Konzept eines mehrbenutzerfähigen Multimedia-Editors

Erik Wilde, ETH Zürich

1 Vorbemerkungen....1
2 Design-Ziele....1
3 Interface Style....2
4 Funktionalität....4
4.1 Editorfunktionalität 4
4.2 Kombinierte Editor- und Konferenzsystem-funktionalität 4
5 Dokumentenmodell....4
5.1 Das ODA-Modell 5
5.1.1 Hyper-ODA 5
5.1.2 "Gerloffsche" Synchronisation 6
5.2 Alternativen zu ODA 6
6 Mehrbenutzerfähigkeit....7
6.1 Funktionalität 7
6.2 Architektur 8
6.2.1 Der Editor-Client 11
6.2.2 Der Editor-Server 13
6.2.3 Das Editor-Protokoll 14
6.2.4 Das Editor-Konferenz-Protokoll 16
6.3 Gesamtarchitektur 16
7 Multimediafähigkeiten....17
7.1 Grundfunktionalität 18
7.2 Erweiterte Funktionalität 19
8 Präsentationsaspekte....20
8.1 Benutzeroberfläche 20
8.2 Dynamische Präsentationen 20
9 Integration in das Konferenzsystem....20
9.1 Aufbau von MultimETH 21
9.2 Zwei Associations 22
9.3 Eine Association 23
9.3.1 Aufbau des Client 23
9.3.2 Aufbau des Server 24
9.3.3 Isolierte E-Clients 24
10 Das Andrew Toolkit (ATK)....24
10.1 Multimediafähigkeiten des ATK 24
10.2 Dokumentenmodell des ATK 26
10.3 ATK und ODA 26
11 Literaturverzeichnis....26
12 Glossar....28
13 Index....31

1 Vorbemerkungen

Das vorliegende Papier stellt zunächst nicht mehr als eine Diskussionsgrundlage dar, in welcher Weise ein mehrbenutzerfähiger Multimedia-Editor für den in [Lubich 90] beschriebenen MultimETH-Prototypen LittlemETH entworfen werden kann. Jegliche Anregungen, Aenderungen oder Erweiterungen vorzunehmen, sind daher willkommen, je mehr Meinungen zu den hier beschriebenen Konzepten geäussert werden, desto eher wird eine halbwegs stabile Grundlage entstehen, auf deren Basis anschliessend die Implementierung aufbauen kann.

Insbesondere die Strukturierung des Papiers ist im derzeitigen Zustand stark verbesserungsfähig, da durch das häufige Einfügen, Verschieben und Löschen von Textteilen der Zusammenhang der einzelnen Teile Gefahr läuft, verloren zu gehen. Besonders hinsichtlich dieses Aspektes bin ich also für jeden Hinweis dankbar, in welcher Weise Aenderungen sinnvoll wären. Ebenfalls nur durch andere Leser kann festgestellt werden, ob Glossar und Index brauchbar sind, also auch zu diesen Abschnitten bin ich für jede (konstruktive) Bemerkung dankbar.

2 Design-Ziele

In diesem Kapitel sollen die Ziele vorgegeben werden, die bei der Gestaltung des Editors verfolgt werden sollen. Während einige der Ziele leichter zu erreichen sein werden, sind andere nur schwer oder nur mit Einschränkungen zu realisieren. Das Design des Editors soll so vorgenommen werden, dass eine möglichst grosse Zahl von Vorgaben vollständig befriedigt werden kann. Eventuelle Einschränkungen müssen durch nicht zu umgehende Probleme erklärt werden können.

– Unabhängigkeit von MultimETH

  Den Rahmen der Editorgestaltung stellt das MultimETH-Projekt dar, dessen Konferenzsystemfähigkeiten um einen verteilten Editor erweitert werden sollen. Das Design des Editors sollte jedoch so vorgenommen werden, dass eine Verwendung als eigenständige Applikation ebenfalls möglich ist. Dies ist dann der Fall, wenn sowohl der Programmcode als auch das Protokoll eine klare Trennung zu MultimETH aufweisen.

– Erweiterbarkeit

  Der Editor sollte nicht als monolithisches Programm, sondern klar modularisiert aufgebaut werden, damit eine einfache Erweiterbarkeit des Codes gewährleistet ist. Dies betrifft sowohl die Funktionalität des Editors (d.h. die dem Benutzer angebotene Unterstützung des Mehrbenutzerbetriebes) als auch die Menge der angebotenen Datentypen.

Diese Liste der Designvorgaben ist nicht vollständig und muss sicher um wichtige Ziele ergänzt werden.

3 Interface Style

Das User Interface (UI) des Editors muss konsistenten Bedingungen für die Gestaltung von Benutzeroberflächen gehorchen, zudem sollte es durch Verwendung eines bekannten *Look and Feel* die Benutzung erleichtern. Als de-facto-Standard für UIs setzt sich zunehmend *OSF/Motif* von der Open Software Foundation durch. Da für die nächste Version des *Andrew Toolkit,* auf dem der Editor aufbauen wird, ebenfalls eine Motif-Oberfläche angekündigt ist, wird das UI gemäss dieser Festlegung gestaltet werden. Die Programmierung eines UI in Andrew wird relativ abstrakt vorgenommen, so dass bei der Verwendung eines anderen Look and Feel durch das ATK die Applikation selbst gar nicht verändert werden muss, sondern lediglich eine erneute Uebersetzung mit der neuen Version des Toolkits nötig ist.

Motif ist eine Bibliothek von Routinen, die auf dem *X Window System* aufbaut und die Programmierung von UIs erleichtert. In der Architektur und Terminologie von X ist Motif ein *Widget Set* des *Xt Toolkit* (aufbauend auf der *Xlib* und einer erweiterten Version der *Xt intrinsics*), das dem Programmierer ermöglicht, von vielen Details der X Programmierung zu abstrahieren und eine "objektorientierte" Schnittstelle anbietet.[1] In [OSF 91] sind Richtlinien (*Compliance Conventions*) festgelegt, denen eine Applikation gehorchen muss oder sollte, um *Style Guide Compliant* zu sein, also den Gestaltungsrichtlinien für Motif UIs zu entsprechen. Die Grundsätze des Style Guide sind die folgenden:

– Berücksichtigung der Benutzerperspektive

  Die Sicht der Benutzer auf eine Applikation muss als Grundlage der Gestaltung des UI dienen. Während ein Programmierer eine Applikation vorwiegend als Sammlung von Funktionen sieht, die dem Benutzer angeboten werden, ist für die Anwendung der Applikation die Präsentation der Funktionen wichtig. Aus diesem Grund sollte zur Gestaltung des UI eine Benutzergruppe herangezogen werden, zumindest jedoch sollte der Programmierer die eigene Applikation anwenden.

– Der Benutzer soll die Kontrolle über die Applikation haben

  Zwei Aspekte sind in diesem Zusammenhang zu bemerken. Zum Einen sollte das UI der Applikation konfigurierbar sein, d.h. der Benutzer sollte in der Lage sein, die Oberfläche seinen Bedürfnissen anzupassen. Nur so kann verhindert werden, dass eine zu starre Festlegung des UI die Benutzung der Applikation für einige Benutzer erschwert. Zudem sollten Funktionen auf unterschiedliche Art ausführbar sein (Tastenkombination, Menüselektion, direkte Manipulation eines Objektes). Zum Anderen sollten die angebotenen Funktionen hinsichtlich ihrer Bedeutung unterschiedlich angeordnet werden. Häufig benutzte Funktionen müssen leicht auszuführen sein, während weniger oft verwendete Funktionen nicht den Zugriff auf die häufigen Funktionen erschweren sollten.

– Verwendung von "Real-World Metaphors"

  Der Benutzer sollte in die Lage versetzt werden, Fähigkeiten, die er beim Umgang mit Objekten der "realen Welt" erlangt hat, mit Hilfe des UI umsetzen zu können. Dazu gehört, dass direkte Manipulationen von Objekten zugelassen sind, d.h. nach der Selektion eines oder mehrerer Ob-

[1] Neben den Bibliotheken, die für die Programmierung von Applikationen verwendet werden, ist ebenso ein Window Manager Teil von Motif. Dieser ist jedoch in diesem Zusammenhang weniger von Interesse, er dient dazu, Benutzerinteraktionen so zu steuern, dass mit Motif erstellte Applikationen optimal unterstützt werden. [Berlage 91] vermittelt eine umfassende Darstellung der Konzepte von Motif sowie der Software.

jekte können Kontrollelemente verwendet werden, um die Selektion zu manipulieren. Um die Vorstellung der direkten Manipulation realisieren zu können, muss die Applikation kurze Antwortzeiten bieten, da ansonsten der Eindruck unmittelbarer Kontrolle verloren geht. Schliesslich muss es möglich sein, Ausgaben bestimmter Komponenten als Eingaben anderer Komponenten zu verwenden, um das Mass an einzugebenden Informationen möglichst gering zu halten.

– Gestaltung intuitiver (natürlicher) UIs

  Die Handhabung eines UI sollte so gestaltet sein, dass der Benutzer grosse Teile intuitiv verwenden kann. Dazu gehört, dass die Distanzen zwischen logisch zusammengehörenden Einheiten möglichst gering gehalten werden, so dass mit der Maus nur kurze Wege zurückgelegt werden müssen, um diese Funktionen zu bedienen. Jeder Weg kompliziert und verlangsamt die Bedienung der Applikation und stört damit die Konzentration des Benutzers auf die eigentliche Aufgabe. Ebenfalls sollte beachtet werden, wichtige Objekte durch starke Kontraste zur Umgebung, auffällige Farbgebung sowie einprägsame Formen hervorzuheben, wobei diese Art der Hervorhebung nicht zu oft angewendet werden darf, um den Benutzer nicht abzulenken.

– Gestaltung konsistenter UIs

  Dieser Aspekt ist das Hauptanliegen des Style Guide, da die Konsistenz eines UI sowie verschiedener UIs untereinander massgeblich dazu beiträgt, den Aufwand zum Erlernen der Bedienung einer Applikation erheblich zu verringern. Zwei Arten der Konsistenz sind zu unterscheiden:

  – *Intraapplication Consistency* (Konsistenz innerhalb einer Applikation)

    Aehnliche Komponenten des UI sollen ähnliches Verhalten zeigen und ähnlich zu benutzen sein. Der Benutzer erwartet, dass die Darstellungen in einer Beziehung zur Fuktion stehen. Die gleiche Aktion sollte immer das gleiche Ergebnis haben, die Funktion von Komponenten sollte sich also nicht kontextabhängig ändern.

  – *Interapplication Consistency* (Konsistenz zwischen Applikationen)

    Das Layout der Komponenten sollte so gewählt sein, dass es dem Benutzer vertraut erscheint. Die Interaktion mit dem UI (Komponentenverhalten, Eingabeverhalten, Selektionen) sollte gemeinsamen Prinzipien gehorchen. Zudem sollte die Anordnung der Komponenten in vertrauter Art geschehen, z.B. hinsichtlich des Aufbaus von Menues oder Eingabe- bzw. Auswahlmasken.

– Information des Benutzers über Aktionen der Applikation

  Eine Applikation sollte den Benuzter über die Aktionen, die ausgeführt werden, informieren. Eingaben oder Selektionen sollten z.B. bestätigt werden, zudem sollte bei längerer Bearbeitung eine Mitteilung erscheinen, dass die Applikation momentan aktiv ist (häufig wird für solche Anzeigen eine veränderte Darstellung des Cursors gewählt). Fehler des Benutzers sollten – soweit möglich – vorhergesehen und durch entsprechende Aktionen (ausschalten von Menuepunkten, Undo-Funktion) berücksichtigt werden. Das Löschen von Ovjekten sollte explizit vom Benutzer bestätigt werden, um unbeabsichtigte Löschungen zu vermeiden.

– Vermeidung verbreiteter Unzulänglichkeiten von UIs

  Verbreitete Unzulänglichkeiten von UIs sollten vermieden werden, indem auch unwichtig erscheinende Details beachtet werden, die Gestaltung des UI iterativ durchgeführt wird (der erste Ansatz ist selten der Beste), wenn möglich in Zusammenarbeit mit späteren Benutzern, sowie Implementierungsdetails vor dem Benutzer verborgen werden, der sich rein auf die Fuktionalität des UI konzentrieren soll.,

Diese Prinzipien des Style Guide gelten nicht nur für die Gestaltung von Motif UIs, sondern sollten im allgemeinen beim Design der Benutzerschnittstelle interaktiver Applikationen berücksichtigt werden. Der Style Guide setzt diese abstrakten Prinzipien in konkrete Anforderungen (*requirements*) und Empfehlungen (*recommendations*) um, in denen beschrieben wird, welche Aktionen einer Applikation mit welchen Elementen des Motif Toolkits unterstützt werden müssen bzw. sollten.

Das bisherige UI des MultimETH-Prototyps wurde zuerst zeichenbasiert (VT100-Schnittstelle), daran anschliessend als graphische Oberfläche mit Hilfe des SunView Window Systems von Sun realisiert. Die SunView Implementierung benutzt einige Elemente der graphischen Oberfläche (so zum Beispiel *Pull-down Menus* und *Cascading Menus*), die nicht mit jedem Window System bzw. jedem Toolkit des X Window Systems verfügbar sind. Bei der Implementierung mit dem Andrew Toolkit muss möglicherweise auf einige Eigenschaften einer Motif-Oberfläche, die langfristig angestrebt wird und diese Möglichkeiten ebenfalls bietet, verzichtet werden, da das ATK nicht alle Elemente der Motif-

Oberfläche zur Verfügung stellt, sondern lediglich eine Untermenge.[2] Auf diese Weise kann sichergestellt werden, dass der Uebergang zu einem Motif UI ohne Aenderungen der ATK Applikationen möglich ist, da sich für jedes Menue-Element des ATK eine entsprechende Abbildung auf ein Widget des Motif UI finden und somit darstellen lässt. Auf einige Möglichkeiten des Motif Toolkit muss jedoch verzichtet werden, da es für sie keine korrespondierenden Elemente innerhalb des ATK gibt. In welcher Weise die fehlenden Elemente ersetzt werden können, muss noch untersucht werden.

Im Prinzip sind zwei Wege möglich, die fehlenden Elemente zu ersetzen. Erste Variante ist ein Redesign der Benutzer-Schnittstelle von MultimETH. Wird die Schnittstelle so gestaltet, dass die im ATK fehlenden Elemente nicht benötigt werden, können ausschliesslich die vom Toolkit angebotenen Möglichkeiten verwendet werden. Nachteil ist bei diesem Vorgehen, dass – zur Bewahrung einer konsistenten Schnittstelle – auch die SunView Implementierung angepasst werden müsste und es notwendig wäre, sich auf eine relativ kleine Menge von UI Gestaltungsmöglichkeiten zu beschränken. Zudem ist die Benutzerschnittstelle von MultimETH auf Grund längerer Ueberlegungen entstanden und sollte nicht ohne Notwendigkeit völlig neu gestaltet werden. Vorteil dieses Vorgehens wäre allerdings die Tatsache, dass ausschliesslich ATK-konforme UI-Elemente verwendet würden und deshalb, z.B. bei der Bereitstellung einer Motif-Oberfläche durch eine neue Andrew Version, nur eine erneute Uebersetzung notwendig würde, ohne den Code ändern zu müssen.

Die zweite Alternative, die fehlenden Elemente zu ersetzen, wäre, diese selbst zu implementieren. Die Gestaltung von UIs wird im ATK mit Hilfe von vorgegebenen Klassen vorgenommen. Eine Erweiterung der Funktionalität dieser Klassen böte die Möglichkeit, neue UI-Elemente anzubieten und diese dann wie die durch das ATK vordefinierten verwenden zu können. Für die aktuelle Version von Andrew ist dieses Vorgehen sicher eine akzeptable Lösung, jedoch ist zum Einen der Aufwand einer solchen Implementierung unklar, zum Anderen ist fraglich, in welcher Weise sich dieses Vorgehen mit einer neuen Version des ATK vertrüge, die eine Motif-Oberfläche anbieten soll. Im schlimmsten Fall muss in diesem Fall die gesamte Implementierung neu vorgenommen werden, was einen erheblichen Aufwand darstellen kann.

