

_losCloseButton

```
private JButton _losCloseButton
```

jMenuBar1

```
private JMenuBar jMenuBar1
```

menu_file

```
private JMenu menu_file
```

menu_item_new

```
private JMenuItem menu_item_new
```

menu_item_open

```
private JMenuItem menu_item_open
```

menu_item_save

```
private JMenuItem menu_item_save
```

menu_item_save_to

```
private JMenuItem menu_item_save_to
```

menu_item_add_to_speakerlist

```
private JMenuItem menu_item_add_to_speakerlist
```

menu_item_exit

```
private JMenuItem menu_item_exit
```

menu_view

```
private JMenu menu_view
```

menu_item_50

```
private JMenuItem menu_item_50
```

menu_item_100

```
private JMenuItem menu_item_100
```

menu_item_200

```
private JMenuItem menu_item_200
```

menu_item_zoom

```
private JMenuItem menu_item_zoom
```

menu_insert

```
private JMenu menu_insert
```

menu_item_card

```
private JMenuItem menu_item_card
```

menu_item_ellipse

```
private JMenuItem menu_item_ellipse
```

menu_item_line

```
private JMenuItem menu_item_line
```
menu_item_circle

```
private JMenuItem menu_item_circle
```
menu_item_cloud

```
private JMenuItem menu_item_cloud
```
menu_persons

```
private JMenu menu_persons
```
jMenuItem16

```
private JMenuItem jMenuItem16
```
menu_windows

```
private JMenu menu_windows
```
menu_help

```
private JMenu menu_help
```
menu_item_about

```
private JMenuItem menu_item_about
```
_userLogedIn

```
private boolean _userLogedIn
```
_nwt

```
private NetworkTool _nwt
```
_loPersonsT

```
private ListOfPersonsTool _loPersonsT
```
_loSpeakersT

```
private ListOfSpeakersTool _loSpeakersT
```
_boardTool

```
private BoardTool _boardTool
```
_myLocalUser

```
private Person _myLocalUser
```
_myName

```
private String _myName
```

Constructors

presentationTool

```
public presentationTool()
```

Methods

()" "

```
static void ()
```
aboutBox

```
private void aboutBox()
```
Zeigt eine Dialog-Box an, die Informationen zum Programm gibt.

actionPerformed

```
public void actionPerformed(ActionEvent e)
```
Abfangen des Ereignisses `actionPerformed`, das auftritt, wenn der Teilnehmer eine Eingabe irgendeiner Art macht. Die Eingaben werden dann an das zuständige Werkzeug zur Bearbeitung weitergeleitet.

See Also:
ListOfPersonsTool, ListOfSpeakersTool, PresentationObjectTool, BoardTool

appendText

```
synchronized void appendText(String s)
```
Hängt den Text an den vorhandenen Text im Chat-Fenster an.

Parameters:
s - Der text, der angehängt werden soll.

constrain

```
public void constrain(Container container,
                      Component component,
                      int grid_x,
                      int grid_y,
                      int grid_width,
                      int grid_height)
```
Shortcut: Set the GridBagConstrain object for a component in a container to use GridbagLayout according to the following parameters Set the following default-values for the remaining GridBagConstrain-parameters: `fill` = GridBagConstraints.NONE `anchor` = GridBagConstraints.EAST `weightx` = 0.0 `weighty` = 0.0 `top` = 5 `bottom` = 5 `left` = 5 `right` = 5

Parameters:
container - the container to which the gridbag layout shall be applied to
component - the component to be inserted
grid_x - the x-value of the GridBagConstrain
grid_y - the y-value of the GridBagConstrain
grid_width - the gridwidth-value of the GridBagConstrain
grid_height - the gridheight-value of the GridBagConstrain

constrain

```
public void constrain(Container container,
                      Component component,
                      int grid_x,
                      int grid_y,
                      int grid_width,
                      int grid_height,
                      int fill,
                      int anchor,
                      double weight_x,
                      double weight_y,
                      int top,
                      int left,
                      int bottom,
                      int right)
```
Shortcut: Set the GridBagConstrain object for a component in a container to use GridbagLayout according to the following parameters