4 Funktionalität

Eine der grundlegenden Entscheidungen bei dem Entwurf eines interaktiven Programmes ist die Festlegung der Funktionalität, d.h. der Möglichkeiten, die dem Benutzer angeboten werden. Im Fall des MultimETH Editors kommen diesem Aspekt in zweierlei Hinsicht besondere Bedeutung zu. Zum Einen muss entschieden werden, in welcher Weise die Mehrbenutzerfähigkeit unterstützt wird, andererseits ist es ebenfalls notwendig, festzulegen, wie die Multimediafähigkeiten aussehen sollen.

4.1 Editorfunktionalität

4.2 Kombinierte Editor- und Konferenzsystemfunktionalität

5 Dokumentenmodell

Als Vorschlag einer Dokumentenarchitektur für das Konferenzsystem wurde bereits in der Definition des Modells und der Funktionalität des MultimETH Systems ein Dokumentenmodell beschrieben, das sich an ODA anlehnt, jedoch nur eine recht kleine Untermenge dieses Standards implementiert. Ob dieses Modell beibehalten wird, ist noch nicht festgelegt, da jedoch die bisherige Implementierung des Prototyps keine Anwendung dieses Modells enthält, liesse sich eine Aenderung ohne Folgen für die bestehenden Programme durchführen. Einzig das Protokoll müsste geändert werden, da dieses schon die Definition der Dokumentenarchitektur enthält. Das vorgeschlagene Dokumentenmodell hat folgende Definition:

```
<Dokument>              =    [<Titeleintrag>] [<Abstract>] [<Inhaltsverzeichnis>]
                             (<Kapitel>)* (<Anhang>)* [<Literaturverzeichnis>]
```

[2] Fehlende Elemente in den Möglichkeiten des ATK, Menues zu verwenden sind z.B Cascading Menus, die dazu verwendet werden können, innerhalb von Pulldown Menus, die – neben *Popup Menus* – die normale Art von Motif Menues darstellen, bei einem Menuepunkt eine weitere Auswahl zu bieten, indem bei der Selektion des *Cascade Button* ein weiteres Pulldown Menu angezeigt wird. Ebenfalls nicht Element des ATK ist die Möglichkeit, nicht anwählbare Menuepunkte darzustellen, um die Präsentation der Menues konsistent zu halten, ohne nicht ausführbare Funktionen anwählbar zu lassen.

```
<Titeleintrag>          =    <Titel> (<Autor>)+
<Titel>                 =    <Inhalt>
<Autor>                 =    <Aufzählung>
<Abstract>              =    <Aufzählung>
                        =    [<Titel>] (<Abschnitt>)+
<Abschnitt>             =    <Inhalt> | <Kapitel> | <Fussnote>
<Aufzählung>            =    <Inhalt> <Aufzählung> | <Inhalt>
<Inhaltsverzeichnis>    =    TEXT
<Anhang>                =    <Kapitel>
<Literaturverzeichnis>  =    TEXT
<Fussnote>              =    <Aufzählung>
<Inhalt>                =    TEXT | IMAGE | GRAFIK | SPREADSHEET | SPRACHE | CHALKBOARD
 (wo ist <Kapitel>? die ASN.1 definition sagt: <Kapitel> = <Titel> (Abschnitt)*)
 (was meint Zeile 7?)
 (<Inhalt> = (...)+? definiert als SET{CHOICE{...}}, nicht eher CHOICE{...}?)
 (möglich ist Kapitel = <Inhalt> (<Kapitel>) <Inhalt>. macht keinen sinn)
 (die definition für TEXT (IA5Text) verunmöglicht formatierten text; troff oder TeX?)
 (wenn inhalts- und literaturverzeichnis, warum nicht abbildungsv. oder index oder glossar?)
 (nicht eher <Aufzählung> = (<Inhalt>)*? entspräche ASN.1 und ist strukturell einfacher)
 (nicht eher <Abschnitt> = <Aufzählung> | <Kapitel> | <Fussnote>?)
 (oder noch eher <Abschnitt> = <Aufzählung> | <Kapitel>? und Fussnote im Text?)
 (die definition von <Titeleintrag> erscheint mir etwas seltsam)
```

Diese Definition ist insofern unvollständig, als z.B. die Inhaltsarchitekturen nicht aufgeführt sind, es also nicht möglich, eine Aussage darüber zu treffen, in welcher Weise die angegebenen Inhaltstypen unterstützt werden.[3]

5.1 Das ODA-Modell

Das in [ISO 8613] definierte ODA-Modell legt eine Dokumentenarchitektur für baumartig strukturierte Dokumente fest, die aus Elementen verschiedener Inhaltstypen aufgebaut werden können. Es ist nicht der Sinn dieses Papiers, eine Beschreibung von ODA zu geben, aus diesem Grund werden hier nur kurz die wichtigsten Aspekte dieser Dokumentenarchitektur erwähnt, um sie hinsichtlich der Verwendung für den Editor bewerten zu können. Eine ausführliche Beschreibung des ODA-Standards ist in [Appelt 90] enthalten, dort finden sich sowohl die zugrundeliegenden Konzepte als auch die Details des Standards beschrieben.

5.1.1 Hyper-ODA

Hyper-ODA als Weiterentwicklung von ODA versucht, die baumförmige Topologie eines herkömmlichen ODA-Dokumentes durch die Netztopologie (ein allgemeiner, keinerlei Einschränkungen unterworfener Graph, der z.B. auch Zyklen enthalten kann) eines Hypertextdokumentes zu ersetzen. Durch diese strukturelle Erweiterung des Dokumentenmodells liesse sich die Menge der repräsentierbaren Dokumente stark vergrössern. Die Vorteile eines Hypertext-Systemes sind in [Conklin 87] folgendermassen beschrieben:

- Hypertext ist eine Datenbankmethode, d.h. ein bestimmter Weg, um Informationen zu verwenden, die in *nodes* enthalten sind und auf die mit *links* verwiesen wird .
- Hypertext ist ein Repräsentationsschema, d.h. eine Technik, mit deren Hilfe Information strukturiert beschrieben und gespeichert werden kann.

[3] Für die Typen Text, Image und Grafik bestände die Möglichkeit, die in ODA definierte Inhaltsarchitektur zu verwenden, doch sind Inhaltsarchitekturen für die anderen Inhaltstypen nicht in ODA definiert, so dass hier eine eigene Definition notwendig ist. Zukünftige Versionen des Standards werden weitere Inhaltstypen definieren, so z.B. Audio, in der aktuellen Version sind solche Festlegungen jedoch noch nicht enthalten.

– Hypertext ist eine Art der Informationspräsentation, das UI wird die Art der Repräsentation i.A. wiederspiegeln durch die Möglichkeit, *nodes* anzuzeigen und *links* mit Hilfe unmittelbarer Manipulationsmöglichkeiten zu verfolgen.

5.1.2 "Gerloffsche" Synchronisation

In [Gerloff 88] wird ein Modell zu Synchronisation von Präsentationen innerhalb von ODA-Dokumenten vorgeschlagen, das das Dokumentenmodell um Eigenschaften erweitert, die notwendig sind, um zeitabhängige Abläufe darstellen zu können.

5.2 Alternativen zu ODA

Auf Grund der Verwendung von Andrew liegt es nahe, als internes, für die Implementierung verwendetes Dokumentenmodell nicht ODA zu verwenden (das ohnehin keine Repräsentation festlegt, abgesehen von den Austauschformaten), sondern ein Dokumentenmodell, das die vordefinierten Andrew-Klassen einschliesst und es ermöglicht, Instanzen dieser Klassen zu Dokumenten zu kombinieren. Jedem Objekt sollte eine Menge von Verwaltungsinformationen zugeordnet werden können, die dazu verwendet werden, Aenderungen und Eigentums- und Zugriffsrechte zu vermerken und darauf basierend Authentifizierungsprüfungen vorzunehmen. Diese Informationen sollten folgende Felder umfassen.

– *Owner* (Eigentümer)

  Jedes Objekt hat einen Eigentümer, der initial durch den Benutzer definiert ist, der das entsprechende Objekt kreiert hat. Der Eigentümer eines Objektes hat die meisten Zugriffsrechte (es sein denn, er schränkt sie explizit ein), er kann anderen Teilnehmern einzeln oder nach Gruppen bzw. Konferenzen gegliedert einzelne Rechte einräumen oder entziehen.

  Es gibt zwei spezielle Arten von Eigentümern. Der Eigentümer *Predescessor* besagt, dass dieses Objekt keinen explizit angegebenen Eigentümer hat, sondern dieser sich implizit durch den Eigentümer des in der Objekthierarchie darüber liegenden Objektes ergibt. Dieser Prozess kann rekursiv fortgesetzt werden. Das in der Hierarchie an höchster Stelle liegende Objekt darf diesen Attributwert nicht verwenden, da ansonsten kein Eigentümer feststellbar ist. Für diesen Fall existiert der spezielle Eigentümer *None*, der besagt, dass das entsprechende Objekt keinen Eigentümer hat, also als öffentlich angesehen werden kann.

– *Creation Date* (Erzeugungsdatum)

  Das Erzeugungsdatum eines Objektes besteht aus Datum und Zeit der Erzeugung des Objektes sowie der Identifikation des Benutzers, der dieses Objekt erzeugt hat. Dieses Attribut kann nicht modifiziert werden, d.h. es hat während der gesamten Lebenszeit des Objektes einen festen Wert.

– *Access Rights* (Zugriffsrechte)

  Die Zugriffsrechte eines jeden Objektes legen fest, welchen Zugriff einzelne Benutzer auf ein Objekt haben. Es gibt die Zugriffsrechte *Read*, *Annotate* und *Write*. Ohne Zugriffsrechte ist dem Benutzer keinerlei Zugriff auf den Inhalt oder die Attribute eines Objektes erlaubt. Dadurch kann auch die Struktur des Dokumentes an dieser Stelle nicht weiter verfolgt werden. Das Read Zugriffsrecht erlaubt das Lesen eines Objektes (sowohl des Inhalts als auch der Attribute), ebenso können in der Hierarchie unter diesem Objekt liegende Objekte gesehen werden. Das Annotate Zugriffsrecht umschliesst das Read Zugriffsrecht sowie die Möglichkeit, Bemerkungen zu diesem Objekt zu schreiben (d.h. das Annotate Attribut zu erweitern). Das Write Zugriffsrecht erlaubt das Lesen und Schreiben sämtlicher Attribute (mit Ausnahme der automatisch modifizierten Attribute Change Log und Reservation, des unveränderlichen Attributs Creation Date sowie der Attribute Owner und Access Rights, das nur durch den Eigentümer selbst geändert werden darf).

  Diese Zugriffsrechte können unterschiedlichen Benutzerkreisen zugeordnet werden. Es gibt die Werte *Owner*, *Group*, *Conference* und *World*. Der Owner ist der Eigentümer des Objektes, der immer (auf bei nicht oder auf Read gesetztem Zugriffsrecht) das Attribut Access Rights modifizieren kann. Eine Group ist eine definierten Menge von Benutzern, die mit Hilfe einer Gruppenverwaltung definiert wird. Eine Conference ist eine durch das Konferenzsystem definierte Menge von Benutzern, die eine Ueber- oder Untermenge einer Group sein kann, ebenso sind Ueberschneidungen möglich. Alle Benutzer sind schliesslich unter World zusammengefasst, d.h. ein diesem Benutzerkreis zugeordnetes Zugriffsrecht kann von allen Benutzern wahrgenommen werden.

– *Change Log* (Liste der Veränderungen)

  Eine Liste der Veränderungen eines Objektes wird automatisch mitgeführt, sie enthält Datum, Zeit und Identifikation des Benutzers einer jeden Aenderung, die am Objekt vorgenommen wurde.

Veränderungen werden auf Grund von Reservierungen vorgenommen, d.h. es wird gespeichert, welcher Benutzer das Objekt zu welchem Zeitpunkt reserviert hat, sofern Aenderungen vorgenommen wurden. Das Attribut Change Log hängt aus diesem Grund vom Attribut Reservation ab, indem es die dort vorgenommen Aenderungen speichert.

– *Reservation* (Reservierungen)

  Im Attribut Reservation wird festgehalten, ob das jeweilige Objekt von einem Benutzer für die Durchführung von Aenderungen reserviert wurde. Ist dies der Fall, wird der Benutzername in diesem Attribut gespeichert und das Objekt ist für weitere Reservierungen gesperrt, ansonsten ist dieses Attribut leer.

– *Annotation* (Bemerkungen)

  Dieses Attribut enthält eine Liste von Paaren aus Texten und Benutzeridentifikationen, die Bemerkungen zu dem betreffenden Objekt darstellen. Eine Bemerkung kann von jedem Benutzer gespeichert werden, der mindestens das Zugriffsrecht Annotate hat, auf diese Weise ist es möglich, Bemerkungen zu Objekten zu schreiben, ohne den Inhalt des Dokumentes zu ändern.

Zusätzlich zu diesen Attributen, die der Dokumentenverwaltung dienen, hat ein Objekt weitere Attribute, die für die Verwendung innerhalb des Dokumentes wichtig sind. Dies sind zum grössten Teil Objekt-spezifische Attribute, doch gibt es ebenso Attribute, die allen Objekten gemeinsam sind (wie z.B. das Inhalts-Attribut).

6 Mehrbenutzerfähigkeit

Heutige Editoren (oder, allgemeiner, Dokumentenverarbeitungssysteme) bieten oft bereits viele Möglichkeiten, verschiedene Informationstypen (im allgemeinen Texte, Graphiken und Rasterbilder) innerhalb eines Dokumentes zu verwenden und zu editieren und können damit als Multimedia-Editoren angesehen werden. Wie weit diese Intergration verschiedener Informationstypen tatsächlich geht, z.B. hinsichtlich der angebotenen Bearbeitungsmöglichkeiten und hinsichtlich der möglichen Kombinationen verschiedener Typen (Text in Graphiken, Graphiken in Tabellen usw.), ist sehr verschieden, ändert aber nichts am Prinzip der Verwendung unterschiedlicher Informationstypen.

Weitaus seltener im Bereich der implementierten Systeme ist die Unterstützung für den Mehrbenutzerbetrieb, d.h. die vom Editor zugelassene und koordinierte Möglichkeit, mehrere (voneinander entfernte) Benutzer gleichzeitig an einem Dokument editieren zu lassen und die Zugriffe auf das Dokument und seine Teile gemäss u.U. veränderbaren Strategien so zu regeln, dass eine sinnvolle Gruppenarbeit ermöglicht wird. Das Bild der Informationsbe- und -verarbeitung ist noch stark vom einzelnen Benutzer geprägt, der isoliert an seinem Arbeitsplatzrechner arbeitet und alleine an "seinem" Dokument editiert, das dann, falls Gruppenarbeit durchgeführt werden soll, mit Hilfe elektronischer Post versendet werden kann.[4] Der Bereich des Mehrbenutzerbetriebes von Editoren ist demnach noch stark in der Entwicklung begriffen und deshalb gibt es auch noch keine stabilen Uebereinkünfte, wie die Funktionalität auf diesem Gebiet aussehen soll, wie das z.B. auf dem Gebiet der Informationstypen der Fall ist (z.B. sind die grundlegenden Operationen, die ein Texteditor bereitstellen soll, relativ bekannt und werden auch von nahezu allen Systemen in verschiedenen Formen zur Verfügung gestellt). Aus diesem Grund wird im folgenden Abschnitt der Aspekt der Funktionalität detailliert beschrieben.

Die Fähigkeit zum Mehrbenutzerbetrieb muss bei der Gestaltung des Aufbaus des Editors berücksichtigt werden, da die Kommunikation zwischen den verschiedenen Teilnehmern innerhalb des Editors gesteuert und ausgeführt werden muss. Es gibt unterschiedliche Möglichkeiten, dieses Ziel beim Design des Editors zu erreichen. Der zweite Unterabschnitt beschäftigt sich deshalb mit den verschiedenen Varianten der Editor-Architektur.

6.1 Funktionalität

Die Funktionalität eines mehrbenutzerfähigen Editors kann in vielen verschiedenen Aspekten gegenüber einem konventionellen Ein-Benutzer-System erweitert werden. Die folgende Liste stellt lediglich eine Zusammenstellung dessen dar, was als Funktionen wünschenswert sein könnte. Sie erhebt keinerlei Anspruch auf Vollständigkeit und muss auch hinsichtlich ihrer Struktur noch überarbeitet werden. Ins-

[4] In diesem Fall handelt es sich um eine asynchrone Variante der Zusammenarbeit, da die Veränderungen ohne Koordination mit anderen von einem Benutzer alleine vorgenommen werden, der die Aenderungen anschliessend an alle anderen Teilnehmer der Gruppenarbeit verteilt. Synchrone Zusammenarbeit wird erst mit einem mehrbenutzerfähigen Editor möglich, der den gleichzeitigen Zugriff der Benutzer auf das gemeinsame Dokument koordiniert und es den Teilnehmern somit gestattet, nebenläufig am gleichen Dokument zu arbeiten.

besondere ist es wichtig, eine klare Trennung zwischen den funktionalen Möglichkeiten des Editors und denen des Konferenzsystemes zu treffen, d.h. die Funktionen, die von allgemeinem Interesse für Gruppenarbeit sind, von Editor-spezifischen Funktionen zu unterscheiden.

- Die mit am wenigsten Aufwand verbundene und einfachste Möglichkeit ist das "shared viewing", also die Fähigkeit des Editors, mehreren Benutzern die Sicht auf ein Dokument zu gestatten. Im Normalfall ist damit jedoch höchstens einem der Teilnehmer die Möglichkeit gegeben, aktiv zu sein, also Aenderungen an dem Dokument vorzunehmen. Alle übrigen Teilnehmer können lediglich sehen, was dieser privilegierte Benutzer für Aenderungen vornimmt und z.B. ihre Bemerkungen mit Hilfe des Konferenzsystemes übermitteln, nicht jedoch mit Hilfe des Editors aktiv eingreifen oder mitarbeiten.[5]
- Eine ebenfalls einfache, wenn auch schon leistungsfähigere, Variante der Mehrbenutzerfähigkeit ist die simple Forderung, dass mehrere Benutzer gleichzeitig ein Dokument bearbeiten können sollen. Dies stellt eine Erweiterung hinsichtlich der Minimalkonfiguration dar, in der die Benutzer nur Leserechte haben und lediglich einer der Teilnehmer am gemeinsamen Editiervorgang Veränderungen am Dokument vornehmen kann. Eine solche Möglichkeit sollte natürlich auch existieren, jedoch nicht die einzig mögliche Betriebsart der hier vorgeschlagenen Variante sein.

  Der grösste Nachteil dieser Vorgehensweise ist, dass die Koordination zwischen den verschiedenen Teilnehmern zwar dazu führt, dass das Dokument korrekt bzw, konsistent verändert wird, auf der anderen Seite jedoch viele unerwünschte Wechselwirkungen zwischen den Aktionen der verschiedenen Teilnehmer auftreten können. So kann z.B ein Benutzer einen gesamten Absatz löschen, in den ein Anderer gerade Text einfügt, der dann verloren geht. In diesem Fall stellen sich also die mangelhaften Steuerungsmöglichkeiten hinsichtlich des Zugriffs der Benutzer auf einzelne Teile als problematisch heraus.
- Als Verbesserung kann deshalb ein Modus vorgesehen werden, in dem das Dokument als ganzes zwar von allen Benutzern verwendet werden kann, der aber die Möglichkeit bietet, dass einzelne Benutzer Teile reservieren, so dass die oben beschriebenen unerwünschten Wechselwirkungen nicht auftreten können. Damit träte die Situation auf, das alle Benutzer das gesamte Dokument lesen können (vorbehaltlich eines ebenfalls möglichen Leseschutzmechanismus'), der schreibende Zugriff auf einen Teil des Dokumentes jedoch immer nur einem Benutzer gestattet wird. Nach dem Vornehmen der gewünschten Aenderungen kann der reservierte Teil wieder freigegeben werden und dann z.B. von einem anderen Benutzer ebenfalls modifiziert werden.

6.2 Architektur

Die Gestaltung des Editors, der Teil des MultimETH-Konferenzsystems ist, muss sich in grundlegenden Design-Entscheidungen an diesem orientieren, da nur so ein Gesamtsystem entsteht, das einem einheitlichen Entwurf entspricht. Die Architektur des Editors muss somit so entworfen werden, dass sie in Verbindung mit der MultimETH-Architektur eine sinnvolle Kombination ergibt. Im Zusammenhang mit dem Editor sind vor allen zwei Aspekte wichtig, die im folgenden aufgeführt werden

- Verwendung von OSI-Kommunikation

  MultimETH stützt sich für die Kommunikation zwischen den verteilt vorliegenden Komponenten auf Diensten des in [ISO 7498] bzw. [X.200] definierten OSI-Modells ab. Dies hat den Vorteil, dass eine international standardisierte Art der Kommunikation zwischen den Systemen innerhalb eines Rechnernetzes verwendet wird, die in der Zukunft zunehmend an Bedeutung und Unterstützung gewinnen wird. Der Nachteil ist, dass neuere Entwicklungen in der Kommunikationstechnik nicht verwendet werden können, da die Standardisierung gegenüber dem aktuellen Stand der Technik um einiges zurückliegt.[6] Im Falle von MultimETH wird der ISODE OSI-Stack ver-

---

[5]Eine solche Möglichkeit lässt sich sehr einfach mit Hilfe eines *X Multiplexers* erzielen, der dazu verwendet werden kann, auf verschiedenen, physikalisch verteilten Geräten (verschiedenen *X Servern*) die Ausgaben eines *X Clients* anzuzeigen. Der Nachteil dieses Ansatzes ist, dass X zum einen kein OSI Protokoll ist, zum anderen entweder nur Eingaben eines Benutzers möglich sind oder die Eingaben aller Benutzer miteinander vermengt werden, da das Modell von X nur einen Server und damit einen Eingabestrom für jeden Client vorsieht, und somit vom Client, dem Editor, nicht mehr korrekt den verschiedenen Benutzern zugeordnet werden können.

[6]In Zusammenhang mit verteilten Editoren wäre z.B. das *X Protocol*, wie es in [X Series 0] beschrieben ist, von Interesse, da es die Trennung von Anzeige (auf dem *X Server*) und der Applikation selbst (dem *X Client*) ermöglicht. Auf diese Weise wäre es möglich, trotz der verteilten Benutzer einen zentralen Editierprozess zu verwenden, der die verteilten Anzeigen verwaltet. Bis X oder vergleichbare Protokolle jedoch den Status eines ISO-Standards erreicht haben werden, wird noch einige Zeit vergehen, wenn es sich auch bereits zum de-facto-Standard entwickelt hat.

wendet, der verschiedene *Common Application Service Elements (CASE)* unterstützt, wie z.B. den *Association Control Service (ACS)*, den *Remote Operations Service (ROS)* oder den *Reliable Transfer Service (RTS)*. In diese Umgebung muss der Editor eingebettet werden.

– Verwendung eines zentralen Modells mit Client-Server-Architektur

  Eine der grundlegenden Entscheidungen, die bei der Gestaltung verteilter Anwendungen zu treffen sind, betrifft die Topologie der zu verteilenden Komponenten des Systems, d.h. die Anordnung der einzelnen Teile im Gesamtsystem. Die bei MultimETH gewählte Lösung verwendet eine zentrale Topologie, in der es einen Server gibt, der eine Menge von Clients bedient. Dies hat den Vorteil, dass die Menge der notwendigen Verbindungen stark reduziert wird, denn bei einer vollständigen Vermaschung ist die Anzahl der Verbindungen ungleich höher als bei einer Stern-Topologie (linearer gegenüber exponentiellem Aufwand). Nachteil eines solchen Ansatzes ist die Verwendung einer zentralen Instanz, die sowohl hinsichtlich der Verarbeitungsleistung als auch betreffs der Ausfallsicherheit zu Problemen führen kann. Der Editor muss dieser Auslegung folgen und mit Hilfe einer auf Client-Server-Prinzipien basierten Stern-Topologie aufgebaut werden.

Die beiden soeben aufgeführten Aspekte des MultimETH-Designs führen dazu, dass sich eine begrenzte Auswahl an möglichen Editor-Architekturen aufführen lässt. Nach einer Vorstellung des allgemeinen Modells, das zur Darstellung der verschiedenen Varianten dient, werden die Alternativen in Unterkapiteln behandelt.

Es wird davon ausgegangen, dass Clients sich eine lokale Kopie eines Dokumentes anfertigen, sobald dieses editiert werden soll. Bei einem Dokument im *Private Workspace (PWS)* ist dies ohnehin möglich (es kann aber auch nicht im Mehrbenutzerbetrieb editiert werden, da dies nur mit Dokumenten geschehen kann, die für mehrere Benutzer sicht- und damit zugreifbar sind), bei einem Dokument aus dem *Shared Workspace (SWS)* erfordert dies zunächst, dass sich der Editor den Inhalt des Dokumentes vom Server übermitteln lässt. Dieses Vorgehen hat den Vorteil, dass der Server bei Modifikationen des Dokumentes durch einen Client lediglich Aenderungen (Differenzen) an die anderen Clients übertragen muss, da der ursprüngliche Inhalt des Dokumentes bei diesen bekannt ist. Insgesamt erinnert diese Strategie stark an *Caching*, bei dem ebenfalls häufig benötigte Daten an einem Ort gespeichert werden, der schneller zugreifbar ist als die eigentliche Datenquelle. Diese Aehnlichkeit zeigt jedoch auch die Probleme, die mit diesem Vorgehen auftreten, denn das Konsistenzproblem ist die zentrale Frage bei allen Caching-Verfahren.

Damit ergibt sich ein Szenario, bei dem alle an einem verteilten Editierprozess teilnehmenden Editoren über eine Kopie des im Server vorliegenden Originaldokumentes verfügen und den Zugriff auf dieses so koordinieren müssen, dass eine konsistente Behandlung gewährleistet ist. Der Server verwaltet das Original des Dokumentes und gibt alle Modifikationen, die auf Grund von Aenderungen eines Clients an diesem vorgenommen werden, an alle beteiligten Clients weiter.[7] Damit entsteht die Situation, dass eine von einem Client am Dokument vorgenommene Modifikation erst dann Gültigkeit erlangt, wenn der Server dieser zugestimmt hat und sie damit in allen lokalen Kopien des Dokumentes vorgenommen werden kann. Vorher hat eine Aenderung lediglich lokalen Signifikanz, d.h. sie existiert ausschliesslich in der lokalen Dokumentenkopie des modifizierenden Clients. In diesem Zusammenhang wird die Granularität der Uebertragung der Modifikationen wichtig, d.h. die Frequenz, mit der Modifikationen vom Client an den Server übermittelt werden. Es sind verschiedene Vorgehensweisen denkbar, die sich an unterschiedlichen Kriterien orientieren. Es wird jeweils davon ausgegangen, dass der Client Aenderungen nicht sofort überträgt, sondern sie erst in einen lokalen Puffer schreibt, der periodisch geleert wird.

– zeitgesteuert

  Der Client kann die Aenderungen in einem bestimmten Zeitraster übertragen, d.h. es wird ein Timeout-Mechanismus verwendet. Die alleinige Anwendung dieser Strategie ist nicht sinnvoll, da das Volumen der uebertragenen Aenderungen sehr unterschiedlich ist und keine maximale Grösse garantiert werden kann, da der Client (zumindest theoretisch) eine beliebige Anzahl von Aenderungen in einem Zeitintervall vornehmen kann.

---

[7] Das heisst nicht, dass, wie bei einer einfachen Broadcast-Strategie, Aenderungen eines Clients automatisch vom Server an alle anderen Clients weitergeleitet werden. Meldet beispielsweise ein Client eine Aenderung an einem Dokumententeil, der von einem anderen Client für Aenderungen reserviert wurde, so weist der Server diese Aenderung zurück und der Client muss diese unzulässige Aenderung lokal rückgängig machen. Die anderen Clients nehmen diese Interaktion zwischen Client und Server überhaupt nicht wahr, da sie an ihren lokalen Kopien keinerlei Aenderungen vorzunehmen haben.

Andererseits sollte im Zusammenhang mit anderen Strategien ein Timeout-Mechanismus immer vorgesehen werden, da es sonst zu zeitlichen Verzoegerungen hinsichtlich der Aktualisierung bei dem Server und damit allen anderen Clients kommen kann, die nicht wünschenswert sind.

– volumengesteuert

  Der Puffer, der im Client verwendet wird, um Aenderungen zu speichern, die an den Server übertragen werden sollen, kann benutzt werden, um den Zeitpunkt der Uebertragung dieser Daten festzulegen. Der Füllstand dieses Puffers kann als Auslöser für die Uebermittlung dienen, damit wäre erreicht, dass die Grösse der an den Server übertragenen Aenderungen insgesamt immer die gleiche (oder zumindest nahezu die gleiche) ist.

  Nachteil dieses Verfahrens ist, dass keine maximale Wartezeit vor der Uebertragung gegeben ist, wird also nur eine Menge von Aenderungen vorgenommen, die den Puffer nicht bis zum die Uebermittlung auslösenden Stand füllt und danach nicht mehr modifiziert, so werden diese Aenderungen für einen langen Zeitraum (u.U. bis zur Beendigung der Editorsitzung) nicht weitergegeben. Andererseits muss gesichert sein, dass vor einem Ueberlaufen des Puffers dieser übermittelt wird, die volumengesteuerte Vorgehensweise sollte also – neben anderen – Anwendung finden.

– nach logischen Einheiten gegliedert

  Am meisten am Benutzer orientiert ist eine Vorgehensweise, die sich nach logischen Einheiten richtet, die vom Benutzer verändert werden. So liesse sich bei der Modifikation von Texten festlegen, dass Buchstaben, Wörter, Sätze oder Absätze übertragen werden sollen.[8] Diese informationstypspezifische Regel liesse sich dahingehend verallgemeinern, dass definierte Teile eines Objektes oder nur ganze Objekte übertragen werden. Damit erreichte man eine Orientierung an den logischen Strukturen des Dokumentes, die aus Benutzersicht für die Uebertragung wünschenswert wäre.

  Der Nachteil dieser Art der Steuerung ist, dass sowohl die Grösse der übertragenen Aenderungen als auch die bis zur Uebermittlung verstreichende Zeit sehr unterschiedlich sind in Abhängigkeit vom Verhalten des Benutzers. Aus diesem Grund sollte ein solches Vorgehen kombiniert werden mit zeit- und mit volumengesteuerten Verfahren, um hinsichtlich dieser Randbedingungen Grenzwerte festlegen zu können.

– benutzergesteuert

  Für aufwendige Arbeiten, die einen hohen Grad an Interaktivität aufweisen und die Uebertragung grosser Datenmengen zur Folge hätten, kann es sinnvoll sein, die Aenderungen asynchron (d.h. nur auf der lokalen Kopie des Clients) vorzunehmen und nur auf Anforderung des Benutzers zum Server zu übertragen. Der Benutzer könnte sich auf anderem Wege, mit Unterstützung des Konferenzsystems, mit den anderen Benutzers über den Zeitpunkt einer Uebertragung einigen.

  Da in diesem Fall die Verantwortung für die Konsistenzerhaltung zwischen den Dokumentenkopien einzig beim Benutzer liegt, ist ein solcher Modus als alleinige Steuerung problematisch. Er könnte allenfalls als Ergänzung anderer Verfahren Anwendung finden, sollte aber bei einer optimalen Gestaltung der automatischen Verfahren nicht notwendig sein.