Parameters:
container - the container to which the gridbag layout shall be applied to
component - the component to be inserted
grid_x - the x-value of the GridBagConstrain

grid_y - the y-value of the GridBagConstrain
grid_width - the gridwidth-value of the GridBagConstrain
grid_height - the gridheight-value of the GridBagConstrain
fill - the fill-value of the GridBagConstrain
anchor - the anchor-value of the GridBagConstrain
weight_x - the weight_x-value of the GridBagConstrain
weight_y - the weight_y-value of the GridBagConstrain
top - the top-attribute of the Insets object of the GridBagConstrain
bottom - the bottom-attribute of the Insets object of the GridBagConstrain
left - the left-attribute of the Insets object of the GridBagConstrain
right - the right-attribute of the Insets object of the GridBagConstrain

constrain

```
public void constrain(Container container,
                      Component component,
                      int grid_x,
                      int grid_y,
                      int grid_width,
                      int grid_height,
                      int top,
                      int left,
                      int bottom,
                      int right)
```

Shortcut: Set the GridBagConstrain object for a component in a container to use GridbagLayout according to the following parameters Set the following default-values for the remaining GridBagConstrain-parameters: `fill` = GridBagConstraints.NONE `anchor` = GridBagConstraints.EAST `weightx` = 0.0 `weighty` = 0.0

Parameters:

container - the container to which the gridbag layout shall be applied to
component - the component to be inserted
grid_x - the x-value of the GridBagConstrain
grid_y - the y-value of the GridBagConstrain
grid_width - the gridwidth-value of the GridBagConstrain
grid_height - the gridheight-value of the GridBagConstrain
top - the top-attribute of the Insets object of the GridBagConstrain
bottom - the bottom-attribute of the Insets object of the GridBagConstrain
left - the left-attribute of the Insets object of the GridBagConstrain
right - the right-attribute of the Insets object of the GridBagConstrain

destroy

```
public void destroy()
```

enter_conversation

```
private void enter_conversation()
```

Sorgt für die richtigen Ausgaben im Chat-Fenster nach dem Betreten einer Moderation

getAppletInfo

```
public String getAppletInfo()
```

getParameter

```
public String getParameter(String key,
                           String def)
```

getParameterInfo

```
public String[][] getParameterInfo()
```

getUserData

```
private Person getUserData()
```

In drei aufeinanderfolgenden Dialogen wird der Teilnehmer nach seinen Daten gefragt:
1. Funktion (Moderator oder einfacher Teilnehmer einer Moderation
2. Vorname
3. Nachname

Returns:
das `Person`-Objekt, das den Teilnehmer dieses Applets darstellt.

See Also:
PresentationRights

gotFocus

```
public boolean gotFocus(Event e,
                        Object arg)
```

Erspart einem das Anklicken des Chat-Fensters wenn man Text eingeben will.

init

```
public void init()
```

Applet initialisieren GUI initialisieren

initiateBackgroundColor

```
private void initiateBackgroundColor()
```

Initialisiert den Selector für die Hintergrundfarbe

initiateBoardTool

```
private void initiateBoardTool(NetworkTool nwt,
                               Person user)
```

Initialisiert die Arbeitsfläche und verbindet sie mit dem Netzewrk.

initiateChatPannel

```
private void initiateChatPannel()
```

Hier wird das Pannel für das ChatTool Initialisiert.

initiateIconURLs

```
private void initiateIconURLs()
```

Hier werden die URLs der Icons angegeben, damit diese über das Netz geladen werden können.

initiateImgFolderSelector

```
private void initiateImgFolderSelector()
```

Initialisiert den Folder Selector

initiateListOfSpeakersButtons

```
private void initiateListOfSpeakersButtons()
```

initiateLists

```
private void initiateLists()
```

Initialisiert die Teilnehmer- und die Rednerliste und verbindet sie mit dem Netzewrk.

initiateLocal

```
private void initiateLocal()
```

Wenn der Teilnehmer noch nicht eingelogged ist, dann veranlaßt diese Methode das. Insbesondere wird das Netzwerk, NetzwerRedner- und Teilnehmerliste und die Arbeitsfläche initialisiert und eine Verbindung zum Server hergestellt.
Die folgenden Methoden dieser Klasse werden dabei aufgerufen:

See Also:
initiateNetwork, initiateLists, initiateBoardTool, enter_conversation

initiateMenu

```
private void initiateMenu()
```

Initialisiert die Menüleiste des Applets.

initiateNetwork

```
private void initiateNetwork()
```

Initialisiert das Netzewrk, verbindet die Benutzungsoberfläche (IAK) mit dem Netzwerk und gibt entsprechende Statusmeldungen im Chat-Fenster aus.

initiateTextColorSelector

```
private void initiateTextColorSelector()
```

Initialisiert den Selector für die TextFarbe

initiateZoomBox

```
private void initiateZoomBox()
```

Initialisiert die Zoom Box

jbInit

```
private void jbInit() throws Exception
```

Initialisiert die grafischen Komponenten der Benutzungsoberfläche (IAK) Zusätzlich werden die entsprechenden Initialisierungsmethoden aufgerufen, um die grafischen Komponenten zu initialisieren:

See Also:

getCodeBase, initiateIconURLs, getUserData, initiateListOfSpeakerButtons, initiateLocal

quitBox

```
private void quitBox()
```

Zeigt eine Dialog-Box an, in der eine Bestätigung zum Verlassen der Moderation und des Programms nachgefragt wird.

refreshJSPSpeakersList

```
private void refreshJSPSpeakersList()
```

Frischt die Anzeige der Rednerliste auf. Es werden die Daten des `ListOfSpeakersTools` verwendet.

See Also:

ListOfSpeakersTool

refreshLOSButtons

```
private void refreshLOSButtons()
```

Frischt die Anzeige der Knöpfe zur Bedienung der Rednerliste neu auf. Dabei gilt folgende Logik:
MODERATOR:
- Zu Beginn ist der Knopf "Liste freigeben" aktiviert und "Liste schließen" deaktiviert.
- Bei jedem Klick auf "Liste freigeben" oder "Liste schließen" invertiert sich die Aktivierung beider Knöpfe
- "Eintragen ist immer deaktiviert, weil der Moderator sich durch den Knopf "Unterbrechen" jederzeit zum Redner machen kann.
TEILNEHMER:
- "Liste freigeben", "Liste schließen", "Unterbrechen" und "Liste löschen" sind immer deaktiveirt - das sind Moderator-Funktionen.
- Zu Beginn ist der Knopf "Eintragen" aktiviert und "Nächster" deaktiviert.
- Der Knopf "Nächster" ist nur dann aktiviert wenn der betreffende Benutzer der derzeitige Redner ist.

refreshMenuPersons

```
private void refreshMenuPersons()
```

Frischt die Anzeige der Teilnehmerliste (Menü "Anwesende") auf. Es werden die Daten des `ListOfPersonsTools` verwendet.

See Also:
ListOfPersonsTool

start

```
public void start()
```

Initialisierung des Applets durch die private Methode `initiateLocal`

stop

```
public void stop()
```

Versendet eine Abschieds-Message und beendet die Verbindung.

See Also:
USER_LEAVES_SESSION

update

```
public void update(Observable obs,
                   Object arg)
```

This method is called whenever the observed object is changed. An application calls an Observable object's notifyObservers method to have all the object's observers notified of the change. Die Klasse presentationObject hat sich bei den Werkzeugen als `Observer` registriert, um über Änderungen informiert zu werden. Tritt nun eine Änderung bei einem dieser Werkzeuge auf, dann ruft dieses die Methode `notifyObservers` welche wiederum diese Methode `update` hier aufruft.

Parameters:
obs - das Werkzeug, das die Methode `notifyObservers` aufgerufen hat.
arg - ein vom Werkzeug übergebenes Message Objekt.

See Also:
Message, notifyObservers, notifyObservers, notifyObservers, notifyObservers, notifyObservers

wordwrap

```
private void wordwrap(String message)
```

Brichtr lange Zeilen, die in das Chatfenster geschrieben werden sollen an der richtigen Stelle um.
KLAPPT LEIDER NOCH NICHT!

Parameters:
message - Der anzuzeigende und umzubrechende Text

Class como.presentationToolServer

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----como.presentationToolServer
```

---

public class **presentationToolServer**
extends Thread PresentationToolServer stellt den Server für eine Moderation mit dem presentationTool dar. PresentationToolServer ist eine Erweiterung der Klasse Multiserver von Jeff Breidenbach, copyright May 1996, welcher wiederum auf Beispiel 9-3 aus Java in a Nutshell von David Flanagan beruht. Es folgt der Originalkommentar:
Modifications include:

- -v verbose switch
- Multiple channels
- Star connection between clients
- Simple handshaking (WELCOME, CHANNEL, OBITUARY)

Multiserver embodies the philosophy that all intelligence and work in a client/server system should be done by the client. The server should be as simple as possible, whose only function is to relay messages between the clients. Multiserver was designed to do as little as possible, and still provide enough power for a complex system.
The complete behavior of Multiserver can be summed up in a single paragraph.
Any message sent to Multiserver will be broadcast to all clients on the same channel, with the following special cases. On connection, the client will receive the message: 0 WELCOME n where n is a unique, non-zero ID number. It will also be assigned to a default channel. When a client disconnects, all other clients on the same channel will receive the message: 0 OBITUARY n where n is the ID number of the deceased. Finally, the client may change its channel by sending the message: CHANNEL newchannel where newchannel is the name of the new channel (string) A channel change is not broadcast to anyone.
You may communicate with this server using the Client in example 9-4 from Java in a Nutshell.
Verion history:

- 1.5 Very experimental asynchronous version
- 1.4 Changed synchronization/sleeping to improve reliability.
- 1.3 Fixed subtle bug in broadcasting, added "quit" capability
- 1.2 Added channel information to the verbose output
- 1.1 Fixed a major bug that was causing channels to bleed into each other
- 1.0 Original release, May 1996

1.5 (Experimental high efficiency version, no graphics)
Author:
David Flanagan, Jeff Breidenbach, Henrik Ortlepp & Wolfgang Riese

---

variables

DEFAULT_PORT