All diese Methoden können natürlich auch miteinander kombiniert werden, d.h. es kann z.B. eine volumengesteuerte Uebertragung mit einem Timeout-Mechanismus kombiniert werden, um eine möglichst günstige Charakteristik der Uebertragung zu gewährleisten. Anzustreben ist eine möglichst flexible Steuerung, die entweder vom Benutzer wählbar ist (dies würde dem Anwender eine Auswahl aus den vorgestellten Strategien erlauben) oder, im Idealfall, automatisch an die spezifischen Bedürfnisse eines Benutzers angepasst wird, um ein optimales Verhalten[9] zu erreichen. Die Strategie kann dabei bestimmt werden durch die Aktivität (Volumen und Frequenz der Modifikationen) und die Anzahl der Benutzer sowie die aktuelle Last des Servers. Da der Server der Engpass hinsichtlich der Ko-

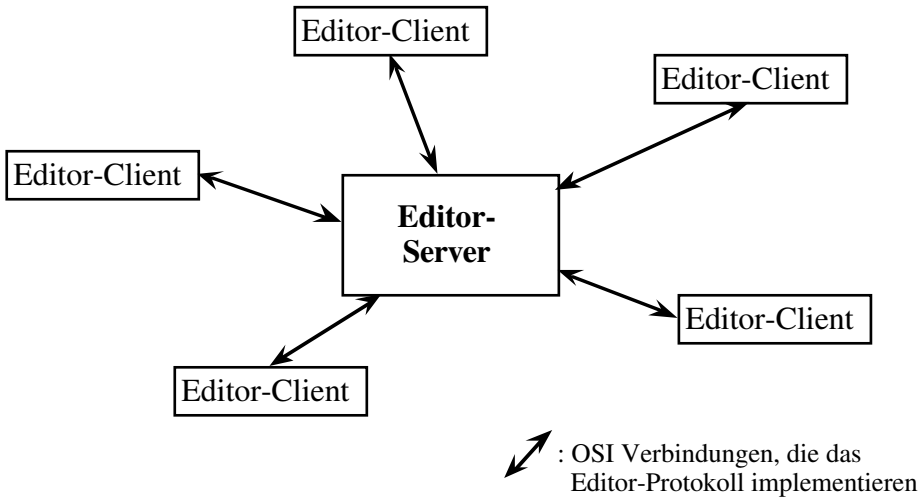
---

[8]Im Prototyp von MultimETH wird diese Vorgehensweise zur Zeit für die Uebertragung von Nachrichten im *Chat* Modus verwendet. Da die verwendeten Remote Operations auf Grund des mit ihrer Ausführung verbundenen Aufwands nicht dazu geeignet sind, Nachrichten zwischen den Clients buchstabenweise zu übertragen, werden entweder Worte oder ganze Zeilen übermittelt (in Kombination mit einem Timeout-Mechanismus).

[9]In diesem Zusammenhang bedeutet "optimal" eine möglichst günstige Relation zwischen der Anzahl der Uebertragungen (d.h. der Belastung der Komunikationsverbindung) und der Konsistenz der lokalen Dokumente (d.h. der Zeit, innerhalb der Inkonsistenzen toleriert werden). Je höher die Frequenz der Uebertragung von Aenderungen ist, desto mehr gleichen sich die lokalen Kopien, doch desto grösser wird die Netzbelastung und die Last des Servers und der Clients.

mmunikation ist, sollte dieser ebenfalls die Möglichkeit haben, die Strategie zu beeinflussen. Dazu muss in der Kommunikation zwischen Server und Clients ein Mechanismus vorgesehen werden, der eine solche Modifikation erlaubt, also z.B. eine Aushandlung der Uebertragungsstrategie ermöglicht.

Die Betrachtungen zur Uebertragungsstrategie zwischen den Clients und dem Server setzt, wie oben erwähnt, voraus, dass ein Puffer in jedem Client exisitiert, der dazu verwendet wird, die Aenderungen bis zu dem durch die angewandte Strategie bestimmten Uebertragungszeitpunkt zu speichern. In diesen Puffer werden folglich bestimmte Einträge geschrieben, die einzelne Modifikationen des Dokumentes repräsentieren. Aus diesem Grund ist es notwendig, ein Format für diese Einträge zu bestimmen. Ein solcher Eintrag wird von jeweils einem Client generiert und vom Server – nach Feststellung der Berechtigung – an die anderen am Editiervorgang beteiligten Clients verteilt, die den Eintrag interpretieren und die Aenderungen auf ihrem lokalen Datenbestand ausführen müssen. Insgesamt müssen bei einer näheren Untersuchung der Mehrbenutzerfähigkeiten also drei Aspekte betrachtet werden, die Gestaltung der Clients, die Gestaltung des Servers sowie die Art der Interaktion (hinsichtlich ausgetauschter Daten sowie hinsichtlich des Verhaltens) zwischen diesen Komponenten.



Diese Darstellung verdeutlicht die drei Hauptkomponenten der Editor-Architektur. Neben dem Editor-Server (oder E-Server) , der die Verbindungen zu allen Editor-Clients (oder E-Clients) aufrecht erhält und Koordinationsaufgaben übernimmt, bestehen die Clients, die jeweils für einen Benutzer des Editors die lokalen Aufgaben übernehmen sowie die Kommunikation mit der zentralen Instanz. Diese Kommunikation zwischen den verschiedenen Komponenten ist in einem Protokoll festgelegt, das ihr Verhalten beschreibt.

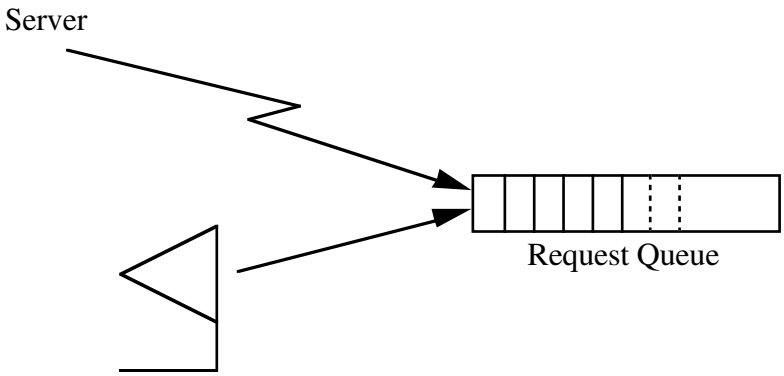
6.2.1 Der Editor-Client

Die E-Clients (im folgenden Abschnitt Clients genannt) bilden im Fall des für MultimETH zu entwickelnden Editors die Schnittstelle zum Benutzer, da jeder Teilnehmer an einem verteilten Editiervorgang innerhalb der Editor-Architektur durch einen Client repräsentiert wird (der seinerseits mit dem E-Server kommuniziert). Demzufolge lässt sich die Aufgabe des E-Server, als koordinierende Instanz in der Kommunikation mit den Clients zu wirken, so interpretieren, dass die Anforderungen der verschiedenen (durch die Clients repräsentierten) Benutzer koordiniert werden müssen. Daraus ergibt sich, dass die Clients selbst keine Verantwortung tragen, diese Koordination vorzunehmen, sondern vornehmlich dazu dienen, die Anforderungen des Benutzers an einen möglichst “ungestörten” (d.h. von fremden Einflüssen freien) Arbeitsablauf[10] mit den Koordinationsaufgaben des Server zu kombinieren und auf dieser Grundlage einen verteilten Editiervorgang zu ermöglichen.

Die Clients haben in dem Modell des verteilten Editors zweierlei Eingabeströme, einerseits vom Benutzer, der lokal am Editor arbeitet und das – lokal im Client repräsentierte – Dokument editiert, andererseits vom Kommunikationsnetz, über das dem Client vom Server die Aenderungen übertragen werden, die am zentral gehaltenen Dokument von anderen Clients vorgenommen wurden. Gegenüber einem normalen Editor ergibt sich damit das Problem der Koordination der Eingabeströme, da Situa-

[10]Ein mehrbenutzerfähiger Editor hat nat¨rlich nicht zum Ziel, die Benutzer völlig voneinander zu entkoppeln (dann könnten auch jeweils private Dokumente editiert und später zusammengefügt werden), jedoch soll der Einfluss anderer Benutzer auf die eigene Arbeit nur so stark sein wie nötig, um die Arbeit nicht unnötig zu behindern.

tionen auftreten können, in denen die Anforderungen des Benutzers und die vom Server übertragenen Informationen miteinander konkurrieren.[11] Im Client wird als Mittel zur Serialisierung der Eingaben deshalb eine *Request Queue* vorgesehen, in die die Eingaben beider Quellen, die *Edit Requests*, geschrieben werden. Der Client interpretiert diese Einträge sequentiell und kann dann auf Grund verschiedener Strategien entscheiden, in welcher Weise mit ihnen verfahren wird.

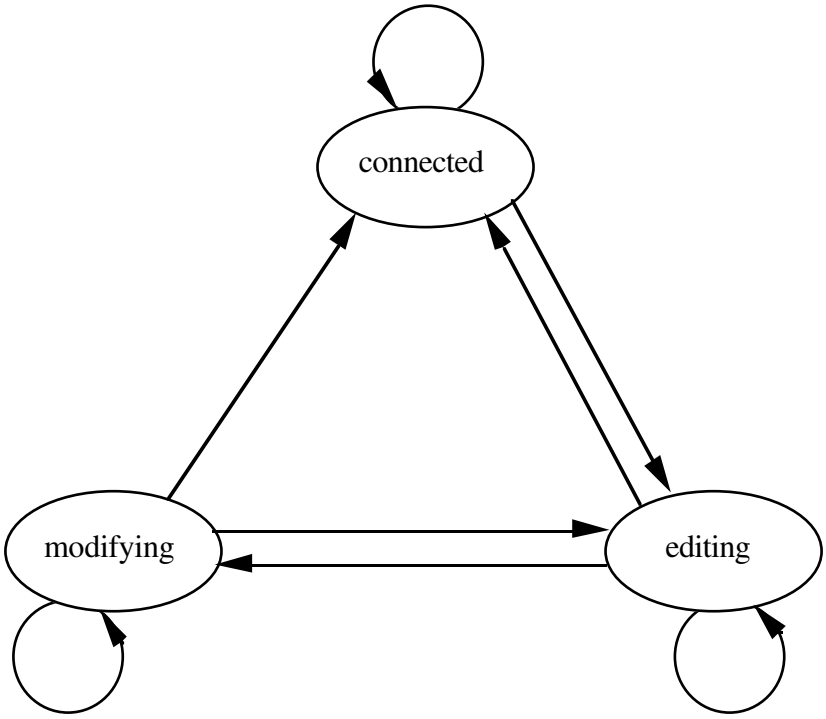


Damit ist prinzipiell festgelegt, in welcher Weise mit den Eingabeströmen von lokalem Benutzer und dem Server verfahren wird. Das Einfügen beider Arten von Eingaben in eine Queue erlaubt es, konkurrierende Situatuionen auszuschliessen. Für Operationen, die einer expliziten Bestätigung durch den Server bedürfen, wie es z.B. bei Reservierungsanfragen der Fall ist, wird nicht die lokale , druch den Benutzer ausgelöste Operation verwendet, sondern erst die Bestätigung des Server wird in die Queue eingefügt, so dass keine Konflikte auftreten können.

Die Ausgabe des lokalen Client muss ebenfalls berücksichtigen, dass es sich bei dem Dokument nur um die lokale Kopie eines zentral gehaltenen Dokumentes handelt, das zudem von verteilt angesiedelten Instanzen modifiziert wird, d.h. es können zeitweilige Inkonsistenzen zwischen den Clients auftreten, die jedoch, solange sie sich nur auf die Präsentation und nicht auf Modifikationen beziehen, akzeptiert werden können.

Der Client kann verschiedene Zustände annehmen, die jeweils bestimmend sind für die Operationen, die erlaubt und sinnvoll sind. Die folgende Darstellung zeigt die möglichen Zustände des Client sowie die durch Operationen möglichen Uebergänge. Dabei können diese Uebergänge sowohl durch vom Benutzer ausgeführte Operationen als auch durch vom Server erhaltene Operationen ausgeführt werden. Welche Operationen im einzelnen die jeweiligen Uebergänge verursachen, kann erst in Zusammenhang mit dem Protokoll zwischen E-Client und E-Server festgelegt werden, das ausserdem von dem hier vorgestellten Modell der möglichen Zustände Gebrauch machen wird, indem Vor- und Nachbedingungen für die durch das Protokoll definierten Operationen angegeben werden, die die Zustände beinhalten.

[11] Ein Beispiel für eine solche Situation wäre die Reservierung eines Dokumententeils am lokalen Client, während ein anderer Benutzer diesen Teil bereits vom Server exklusiv zugeordnet bekommen hat. In diesem Fall erhält der lokale Client zwei Eingaben, die die Reservierung dieses Teiles durch verschiedene Benutzer zur Folge haben. Diese beiden Eingaben schliessen einander aus und es muss eine Strategie für solche Fälle vorgesehen werden.



Die Zustände connected, modifying und editing können wie folgt charakterisiert werden. Im Zustand connected ist der Client mit dem Server verbunden, es wird jedoch kein Dokument editiert. Durch das Laden eines Dokumentes oder das Erzeugen eines neuen Dokumentes kann der Client in den Zustand editing übergehen, in dem sich ein Dokument in Bearbeitung befindet, jedoch keine Modifikationen möglich sind. Reserviert der Benutzer einen Teil des Dokumentes für die Modifikation, so wechselt der Zustand zu modifying, in diesem Zustand können Aenderungen an dem reservierten Teil vorgenommen werden.

Umgekehrt lassen sich die Zustandsübergänge folgendermassen beschreiben. Bricht der Benutzer die Editorsitzung ab, so kann direkt vom Zustand modifying in den Zustand connected übergegangen werden. Wird ein reservierter Teil freigegeben und ist dies der letzte reservierte Teil, so geht der Client wieder in den Zustand editing, der anzeigt, dass keine Reservation aktiv ist. Wird das so editierte Dokument dann verlassen, erfolgt der Uebergang zum Zustand connected. In allen Zuständen können des weiteren Operationen ausgeführt werden, die den Zustand nicht verändern, so z.B. Statusabfragen oder Operationen, die am Status des Dokumentes (z.B. hinsichtlich reservierter Teile) nichts verändern.

6.2.2 Der Editor-Server

Während die E-Clients innerhalb der Editor-Architektur die Repräsentanten der Benutzer darstellen, übernimmt der E-Server (im folgenden Abschnitt Server genannt) die koordinierenden Aufgaben, die es ermöglichen sollen, ein sinnvolles nebenläufiges Arbeiten mehrerer Benutzer mit einem gemeinsamen Dokument durchzuführen. Ausserdem bildet der Server die Schnittstelle zum Konferenzsystem, mit dem er Informationen über Benutzer (Berechtigungen) und Dokumente (Attribute, Inhalte) austauschen muss. Zu diesem Zweck nimmt der Server die Anfragen der mit ihm verbundenen Clients entgegen und bewertet sie nach einem internen Schema, das sich an den folgenden Kriterien orientiert.

- aktuelle Strategie

  Es gibt verschiedene Strategien, mit denen der Server den Clients Veränderungen des Dokumenteninhaltes bekanntgibt. Es wurden zeitgesteuerte, volumengesteuerte, gemäss logischen Einheiten gesteuerte und benutzergesteuerte Verfahren vorgestellt. Je nachdem, welches oder welche Kombination dieser Verfahren ausgewählt sind, regelt der Server die Uebertragung von Aenderungen an die Clients.

- Zustand des Client

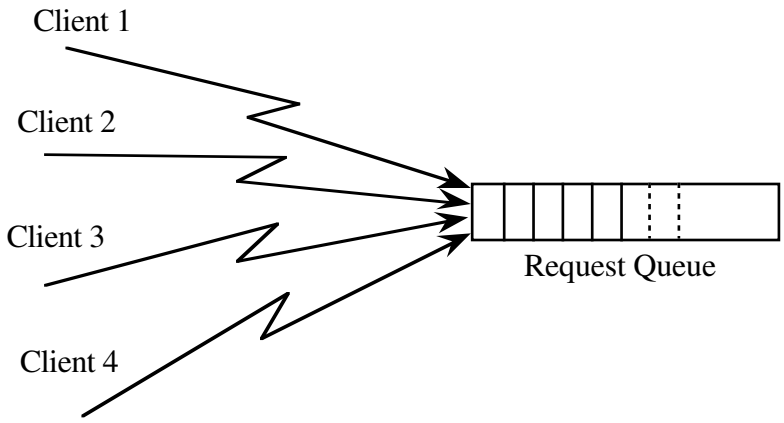
  Clients müssen, um Modifikationen an einem Dokumententeil vornehmen zu können, diesen zuvor reservieren, um Konflikte mit anderen Clients zu vermeiden. Aenderungen, die an nicht oder von anderen Clients reservierten Dokumententeilen vorgenommen werden sollen, werden somit vom Server zurückgewiesen und müssen vom betreffenden Client lokal rückgängig gemacht werden.

– Zustand des Dokumentes

  Ein Dokument ist, wenn es von mehreren Benutzern editiert wird, in den meisten Fällen nicht an allen Stellen für alle Clients änderbar, sondern Teile sind generell nicht änderbar, z.B. wenn sie von einem Benutzer erstellt wurden, der anderer Benutzern nur Read-Only Rechte einräumt, andere Teile sind momentan von anderen Clients reserviert, die diese modifizieren wollen. Aus diesen Betrachtungen resultiert, dass Anfragen eines Clients zur Reservation eines Dokumententeiles nicht notgedrungen erfolgreich bearbeitet werden können.

Nur, wenn eine Ueberprüfung all dieser Kriterien ergibt, dass ein Client berechtigt ist, eine dem Server gemeldete Modifikation eines Dokumententeiles auszuführen oder eine angeforderte Reservation durchzuführern, wird die Anfrage des Client positiv beantwortet und die Aenderung – gemäss der gewählten Strategie – an alle am Editiervorgang beteiligten Clients weitergegeben oder – im Falle der Reservation – als Zustandsänderung des Dokumentes im Server gespeichert. Eine negative Antwort des Servers bedeutet, dass die vom Client gewünschte Modifikation des Dokumentes nicht vorgenommen werden kann. Das Protokoll legt dabei fest, ob diese negative Antwort spezifisch, d.h. mit einer erklärenden Komponente, oder generell ist.

Der Server muss die Anfragen mehrerer Clients bearbeiten können, auch wenn diese parallel konkurrierende Anfragen stellen. Dies kann z.B. auftreten, wenn mehrere Benutzer gleichzeitig einander überlappende Teile des Dokumentes reservieren wollen. Der Server muss in einer solchen Situation in der Lage sein, eine der beiden Reservierungen korrekt durchzuführen und die andere abzuweisen, ohne die Integrität der internen Datenstrukturen zu verletzen. Zu diesem Zweck werden die von den Clients empfangenen Nachrichten sequentialisiert, so dass parallele Anfragen der Clients sequentiell bearbeitet werden. Die folgende Abbildung verdeutlicht diesen Aufbau.



Dieser Aufbau der Request Queue stellt sicher, dass keine Konflikte in der Bearbeitung der Anfragen der Clients auftreten können. Bedingung dafür ist die vollständige Bearbeitung eines Request, bevor der nächste aus der Queue genommen und interpretiert wird.

6.2.3 Das Editor-Protokoll

Die Kommunikation zwischen Clients und Server muss gemäss einer Vereinbarung stattfinden, die es allen Beteiligten erlaubt, ein bestimmtes Verhalten des jeweiligen Kommunikationspartners vorauszusagen. Die Gesamtheit dieser Vereinbarungen bildet ein Protokoll, das aus zwei Komponenten besteht.

– *Protocol Data Units (PDUs)*

  Die Protokolldateneinheiten eines Protokolls geben an, welche Struktur die Daten haben, die zwischen den Kommunikationspartnern ausgetauscht werden und welche Bedeutung[12] den einzelnen Komponenten der Struktur zukommt. Sie bilden somit die syntaktische Grundlage des Protokolls, die notwendig ist, um beiden Seiten eine einheitliche Interpretation der ausgetauschten Daten zu ermöglichen.

[12]In diesem Fall ist lediglich die Bedeutung für das Protokoll gemeint, z.B. Laufnummern oder Bestätigungen, die Bedeutung der Nutzdaten ist i.A. nicht durch das Protokoll geregelt und wird von den Applikationen bestimmt, die das Protokoll zur Kommunikation verwenden.

– Verhaltensbeschreibung

  Während die PDUs eine statische Beschreibung der ausgetauschten Daten darstellen, dient die Verhaltensbeschreibung dazu, die Reaktion der Kommunikationspartner auf eine bestimmte Aktion festzulegen. Diese dynamische Beschreibung legt demnach fest, in welcher Weise sich eine Komponente zu verhalten hat, wenn sie gemäss einem definierten Protokoll kommunizieren will. In diesem Zusammenhang werden oft Zustandsmodelle verwendet, die den Austausch bestimmter PDUs nur in bestimmten Zuständen erlauben und den Uebergang der Komponenten zwischen den verschiedenen Zuständen definieren.

Um ein Protokoll vollständig beschreiben zu können, müssen demzufolge diese beiden Aspekte des Protokolls definiert werden. Das allgemein verbreitete Vorgehen besteht darin, die PDUs formal zu beschreiben (was bei der Implementierung ohnehin notwendig wird) und das Verhalten informell zu spezifizieren, z.B. durch einen Zustandsgraphen mit Angaben, welche PDUs in welchen Zuständen erlaubt sind und welche Uebergänge zwischen den Zuständen existieren. Für die Beschreibung der PDUs hat sich im Kontext von OSI-Kommunikation die in [X.208] bzw. [ISO 8824] beschriebene *Abstract Syntax Notation One (ASN.1)* etabliert, die eine von Repräsentationen unabhängige Syntaxbeschreibung gestattet. Der Austausch von Dateneinheiten wird auf Basis dieser Notation mit Hilfe der in [X.209] bzw. [ISO 8825] festgelegten *Basic Encoding Rules* vorgenommen, die die Daten aus der lokalen Syntax des Senders in eine einheitliche Transfersyntax und anschliessend in die lokale Syntax des Empfängers umformen.

Eine abstrakte Methode zur Definition von ausgetauschten Protokolldateneinheiten ist die der im RO-Standard ([X.219] und [X.229] bzw. [ISO 9072]) definierten *Remote Operations*, auf die in Kapitel 9 näher eingegangen werden wird. Die Notation der Remote Operations besteht aus einigen ASN.1-Makros, die die Festlegung eines Namens jeder Operation, ihrer Argumenttypen, Resultattypen und Fehlercodes erlauben. Auf diese Weise kann eine jede Operation, die auf dem entfernten System ausgelöst wird, vollständig beschrieben werden. An dieser Stelle soll nich näher auf ROSE eingegangen werden, es soll lediglich die Notation vorgestellt werden. Sie besteht im wesentlichen aus den soeben erwähnten Komponenten.

```
Loesche OPERATION
   ARGUMENT    Objekt
   RESULT      Vorgaenger
   ERRORS      { LoeschenVerboten, ObjektUndefiniert }
   ::=         1
```

Das vorangegangene Beispiel definiert eine Operation `Loesche`, die als Argument einen Parameter vom Typ `Objekt` erhält. Das Ergebnis der Operation (falls die Ausführung erfolgreich ist) ist eine Liste der `Vorgaenger` des Objektes, mögliche Fehler sind die Reservation eines Objektes oder die Spezifikation eines nicht existierenden Objektes. Der Operation wird die Nummer 1 zugewiesen. Weitere ASN.1-Definitionen müssen die verwendeten Datentypen (`Objekt` und `Vorgaenger`) definieren, bis sie nur noch mit Hilfe von Basic Types festgelegt sind.

Eine Operation kann bestätigt oder nicht bestätigt sein, d.h. die die Operation auslösende Instanz kann vom Erfolg oder Misserfolg der Operation unterrichtet werden, oder nach dem Auslösen der Operation verläuft deren Ausführung vollkommen asynchron, ohne dass die auslösende Instanz vom Ergebnis der Operation (Resultat oder Fehler) erfährt. Für ein auf Remote Operations aufbauendes Protokoll ist es entscheidend, welche Charakteristik die verwendeten Operationen aufweisen. Die folgenden Ausführungen sollen diese unterschiedlichen Formen und ihre Verwendung beschreiben.

– Alle Operationen sind bestätigt

  Sind alle Operationen zwischen E-Client und E-Server bestätigt, so ist ein Höchstmass an Sicherheit bezüglich der Operationsausführung gegeben. Für einige Operationen ist eine Bestätigung unerlässlich, so z.B. für die Reservierung eines Teils des Dokumentes, um dieses zu modifizieren. Nur im Falle einer Bestätigung durch den Server können die beabsichtigten Aenderungen vorgenommen werden, im Fehlerfall ist der Teil entweder bereits durch einen anderen Teilnehmer reserviert oder die Reservierung ist auf Grund der Zugriffsrechte nicht erlaubt.

– Alle Operationen sind unbestätigt

  Ein Protokoll, das ausschliesslich unbestätigte Operationen enthält, sichert ein Maximum an asynchroner Kommunikation zu, indem die auslösende Instanz niemals gezwungen ist, auf das Ergebnis einer Operation zu warten, sondern sofort nach dem Auslösen der Operation weiterarbeiten kann. Für einige Operationen, die für das Arbeiten der auslösenden Instanz nicht unabdingbar notwendig sind (wie z.B. die Mitteilung der aktuellen Cursorposition), ist dieses Vorgehen günstig, da so erreicht wird, dass nicht unnötig auf Ergebnisse gewartet wird.

– Mischform

  Die vorangegangenen beiden Abschnitte zeigen, dass eine Beschränkung auf ausschliesslich bestätigte oder ausschliesslich unbestätigte Operationen wenig sinnvoll ist, sondern – abhängig von der jeweiligen Operation – einzeln festgelegt werden sollte, ob eine Operation bestätigt oder unbestätigt ist. Für Operationen, die Modifikationen am Dokument zur Folge haben, die alle Teilnehmer am Editiervorgang betreffen, ist eine Bestätigung unerlässlich, da die Ueberprüfung, ob eine solche Operation gestattet ist, zunächst vom Server vorgenommen werden muss, bevor der Benutzer mit den Erfolg der Operation rechnen kann. Andererseits gibt es Operationen deren Ausführung nicht unbedingt erfolgreich sein muss, um die Fortsetzung des Editiervorganges zu ermöglichen, diese sollten aus Effizienzgründen unbestätigt ausgelöst werden.

– Spezielle RO-Formen

  Die in Kapitel 9 näher beschriebenen Remote Operations erlauben neben den beiden hier angeführten Formen von Operationen ebenso spezielle Formen, bei denen jeweils nur das Resultat bzw. der Fehlerfall bestätigt sind, während im jeweils anderen Fall keine Bestätigung erfolgt. Es muss im Einzelfall untersucht werden, inwieweit diese Operationen Verwendung finden können. Des weiteren wird im RO-Standard zwischen synchronen und asynchronen bestätigten Operationen unterschieden, die die auslösende Operation bei der Ausführung der Operation blockieren oder nicht. Auch für diese Unterscheidung gilt das oben Gesagte.

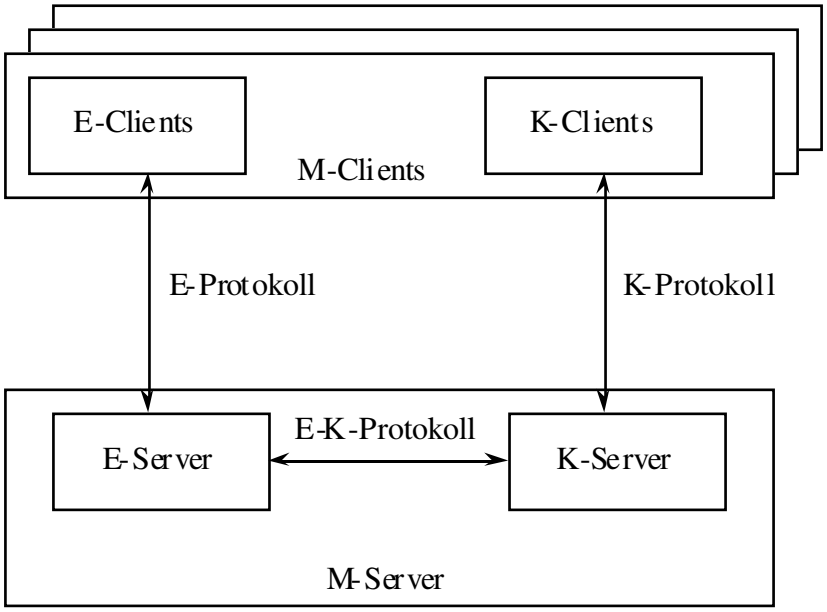
Aufbauend auf diesen Ueberlegungen zu den Eigenschaften der Operationen des Protokolls kann nun ein Protokoll definiert werden, das aus einer Menge von Operationen sowie den jeweils geltenden Vor- und Nachbedingungen besteht. Zudem muss für jede Operation angegeben werden, ob sie bestätigt (in einer der drei möglichen Formen) oder unbestätigt ist und ob sie im Falle einer Operation, die sowohl hinsichtlich des Resultates als auch hinsichtlich eines Fehlers bestätigt ist, synchron oder asynchron ausgeführt werden soll. Um die Vor- und Nachbedingungen anzugeben, ist es sinnvoll, ein Modell für die Zustände von Clients, Server und Dokument einzuführen, das die einfache Formulierung von Zustandsübergängen erlaubt. Ein Modell für E-Clients wurde bereits im vorangegangenen Abschnitt vorgestellt, dort wurde zwischen drei Zuständen connected, editing und modifying unterschieden, der Server ist über die Zustände aller Clients informiert.

6.2.4 Das Editor-Konferenz-Protokoll

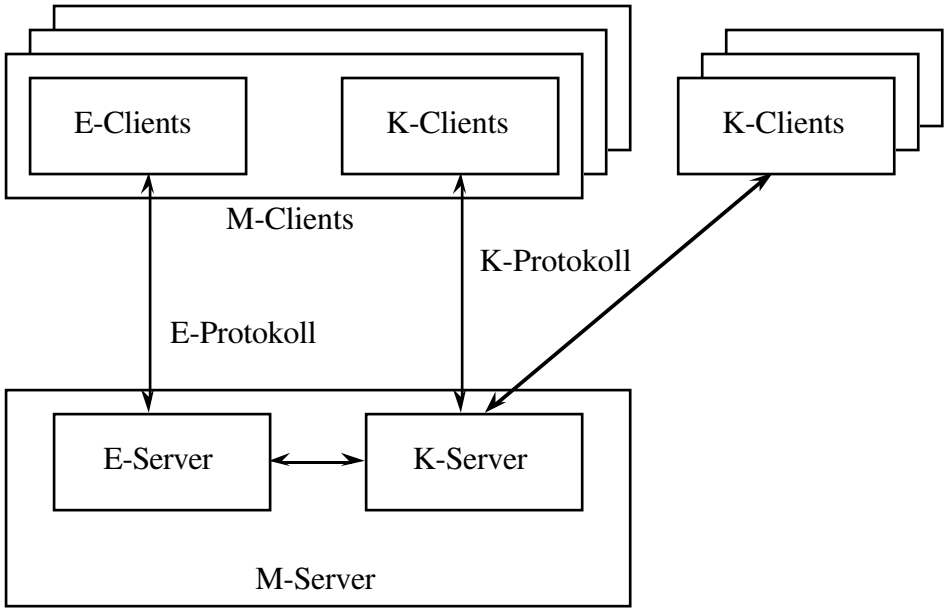
Nach den zuvor beschriebenen Ueberlegungen ist es nun notwendig, ein weiteres Protokoll zu definieren, das den Austausch von Informationen zwischen dem E-Server und dem K-Server ermöglicht. Dies ist notwendig, um Verwaltungsinformationen (über Benutzerauthentifizierungen oder Zugriffsrechte) und Dokumente zwischen dem vom Konferenzsystem verwalteten *Shared Workspace (SWS)* und dem Editor auszutauschen.

6.3 Gesamtarchitektur

Die im vorangegangenen Abschnitt vorgestellte Architektur des Editors orientiert sich, wie Eingangs dargestellt, an der Architektur von MultimETH, d.h. es wird ein Client/Server-Modell verwendet. Das Gesamtsystem (unter Einbeziehung des bereits existierenden Konferenzsystems) lässt sich somit folgendermassen darstellen.