```
protected static final int DEFAULT_PORT
```
Default port to listen on

port

```
private int port
```

listen_socket

```
private ServerSocket listen_socket
```

threadgroup

```
protected ThreadGroup threadgroup
```

connections

```
protected Vector connections
```

vulture

```
protected Vulture vulture
```

loggedMessages

```
protected Vector loggedMessages
```
Save all sent Messages to be able to inititalize new client

Constructors

presentationToolServer

```
public presentationToolServer(int port,
                              boolean verbose)
```
Erstelle ein Server-Objetk für eine Moderation. Der Server wartet auf der `ServerSocket` auf Clients, die sich verbinden.

Parameters:

port - Der Server Port auf dem der Server serviert.
verbose - Wenn verbose==true, wird Ausgabe erzeugt.

Methods

fail

```
public static void fail(Exception e,
                        String msg)
```
Exit with an error message, when an exception occurs.

logMessage

```
protected void logMessage(Message msg)
```
log all Messages to be able to initiate new client!

main

```
public static void main(String args[])
```
Start the server up, listening on an optionally specified port

run

```
public void run()
```
The body of the server thread. Loop forever, listening for and accepting connections from clients. For each connection, create a Connection object to handle communication through the new Socket. When we create a new connection, add it to the Vector of connections. Note that we are running asynchronously.
We used to use synchronized to lock the Vector of connections. The Vulture class does the

same, so the vulture won't be removing dead connections while we're adding fresh ones. This version seems more resistant to deadlock.

Overrides:

run in class Thread

Class Hierarchy

- class java.lang.Object
  - class como.Board (implements java.io.Serializable)
  - class como.BoardMessageType (implements java.io.Serializable)
  - class como.BoardToolMessageType (implements java.io.Serializable)
  - class como.CardShape (implements java.io.Serializable)
  - class como.ChatMessageType (implements java.io.Serializable)
  - class como.ClientMessageType (implements java.io.Serializable)
  - class como.LineType (implements java.io.Serializable)
  - class como.ListOfPersons (implements java.io.Serializable)
    - class como.ListOfSpeakers (implements java.io.Serializable)
  - class como.ListOfPersonsMessageType (implements java.io.Serializable)
  - class como.ListOfPersonsToolMessageType (implements java.io.Serializable)
  - class como.ListOfSpeakersMessageType (implements java.io.Serializable)
  - class como.ListOfSpeakersToolMessageType (implements java.io.Serializable)
  - class como.Message (implements java.io.Serializable)
  - class como.MyText (implements java.io.Serializable)
  - class java.util.Observable
    - class como.BoardTool (implements java.util.Observer)
    - class como.ListOfPersonsTool (implements java.util.Observer)
    - class como.ListOfSpeakersTool (implements java.util.Observer)
    - class como.NetworkTool (implements java.lang.Runnable)
    - class como.PresentationObjectTool (implements java.util.Observer)
  - class como.Person (implements java.io.Serializable)
  - class como.PresentationObject (implements java.io.Serializable)
    - class como.Card (implements java.io.Serializable)
    - class como.FreeText (implements java.io.Serializable)
    - class como.Graphic (implements java.io.Serializable)
    - class como.Line (implements java.io.Serializable)
      - class como.Connection (implements java.io.Serializable)
  - class como.PresentationObjectMessageType (implements java.io.Serializable)
  - class como.PresentationRights (implements java.io.Serializable)
  - class como.PresentationType (implements java.io.Serializable)
  - class como.TextSize (implements java.io.Serializable)
  - class java.lang.Thread (implements java.lang.Runnable)
    - class como.NConnection
    - class como.Vulture
    - class como.presentationToolServer
  - class como.VersionMessageType (implements java.io.Serializable)
  - class como.VersionTool