Die beiden Typen von Clients sind nicht miteinander verbunden, was den Grund darin hat, dass auf diese Weise Inkonsistenzen vermieden werden können, da die Kommunikation der beiden Teilsysteme Editor und Konferenzsystem nur innerhalb des M-Server stattfindet (über das E-K-Protokoll). Dieser Aufbau ist ebenfalls dazu geeignet, eine klare Trennung zwischen dem Editor und dem Konferenzsystem herzustellen, die benötigt wird, um die beiden Komponenten möglichst gut voneinander zu trennen und somit, gemäss den in Kapitel 2 beschriebenen Zielen, einen getrennten Einsatz zu ermöglichen. Dieser liesse sich mit der folgenden Darstellung veranschaulichen, die die kombinierte Verwendung



7 Multimediafähigkeiten

Neben den Mehrbenutzerfähigkeiten sollte der Editor ebenfalls Multimediafähigkeiten bereitstellen, d.h. es sollen verschiedene Informationstypen unterstützt werden. Die Unterstützung muss sowohl das verwendete Datenformat beinhalten (also Möglichkeiten zur Speicherung eines solchen Informationstypes bereitstellen), als auch eine Benutzerschnittstelle anbieten, die die Erstellung und Manipulation solcher Informationstypen ermöglicht. Da es sich bei den Informationstypen um an der Be-

nutzersicht orientierte Gruppierungen von Informationen handelt, lassen sich beliebig viele solcher Typen denken.

Im Rahmen des MultimETH-Projektes wird es sicher nicht möglich sein, eine weit über Basisfunktionen hinausgehende Lösung zu implementieren, aus diesem Grund wird eine Unterteilung vorgenommen in Grundfunktionalität, d.h. Informationstypen, die auf jeden Fall unterstützt werden sollen, und erweiterte Funktionalität, also Informationstypen, die für spezielle Benutzergruppen wünschenswert, jedoch nicht unbedingt notwendig sind. Aus diesen kann für Demonstrationszwecke ein Typ ausgewählt werden, dessen Implementierung die Möglichkeiten des Editors zeigen kann.

7.1 Grundfunktionalität

Unter der Grundfunktionalität eines Editors hinsichtlich der Multimediafähigkeiten werden diejenigen Informationstypen verstanden, deren Bereitstellung als minimale Anforderung verlangt wird. Dies sind die Informationstypen, die universell (für beliebige Benutzergruppen) einsetzbar sind[13] und die den traditionellen Begriff des ausschliesslich textbasierten Dokumentes deutlich erweitern.

- Text

  Grundlage nahezu jeden Dokumentes ist eine textuelle Beschreibung, somit gehört dieser Informationstyp zu den Grundfunktionalitäten. Neben einfachem, unstrukturiertem Text sollten formatierte Texte verwendet werden können. Die logische Struktur eines Textes, d.h. die Einteilung in Abschnitte, Kapitel u.ä., ist nicht Bestandteil des Infromationstyps, sondern der Dokumentenarchitektur, dagegen sind Formatieranweisungen, wie Hervorhebungen oder Wechsel des Fonts, Teil eines Textteiles und deshalb Bestandteil des Informationstypes.

- Graphiken

  Für viele Sachverhalte kann die Information, die mit Hilfe eines Bildes vermittelt werden kann, sehr viel nützlicher sein als eine textuelle Beschreibung. Besonders im Fall der Darstellung von komplexen Zusammenhängen und Strukturen ist eine graphische Darstellung nicht durch Text ersetzbar. Aus diesem Grund gehört die Fähigkeit, Graphiken in ein Dokument zu integrieren, zu den grundlegenden Anforderungen an ein multimediafähiges System.

  Graphiken sollten nicht nur aus einfachen Liniensegmenten zusammengesetzt werden können, sondern es sollten graphische Primitiven existieren (z.B. Linien, Rechtecke, Kreise oder ein Mechanismus, eigene Primitiven zu definieren), aus denen eine Darstellung aufgebaut werden kann. Geschlossene Linienzüge sollten mit einem Füllmuster versehen werden können. Zudem sollten Ausschnittsdefinitionen und Aenderungen des Massstabs unterstützt werden, um verschiedene Sichten auf die gleiche Graphik zu ermöglichen.

- Rasterbilder (Bitmap Images)

  Rasterbilder dienen der digitalisierten Darstellung von kontinuierlichen Bildern, die sowohl hinsichtlich der räumlichen Auflösung als auch hinsichtlich der farblichen Auflösung in Mengen von diskreten Werten umgewandelt werden müssen, um digital repräsentierbar zu sein. Die räumliche Auflösung wird in einem Mass von Bildpunkten pro Entfernungseinheit angegeben, meist in *Dots per Inch (dpi)*, die farbliche Auflösung in der sogenannten Farbtiefe, die meist in Bit/Pixel angegeben wird. Uebliche Werte sind 72dpi oder 300dpi sowie Farbtiefen von 1 Bit (was einer Schwarz-Weiss-Darstellung entspricht) bis zu 32 Bit (24 Bit für die Farbinformation, z.B RGB oder andere Farbmodelle, und 8 Bit für die Helligkeit).

  Da die bei Rasterbildern auftretenden Datrenmengen sehr gross werden können, sind viele verschiedene Methoden zur Komprimierung entwickelt worden, die von einfachen Lauflängenverfahren bis zu Transformationscodierungen reichen, die die Bildinformation von 24 Bit Bildern ohne grosse Verluste auf deutlich weniger als 1 Bit/Pixel reduzieren können. Der Unterschied dieser Methoden besteht zum einen darin, ob es sich um verlustfreie oder verlustbehaftete Verfahren handelt, zum anderen im Rechenaufwand, der für die Komprimierung und die Dekomprimierung notwendig ist.

Die hier aufgeführten Informationstypen entsprechen denen, die in der aktuellen Version des ODA-Standards [ISO 8613] als zulässige Inhaltsarchitekturen (*Content Architectures*) definiert sind. ODA verwendet als graphischen Informationstypen den *Computer Graphics Metafile (CGM)*, der in [ISO 8632] standardisiert ist. Dieser bietet vielfache Möglichkeiten, aus graphischen Elementen bestehende

[13] Beispiele für nicht universell einsetzbare Informationstypen, die speziell auf die Anforderungen bestimmter Benutzergruppen zugeschnitten sind oder nur von einer kleinen Menge von Benutzern benötigt werden, finden sich im folgenden Abschnitt.

Darstellungen zu speichern. Rasterbilder werden gemäss den CCITT Richtlinien [T.4] oder [T.6] oder nach einem ODA-spezifischen *Bitmap Encoding Scheme* codiert, damit besteht die Möglichkeit, je nach Anwendungen verschiedene Codierungen zu verwenden.

7.2 Erweiterte Funktionalität

Neben den im vorangegangenen Abschnitt beschriebenen Informationstypen, die in einem multimediafähigen System vorhanden sein müssen, gibt es weitere Typen, deren Vorhandensein nicht unbedingt notwendig ist, die jedoch die Leistungsfähigkeit und damit die potentielle Benutzergruppe eines Editors jeweils erweitern. Die im folgenden aufgeführten Informationstypen sind für verschiedenste Anwendungen nutzbar und können deshalb noch von einem allgemein verwendbaren System angeboten werden.

- Tabellen

- Spreadsheets

- Mathematische Formeln

- Audio

  Neben all den bisher aufgeführten Informationstypen, die sich zweituabhängig auf einem Bildschirm präsentieren lassen, zeigt der Informationstyp Audio neue Eigenschaften. Er lässt sich weder auf einem Bildschirm darstellen, noch kann die Präsentation zeitunabhängig geschehen. Aus diesem Grund bietet dieser Informationstyp neue Qualitäten in der Dokumentenverarbeitung, die durch keinen der bisher beschriebenen Typen ersetzt werden können.

  Audio lässt sich in vielen verschiedenen Formen repräsentieren, die sich in der Qualität und im Grad der Abstraktion unterscheiden. Eine verbreitete Form der Audiorepräsentation ist die einer digitalisierten Schallwelle, in deisem Fall bestimmen die Parameter Abstastfrequenz und Quantisierung die Qualität der Aufzeichnung. Es gibt auf diesem Gebiet verschiedene Standards oder de-facto-Standards, die von für das menschliche Ohr fehlerfreier Reproduktion bis zur Wiedergabe in Telephonqualität reichen. Ebenso wie bei Rasterbildern (und dem anschliessend beschriebenen Video) stellen Kompressionsverfahren angesichts der grossen Datenmengen eine wichtige Rolle, und auch hier existiert die Unterscheidung in verlustfreie und verlustbehaftete Verfahren, die sich .zudem durch den Codierungsaufwand stark unterscheiden.

  Der Informationstyp Audio sollte einige Modifikationen des repräsentierten Signals zulassen, vergleichbar den Hervorhebungen bei normalem Text, so z.B eine Aenderung der Lautstärke, idealerweise zeitabhängig und getrennt für verschiedene Frequenzbänder (Klangregelung), die Repräsentation von Mehrkanal-Audio sowie die Festlegung von Sequenzen innerhalb des Objektes, um beliebige Kombinationen von Ausschnitten präsentieren zu können.

- Video

Als dritte Gruppe von Informationstypen liessen sich die benutzerspezifischen Informationstypen betrachten, die nicht allgemein für alle Anwendungen eines Editors festgelegt werden können, sondern von beschränkten Benutzergruppen verwendet werden. Zu diesen Typen gehören produktdefinierende Daten (obwohl auch hier internationale Standards in der Entwicklung sind) sowie jegliche Arten von Informationstypen, die sich mit Hilfe der vorhandenen Grundtypen zusammensetzen lassen. Da es sich bei diesen Informationstypen um Typen handelt, die sich nicht vorab festlegen lassen, sondern mit den Anforderungen einer jeden Gruppe entstehen, können hier keine Festlegungen getroffen werden.

8 Präsentationsaspekte

8.1 Benutzeroberfläche

8.2 Dynamische Präsentationen

9 Integration in das Konferenzsystem

Wie bereits in den vorangegangenen Abschnitten erwähnt wurde, soll der Editor nicht als isoliertes Programm ohne Integrationsmöglichkeiten erstellt werden, sondern eine funktionale Einheit mit dem Konferenzsystem MultimETH bzw. dem Prototypen LittlemETH bilden. Eines der Designziele des Editors ist, ihn sowohl alleine als auch in Verbindung mit dem Konferenzsystem verwenden zu können. Zu diesem Zweck muss die Software beider Komponenten eine deutliche und möglichst einfache Schnittstelle aufweisen (das E-K-Protokoll), die benutzt werden kann, um die Interaktion zwischen diesen abzuwickeln. Im vorliegenden Kapitel soll untersucht werden, in welcher Form die Software organisiert werden kann, um die notwendige Kommunikation zwischen den einzelnen Komponenten zu ermöglichen. Es wird somit eine nähere Betrachtung der in Abschnitt 6.3 vorgestellten Gesamtarchitektur vorgenommen, die sich an der Implementierung von LittlemETH mit den daraus resultierenden Bedingungen orientiert.

Association Control Service Element (ACSE) ([X.217] bzw. [ISO 8849] und [X.227] bzw. [ISO 8850])

Ein weiterer Teil der verwendeten OSI-Services ist das *Remote Operations Service Element (ROSE)*, das in [X.219] und [X.229] bzw. [ISO 9072] standardisiert ist. ROSE erlaubt die Strukturierung der Kommunikation in Anfrage/Antwort-Interaktionen (*Request/Reply Interactions*), die gut dazu geeignet sind, eine Client/Server-Architektur zu unterstützen, sie werden im folgenden als Operationen bezeichnet. Operationen sind durch eine Identifikation, die den Tyop der Operation bezeichnet, Parameter, die Argumente der Operation definieren, und mögliche Ergebnisse oder Fehler charakterisiert. Ja nachdem, in welcher Weise sich die beteiligten Instanzen während der Interaktion verhalten, können unterschiedliche Klassen von Operationen unterschieden werden.

- Operation Class 1: Synchronous, reporting success or failure (result or error).

  In dieser Klasse ist die anfragende Instanz, der *Invoker*, solange blockiert, bis die antwortende Instanz, der *Responder*, mit einem Ergebnis oder einem Fehler geantwortet hat, es handelt sich hierbei demnach um eine synchrone Ausführung der Operation.

- Operation Class 2: Asynchronous, reporting success or failure (result or error).

  Aehnlich wie bei der Klasse 1 werden hier sowohl Ergebnisse als auch Fehler an den Invoker gemeldet, jedoch ist dieser während der Bearbeitung der Operation durch den Responder nicht blockiert, sondern kann asynchron weiterarbeiten. Je nach Art der ROS-Schnittstelle wird der Invoker bei vorliegender Antwort unterbrochen (aktive Schnittstelle) oder er kann diese abfragen (passive Schnittstelle).

- Operation Class 3: Asynchronous, reporting failure (error) only, if any.

  Diese Klasse gleicht der Klasse 2 mit dem Unterschied, dass lediglich Fehler gemeldet werden, sie eignet sich also für Operationen, die kein Ergebnis haben, das dem Invoker mitgeteilt werden muss.

- Operation Class 4: Asynchronous, reporting success (result) only.

  Operationen der Klasse 4 sind ähnlich denen der Klasse 2, nur dass hier ein eventuell aufgetretener Fehler nicht gemeldet wird.

- Operation Class 5: Asynchronous, outcome not reported.

  Die letzte Klasse der Opeartionen schliesslich ist dadurch definiert, dass dem Invoker weder Ergebnisse noch Fehler gemeldet werden, er über Erfolg oder Misserfolg einer Operation also keinerlei Informationen erhält. Diese Klasse eignet sich besonders für Operationen, deren Ergebnis nicht von grosser Wichtigkeit für den Invoker ist.

Ein weieteres Konzept, Operationen auszuführen, besteht in den Linked Operations, die auf Grund der Annahme eingeführt wurden, dass der Responder die geforderte Aufgabe nicht in jedem Fall ohne Rückfrage bearbeiten kann, sondern u.U. nähere Informationen vom Invoker benötigt. Ist dies der Fall, so kann vor Beantwortung der *Parent-Operation*, also der vom Invoker ausgelöste Remote Operation, *Child-Operations* auf dem Invoker ausführen, um für die Bearbeitung notwendige Informationen zu erhalten. Dieses Vorgehen kann rekursiv fortgesetzt werden, d.h. der Invoker kann diese Child-Operations seinerseits als Parent-Oprations auffassen und daraufhin *Child-Operations* auf dem Responder

ausführen. In diesem Zusammenhang wird es wichtig, zu betrachten, welche der beteiligten Instanzen das Recht bzw. die Möglichkeit hat, Operationen auszulösen. ROSE unterscheidet zu diesem Zweck zwischen drei verschiedenen Association Classes.

– Association Class 1

  Diese Klasse beinhaltet die Einschränkung, dass nur die Instanz, die die Association etabliert hat, (der *Association-Initiator*) Operationen ausführen darf. Demzufolge kann der Association-Responder nur Operationen bearbeiten, er kann jedoch keine Operationen ausführen lassen.

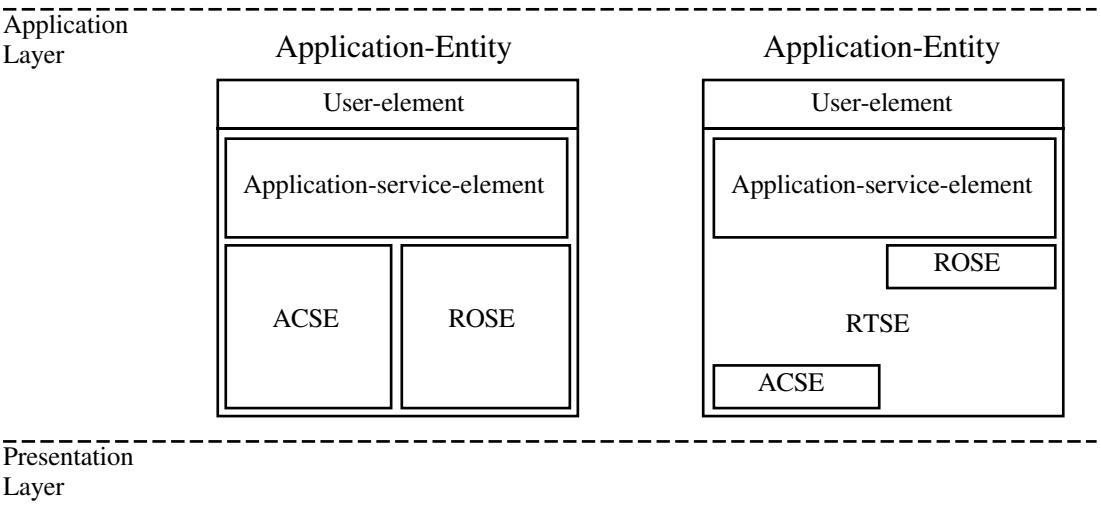
– Association Class 2

  In dieser Klasse sind die Rechte der Operationsausführung vertauscht, d.h. der Association-Initiator darf nur Operationen ausführen, während der Association-Responder das Recht hat, Operationen zu initiieren.

– Association Class 3

  Keine Einschränkungen hinsichtlich der Operationsausführung gibt es in der Klasse 3, wo beide Instanzen das Recht haben, Operationen zu initiieren und demzufolge auch beide Instanzen als Responder arbeiten können. Daraus folgt unmittelbar, dass für die Verwendung von Linked Operations eine Association der Klasse 3 notwendig ist.

Ein optionales, mit dem Remote Operations Service Element zusammenarbeitendes, Service Element, ist das *Reliable Transfer Service Element (RTSE)* , das in [X.218] und [X.228] bzw. [ISO 9066] standardisiert ist. RTSE dient der zuverlässigen Uebertragung von grossen Datenmengen. Wird RTSE nicht verwendet, so müssen die ROSE verwendenen Service Elements eine Association explizit mit ACSE herstellen. Bei der Verwendung von RTSE benutzen die darauf aufbauenden Service Elements nur noch die Services von RTSE und ROSE, RTSE verwendet dann ACSE, um eine Association aufzubauen und dient weiterhin als Basis für ROSE. Diese beiden möglichen Konfigurationen bei der Anwendung von Remote Operations sind in der folgenden Darstellung verdeutlicht.



RTSE wird verwendet, um die Uebertragung von Application-PDUs zwischen zwei OSI-Endsystemen zu vereinfachen. Der Reliable Transfer ist unempfindlich gegen Störungen des Kommunikationssystems oder eines Endsystems, so dass die Menge der von den Applikation erneut zu uebertragenden PDUs klein gehalten werden kann. RTSE baut auf ACSE auf und bietet die von diesem angebotenen Services, nämlich den Auf- und Abbau einer Association, dem Benutzer an der eigenen Schnittstelle als RT-Services an.

9.1 Aufbau von MultimETH

Der verteilte Editor soll innerhalb der durch die bisherigen Implementierungsarbeiten am Protoypen vorgegebenen Umgebung realisiert werden. Diese kann durch folgende Eigenschaften charakterisiert werden.

– Client/Server-Kommunikation über ISODE

  Die derzeitige Implementierung von K-Server und K-Client verwendet zur Kommunikation dieser beiden Komponenten ISODE. Das Protokoll ist als eine Menge von Remote Operations gemäss dem ROS-Standard definiert.

– Verwendung des Andrew Toolkit für das GUI

  Als X Toolkit für die Programmierung des GUI wird das Andrew Toolkit verwendet. Es ermöglicht objektorientierte Programmierung und unterstützt durch eine vordefinierte Menge an Klassen eine grosse Zahl von Informationstypen.

– Trennung des Client in zwei Prozesse

  Auf Grund von Interworking-Problemen zwischen der Kommunikationssoftware (ISODE) und dem für das GUI verwendeten Andrew Toolkit sind diese beiden, gemeinsam den K-Client implementierenden Komponenten als getrennte Prozesse realisiert, die über den IPC-Mechanismus einer *Message Queue* miteinander kommunizieren. Es wird eine Message Queue verwendet, die die Nachrichten in beide Richtungen aufnimmt. Dabei wird ausgenutzt, dass ein Prozess selektiv aus einer Message Queue lesen kann, also nur Nachrichten spezifischer Typen.

– ???

– ???

Die folgenden Abschnitte untersuchen die Frage, wie man die in Abschnitt 6.3 vorgestellte Architektur so implementieren kann, dass der Aufwand für eine Integration in den bestehenden Prototypen einerseits möglichst gering gehalten wird, andererseits jedoch die alleinige Verwendung des verteilten Editors ohne grosse Modifikationen ermöglicht wird. Die prinzipielle Unterscheidung der zwei vorgestellten Ansätze beruht darauf, ob für die Kommunikation zwischen E-Server und E-Client eine eigene OSI-Association verwendet wird, oder ob die bereits vorhandene Verbindung zwischen K-Server und K-Client verwendet wird, um Nachrichten des E-Protokolls zu übertragen.

9.2 Zwei Associations

Ueberträgt man die in Abschnitt 6.3 dargestellte Architektur direkt in ein zu implementierendes Modell, so ergibt sich eine Struktur, in der zwei voneinander unabhängige Verbindungen zwischen den K-Komponenten und den E-Komponenten bestehen. Daraus ergeben sich vorrangig drei Konsequenzen, die dieses Modell als weniger gut geeignet für eine Implementierung erscheinen lassen.

– Zweifacher Verwaltungsaufwand

  Werden zwei Associations verwendet, so besteht die Notwendigkeit, all den Verwaltungsaufwand, der mit einer OSI-Verbindung zusammenhängt, auf beiden Endsystemen zweifach aufzubringen. Dies erscheint nicht unbedingt notwendig, da die Verbindung ohnehin zwischen den gleichen Endsystemen erfolgt. Weiterhin resultiert aus dem doppelten Verwaltungsaufwand die Notwendigkeit, grosse Teile des existierenden Codes zu kopieren, da die gesamte Kommunikationsbehandlung sowohl in den K-Komponenten als auch in den E-Komponenten vorhanden sein müsste.

– Aufspaltung des E-Client

  Die Interworking-Probleme, die im Falle des K-Client dazu geführt hatten, eine Trennung in zwei Prozesse vorzunehmen, träten bei der Implementierung des E-Client vermutlich ebenso auf, da auch in diesem Fall ISODE und das Andrew Toolkit zusammenarbeiten müssen. Eine Aufspaltung des E-Client in Andrew- bzw. ISODE-spezifische Teile wäre in diesem Fall ebenso notwendig, womit auch dieser Code (zur Kopplung der Prozesse über eine Message Queue) in zweifacher Ausfertigung vorhanden wäre.

– Probleme der Server/Server-Kopplung

  Wie in Abschnitt 6.2.4 beschrieben, müssen die beiden Server miteinander kommunizieren. Dies geschieht ebenfalls über ein Protokoll, das die Schnittstelle zwischen diesen Komponenten definiert. Wird nun diese Kommunikation ebenfalls mit RO durchgeführt, so muss einer der Server in dieser neuen Kommunikationsbeziehung die Rolle des Client übernehmen, was zu Schwierigkeiten in Verbindung mit der ihm ebenfalls zugewiesenen Rolle eines Server führen kann. Eine mögliche Modellierung der Abhängigkeit zwischen K- und E-Server könnte mit Linked Operations vorgenommen werden, doch selbst dann bliebe das Problem, welcher der beiden Server als Client betrachtet werden soll.

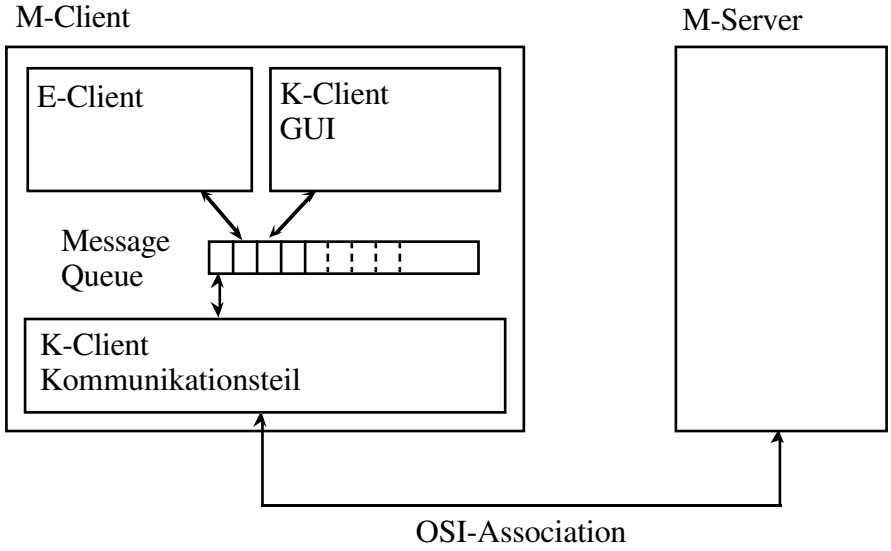
Die beiden angeführten Probleme lassen die Lösung mit zwei getrennten Associations als wenig vorteilhaft erscheinen. Im folgenden Abschnitt wird daher eine Lösung vorgestellt, die diesen Overhead umgeht und bereits realisierte Konzepte des Prototypen verwendet, um mit nur einer OSI-Association die Kommunikation zwischen den K- und E-Komponenten durchzuführen.

9.3 Eine Association

Die im vorangegangenen Abschnitt beschriebenen Nachteile können umgangen werden, wenn die bereits existierende OSI-Association zwischen K-Server und K-Client verwendet wird, um Nachrichten zwischen E-Server und E-Client zu transportieren. Diese Lösung kann natürlich nicht angewendet werden, wenn ein isoliert arbeitender E-Client (d.h. ohne K-Client) verwendet werden soll, dieser Sonderfall wird daher anschliessend betrachtet werden und es wird sich zeigen, dass das entwickelte Modell auch in diesem Fall eine Lösung ermöglicht. In den beiden folgenden Abschnitten soll zunächst auf die Modellierung von M-Client und M-Server eingegangen werden, die notwendig ist, um die beschriebene gemeinsame Nutzung der Association zu ermöglichen.

9.3.1 Aufbau des Client

Der M-Client soll so aufgebaut sein, dass einerseits nur eine OSI-Association verwendet wird, andererseits jedoch trotzdem eine klare Trennung zwischen E-Client und K-Client existiert. Zu diesem Zweck wird die existierende Trennung zwischen den beiden Komponenten Kommunikationsteil und GUI des K-Client dazu genutzt, den E-Client in diese Konfiguration einzubinden. Die geschieht dadurch, dass die zwischen den beiden K-Client-Komponenten aufgebaute Message Queue ebenfalls vom E-Client verwendet wird. Er erhält auf diese Weise eine Verbindung zum Kommunikationsteildes K-Client und kann mit dessen Hilfe die gemeinsame Association verwenden. Die folgende Darstellung veranschaulicht diesen Aufbau des M-Client.



Aus der Abbildung ist ersichtlich, dass die Message Queue von drei Komponenten für die Kommunikation verwendet wird. Dabei tauschen der E-Client und der GUI-Teil des K-Client keine Nachrichten miteinander aus, was dem Konzept der Trennung der beiden Clients entspricht. Um die Nachrichten innerhalb der Message Queue identifizieren zu können, muss den bisher definierten zwei Typen ein weiterer Typ hinzugefügt werden, der für an den E-Client gerichtete Messages verwendet wird. Das UNIX Betriebssystem bietet die Möglichkeit, ausschliesslich Messages eines bestimmten Typs zu lesen, so dass jede der drei Komponenten mit dem entsprechenden Systemaufruf auf für sie bestimmte Messages warten kann, um diese dann zu lesen und damit aus der Queue zu entfernen.

Um den E-Client nicht auf die Kommunikation mittels einer Message Queue festlegen zu müssen, sondern eine abstrakte Schnittstelle zu einem entfernten Dienst anzubieten, wird in diesem Teil des M-Client ein an die Ideen des *Remote Procedure Call* (der ausführlich in [Nelson 81] beschrieben ist) angelehntes Konzept verwendet. In der prozeduralen Programmierung kann die Verteilung einer Anwendung durch RPC-Mechanismen verborgen werden, deren grundsätzliche Idee darin besteht, sogenannte Stub-Routines bereitzustellen, die die lokale Schnittstelle zu einer entfernt auszuführenden Prozedur darstellen. Wie diese entfernte Ausführung dann tatsächlich realisiert wird, ist für den Benutzer der Stubs unerheblich, er ruft lediglich die lokale Prozedur auf. In einer objektorientierten Umgebung lässt sich eine analoge Modellierung durch die Einführung eines Stub-Object durchführen. Dieses Objekt bietet an seiner Schnittstelle alle die Methoden an, die auf einem entfernten Objekt ausgeführt werden sollen. Die Kommunikation findet nur innerhalb des Stub-Object statt, das damit eine rein lokale Schnittstelle anbieten kann.

9.3.2 Aufbau des Server

9.3.3 Isolierte E-Clients

10 Das Andrew Toolkit (ATK)

In [Borenstein 90] wird das *Andrew Toolkit (ATK)* beschrieben, das Teil des an der Carnegie Mellon University entwickelten *Andrew Systems* ist. Neben dem ATK enthält das Andrew System ein Filesystem (*Andrew File System, AFS*), ein Message System für Multimedia Messages (*Andrew Message System, AMS*) sowie eine eigene Enwicklungsumgebung (*Andrew Development Environment Workbench, ADEW*). Das ATK hat den Zweck, als Grundlage für die Entwicklung von interaktiven Multimedia-Anwendungen zu dienen, so ist es z.B. die Basis des AMS. Es stützt sich dabei auf das X Window System ab, das die Aufgabe erfüllt, eine verteilte Fensterumgebung bereitzustellen. Das ATK verwendet als Basis die Bibliothek *Xlib,* die als unterste Programmierschnittstelle zum *X Protocol* dient. Somit ist das ATK in der X Terminologie ein eigenständiges Toolkit, das neben anderen, wie dem weit verbreiteten Xt Toolkit (mit den UI Komponenten der Athena Widgets oder der OSF/Motif Oberfläche), InterViews oder XView (das die Open Look Oberfläche von Sun unterstützt) existiert.

10.1 Multimediafähigkeiten des ATK

Das Andrew Toolkit unterstützt verschiedene Informationstypen. Diese können mit Objekten verschiedener Klassen erstellt und manipuliert werden. Die Implementierungen einiger dieser Klassen sind im momentanen Zustand von Andrew eher auf einem experimentellen Niveau, so dass die Reimplementierung einiger Klassen bzw. Informationstypen wünschenswert ist.

- Multi-Font Text

  Objekte dieses Informationstyps können Text in verschiedenen Formatierungen enthalten. Damit geht dieser Typ weit über die Möglichkeiten von einfachem ASCII-Text hinaus. Es bestehen unter anderem Formatierungen für Textblöcke (linke und rechte Begrenzungen, Blocksatz, Zeilenabstände), zwischen Textblöcken (Abstand) und für Text (Font, Hervorhebungen, Schriftgrösse).

  Die hier aufgeführten Formatierungsmöglichkeiten sind nur Beispiele, darüber hinaus gibt es weitere Arten der Manipulation von Text innerhalb von Multi-Font Textobjekten. Es erscheint möglich, dass die ATK Textformatierungen ausreichend für den Editor sind und keine spezielle Formatierkomponente implementiert werden muss. Dazu müssen jedoch noch Tests durchgeführt werden, inwieweit die Formatierung den Ansprüchen genügt, die an einen Editor gestellt werden, der die Bearbeitung grosser und komplexer Dokumente erlauben soll.

- Tabellen

  Objekte dieses Typs sind rechteckige Tabellen mit einer beliebigen Anzahl von Spalten und Zeilen. Einträge in den einzelnen Zellen können zentriert oder links- bzw. rechtsbündig ausgerichtet werden. Zellen können mit einer Box umgeben werden, die beim Drucken als einfache schwarze Linie erscheint (im Tabellen-Editor sind immer Linien um die Zellen herum sichtbar), ausserdem können einzelne Linien gezeichnet werden. Die Grösse der Tabelle bzw. der Zeilen und Spalten kann interaktiv verändert werden, des weiteren ist es möglich, ganze Zeilen oder Spalten zu löschen, zu duplizieren oder innerhalb der Tabelle zu verschieben. horizontal und/oder vertikal benachbarte Zellen können miteinander kombiniert werden, um Spalten- oder Zeilenübergreifende Zellen zu schaffen.

- Spreadsheets

  Tabellen können in Andrew auch als Spreadsheets verwendet werden, so dass diese beiden Informationstypen keine grundlegenden Unterschiede aufweisen, sondern Spreadsheets eine Obermenge der von Tabellen angebotenen Funktionalität zur Verfügung stellen. Aus diesem Grund werden hier nur die über die normale Tabellenfunktionalität hinaushgehenden Aspekte der Spreadsheets erläutert, die anderen Eigenschaften sind unter dem Informationstyp Tabelle beschrieben.

  Innerhalb von Spreadsheets können Zellen verwendet werden, die Rechenvorschriften sowie Referenzen auf andere Zellen und Kombinationen von beiden enthalten können. Die Rechenvorschriften werden dynamisch ausgeführt, d.h. wird eine referenzierte Zelle modifiziert, ändern sich die Werte all der Zellen, in denen dieser Wert verwendet wird. Die Rechenvorschriften sind die einfachen arithmetischen Funktionen sowie eine Vielzahl anderer Funktionen und einfache bedingte Anweisungen. Ebenfalls können Funktionen auf beliebigen Bereichen von Zellen ausge-

führt werden, so können z.B alle Elemente einer Spalte addiert werden. Referenzen auf andere Zellen können absolut oder relativ angegeben werden

– Linienzeichnungen (Graphiken)

  Linienzeichnugen können verschiedene graphische Elemente enthalten, zur Auswahl stehen Textelemente, einfache Linien, Polylines, regelmässige Polygone, Rechtecke, Freihandkurven (*paths*), Insets (also beliebige Arten von ATK Objekten), Kreise, Ellipsen, Kurvenbögen, Rechtecke mit abgerundeten Ecken, Polylines mit Pfeilen sowie eine Menge von vordefinierten Symbolen. Diese Grundelemente können skaliert werden, so dass sich beliebige Varianten erzeugen lassen. Die geschlossenen Figuren können mit einem Füllmuster versehen werden, dass unter einer Auswahl von zehn Mustern gewählt werden kann.

  Die Graphik kann ausserdem mit einem Rasterbild als Hintergrund versehen werden, doch werden die Informationen nicht innerhalb des Fileformates abgespeichert, so dass bei erneutem Einlesen der Graphik der Hintergrund neu gesetzt werden muss. Ebenso beim Speichern verloren gehen Informationen über Skalierungen oder Ausschnittbestimmungen der ganzen Graphik, so dass immer die gesamte Graphik angezeigt wird.

– Mathematische Gleichungen

  Das ATK ermöglicht die Erstellung mathematischer Gleichungen mit Hilfe eines speziellen Formel-Editors, der verwendet wird, um die Gleichungen interaktiv zu erstellen. Dieser Informationstyp kann nicht verwendet werden, um Gleichungen auszuwerten (dazu können Spreadsheets benutzt werden), er dient nur der Darstellung von Formeln. Formeln können aus unterschiedlichen Komponenten zusammengesetzt werden, es gibt normale Zahlen und Buchstaben, mathematische Sonderzeichen (wie Element-Symbole oder logische Operatoren), mathematische Funktionen sowie spezielle Konstrukte, die ein besonderes Layout erfordern (wie Brüche oder Wurzeln).

– Rastergraphiken (bitmap images)

  Das ATK erlaubt lediglich die Erstellung und Manipulation von monochromen Rastergraphiken, d.h. es können werder Grauwerte noch Farbinformationen über Pixel gespeichert werden. Pixel befinden sich entweder im Zusatnd “gesetzt” oder “ungesetzt”, so dass die Speicherung auf Bit-Basis vorgenommen werden kann. Zur Verfügung stehen verschiedene Speicherformate, dies sind zwei interne Andrew Formate (BE2/ATK Raster Version 1 und BE2/ATK Raster Version 2), das MacPaint Format, als PostScript File, als X Window Dump oder als X Bitmap.

  Ebenso wie bei Linienzeichnungen werden Informationen über gewählte Vergrösserungen oder Verkleinerungen sowie Auschnittsbestimmungen nicht gespeichert , so dass immer das gesamte Rasterbild angezeigt wird. Damit müssen solche Manipulationen auf dem Rasterbild tatsächlich ausgeführt werden, um das Ergebnis in einem File abspeichern zu können, wenn dieses an anderer Stelle in manipulierter Form verwendet werden soll.

– Einfache Animationen

  Das ATK bietet mit Hilfe des *FAD (Framed Animation Drawing Tool)* die Möglichkeit, animierte Liniengraphiken zu erstellen und als Objekte zu speichern. Diese Objekte bestehen aus einzelnen Bildern (*Frames*), zwischen denen zur Präsentationszeit ein automatischer Uebergang erzeugt und angezeigt wird[14]. Als Elemente dieser Zeichnungen können Linien, Icons (kleine vordefinierte Rasterbilder), Label, Boxen und *Action Buttons* verwendet werden. Letztere dienen dazu, dem Benutzer zur Präsentationszeit Eingriffsmöglichkeiten zu geben.

  Die Animationen können gemäss der Andrew Architektur überall dort integriert werden, wo auch Objekte statischer Typen verwendet werden können. Bei der Präsentation eines Dokumentes ist es somit möglich, dynamische Objekte darzustellen. Bei der Darstellung durch Medien, die keine zeitabhängigen Präsentationen zulassen, also z.B. beim Drucken, wird das zuletzt selektierte Bild ausgegeben.

Es gibt also bereits eine reichhaltige Auswahl an Informationstypen innerhalb des ATK, die für den Editor verwendet werden können, ohne eigenständige Entwicklungen durchführen zu müssen.[15] Pro-

---

[14]Der Algorithmus zur Errechnung der generierten Zwischenbilder orientiert sich an der Reihenfolge, in der die einzelnen Elemente der Bilder gezeichnet wurde. Es kann deshalb dazu kommen, dass Teile eines Bildes bei der Animation plötzlich verschwinden und andere plötzlich erscheinen, was auf eine falsche Reihenfolge der gezeichneten Elemente schliessen lässt. Die Zwischenbilder werden bei jeder Präsentation neu bestimmt, d.h. es werden nur die vom Benutzer erstellten Bilder abgespeichert.

[15]Dabei gilt das oben gesagte, dass einige der Klassen (z.B. Tabellen bzw. Spreadsheets) in einem Zustand sind, der eine Neuimplementierung wünschenswert erscheinen lässt. Da es sich bei diesen Aufgaben um klar getrennte Stücke

blematisch ist einzig die Integration von dynamischen Aspekten, da es zwar einen zeitabhängigen Informationstypen innerhalb des ATK gibt, dieser aber nicht dazu verwendet werden kann, andere Objekte dynamisch zu präsentieren, sondern nur die in ihm selbst enthaltenen Informationen (Liniengraphiken).

10.2 Dokumentenmodell des ATK

Das ATK selbst definiert zunächst kein vollständiges Dokumentenmodell im Sinne dieses Begriffes, wie er in der Standardisierung verwendet wird. Fasst man diesen Begriff so auf, dass ein Modell sowohl die vorhandenen Informationstypen als auch die möglichen Relationen zwischen ihnen beschreiben muss, so wird innerhalb des ATK keine Aussage über den zweiten Aspekt gemacht, d.h. es gibt keinerlei strukturelle Einschränkungen. Die Objekte verschiedener Typen können beliebig kombiniert werden, ohne dass es eine Regelmenge oder Strukturbeschreibung gibt, die legale von illegalen Kombinationen unterscheidet und so eine generische Dokumentenstruktur vorgibt. Für den Programmierer ist diese Flexibilität sicher von Vorteil, für die Verwendung innerhalb eines Dokumentenverarbeitungssystemes muss sie jedoch eingeschränkt werden. Erst die Anwendung generischer Dokumentenbeschreibungen erlaubt es, Dokumente verschiedener Typen zu erstellen und die Erzeugung dieser Dokumente zu unterstützen, z.B. durch die Auflistung der an einer vorgegebenen Stelle zulässigen Objekte.

10.3 ATK und ODA

Ein erklärtes Ziel des MultimETH-Prototypen ist es, auf stabilen Standards aufzubauen, um so das grösstmögliche Mass an Flexibilität und Kompatibilität zu erreichen. Ein wichtiger Punkt kommt dabei auch dem verwendeten Dokumentenmodell zu. Welches Modell der Editor intern verwendet, ist in diesem Zusammenhang zunächst nicht von Interesse[16], vielmehr ist es wichtig, darauf zu achten, dass der Austausch von Dokumenten zwischen dem MultimETH Editor und anderen Editoren ermöglicht wird. Aus diesem Grund soll ein Backend existieren, das einen solchen Austausch sicherstellen kann.

Im EXPRES-Projekt, das in [Rosenberg et al 91] beschrieben ist, wurde die soeben beschriebene Fragestellung untersucht und auf eine Weise gelöst, die die Ergebnisse dieses Projektes äusserst nützlich erscheinen lässt. Problemstellung dieses Projektes war der Austausch von Multimedia-Dokumenten zwischen Applikationen, die völlig verschiedene Dokumentenmodelle verwenden. Bei den Dokumentenrepräsentationen handelt es sich dabei oft nahezu um Entsprechungen der in der Applikation verwendeten Datenstrukturen, die in verschiedenen Applikationen äusserst unterschiedlich sind. Um nicht für jede neue, zum Austauschszenario hinzukommende, Applikation Konvertierungen in alle anderen beteiligten Formate implementieren zu müssen, wurde ein neutrales Austauschformat gewählt, das mächtig genug ist, die relevanten Informationen zu übermitteln. Die Wahl fiel dabei, sowohl auf Grund der umfassenden Funktionalität als auch durch den Status als internationaler Standard, auf ODA.

Für alle im EXPRES-Projekt verwendeten Dokumentenmodelle wurden demzufolge Abbildungen von und nach ODA definiert und implementiert. Da auch das Andrew-System innerhalb des Projektes verwendet wurde, steht als Teilergebnis des Projektes somit Software zur Konvertierung von Andrew-Dokumenten von und nach ODA zur Verfügung. Die Verwendung dieser Konverter bietet sich damit an, da einerseits der erhebliche Aufwand selbst entwickelter Konverter vermieden werden kann, andererseits auch auf dem Gebiet der Dokumentenarchitektur damit ein internationaler Standard unterstützt wird, der zunehmend an Bedeutung gewinnt. Wie bereits erwähnt, ist die Konvertierung zwischen ODA und Andrew Dokumenten nicht ohne Informationsverlust möglich. Da beide Modelle Merkmale besitzen, die das jeweils andere nicht unterstützt, kann bei dieser Konvertierung der Informationsverlust sogar in beiden Richtungen der Umwandlung auftreten.

11 Literaturverzeichnis

[Appelt 90] Wolfgang Appelt, “Dokumentenaustausch in Offenen Systemen”, Springer-Verlag, Berlin, 1990.

---

im Verhältnis zum Editor handelt, käme eine Ausschreibung als Semester- oder Diplomarbeiten in Frage. Bis dahin muss die gewünschte Funktionalität der Klassen definiert sein, die im Augenblick noch ungeklärt ist.

[16]Natürlich gilt dies nur insofern, als das Austauschformat syntaktisch völlig unabhängig von der intern verwendeten Repräsentation ist. Es muss jedoch beachtet werden, dass wichtig ist, in welchem Verhältnis die Funktionalität beider Formate zueinander steht, da bei einem Missverhältnis die Konvertierung in Richtung auf das semantisch eingeschränktere Format immer mit einem Informationsverlust verbunden ist, der auch beim Uebergang in das mächtigere Format nicht rückgängig gemacht werden kann. In [Rosenberg et al 91] finden sich einige Beschreibungen dieser Probleme, die bei der Realisierung der EXPRES Software aufgetreten sind.

[Berlage 91] Thomas Berlage, “OSF/Motif: Concepts and Programming”, Addison-Wesley, Reading, Massachusetts, 1991.

[Borenstein 90] Nathaniel S. Borenstein, “Multimedia Applications Development with the Andrew Toolkit”, Prentice Hall, Englewood Cliffs, New Jersey, 1990

[Conklin 87] Jeffrey D. Conklin, “Hypertext: An Introduction and Survey”, IEEE Computer, Vol. 20, Nr. 9, September 1987, pp 17–41.

[Gerloff 88] Harald Gerloff, “Vergleich und Bewertung von Dokumentenarchitektur-Modellen für Multimedia-Dokumente”, ETH Zürich, Diplomarbeit, April 1988.

[ISO 10744] “Information technology – Hypermedia/Time-based Structuring Language (HyTime)”, International Organisation for Standardisation (ISO), 1991, ISO/DIS 10744

[ISO 7498] “Information processing systems – Open Systems Interconnection (OSI) – Basic Reference Model”, International Organisation for Standardisation (ISO), First Edition 1984, ISO/IS 7498

[ISO 8613] “Information processing – Text and Office Systems – Office Document Architecture (ODA) and Interchange Format”, International Organisation for Standardisation (ISO), 1989, ISO/IS 8613

[ISO 8632] “Information processing Systems – Computer Graphics – Metafile for the Storage and Transfer of Picture Description Information (CGM)”, International Organisation for Standardisation (ISO), 1987, ISO/IS 8632

[ISO 8649] “Information processing systems – Open Systems Interconnection (OSI) – Service Definition for the Association Control Service Element (ACSE)”, International Organisation for Standardisation (ISO), 1988, ISO/DIS 8649

[ISO 8650] “Information processing systems – Open Systems Interconnection (OSI) – Protocol specification for the Association Control Service Element (ACSE)”, International Organisation for Standardisation (ISO), 1988, ISO/DIS 8650

[ISO 8824] “Information processing Systems – Open Systems Interconnection (OSI) – Specification of Abstract Syntax Notation One (ASN.1)”, International Organisation for Standardisation (ISO), 1987, ISO/IS 8824

[ISO 8825] “Information processing Systems – Open Systems Interconnection (OSI) – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)”, International Organisation for Standardisation (ISO), 1987, ISO/IS 8825

[ISO 9066] “Information processing systems – Text Communication – Relieable Transfer”, International Organisation for Standardisation (ISO), 1987, ISO/DIS 9066

[ISO 9072] “Information processing systems – Text Communication – Remote Operations”, International Organisation for Standardisation (ISO), 1987, ISO/DIS 9072

[Lubich 90] Hannes P. Lubich, “MultimETH: Ein Beitrag zur Konzeption eines Echtzeit-Multimedia-Konferenzsystems”, Dissertation, ETH Zürich, 1990,

[Nelson 81] B. J. Nelson, “Remote Procedure Call”, PhD Thesis, Report number CMU-CS-81-119, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, 1981

[OSF 91] Open Software Foundation, “OSF/Motif Style Guide (Release 1.1)”, Prentice Hall, Englewood Cliffs, New Jersey, 1991

[Rosenberg et al 91] Jonathan Rosenberg, Mark Sherman, Ann Marks, Jaap Akkerhuis, “Multi-media Document Translation”, Springer-Verlag, New York, 1991.

[T.4] “Standardization of Group 3Facsimile Apparatus for Document Transmission”, CCITT Recommendation T.4, Melbourne, 1988 (Blue Book)

[T.6] “Facsimile Coding Schemes and Coding Control Functions for Group 4 Facsimile Apparatus”, CCITT Recommendation T.6, Melbourne, 1988 (Blue Book)

[X Series 1] Adrian Nye, "The X Window System Series Volume 1: Xlib Programming Manual", O'Reilly & Associates, Inc., Sebastapol, California, 1990

[X Series 2] "The X Window System Series Volume 2: Xlib Reference Manual", O'Reilly & Associates, Inc., Sebastapol, California, 1990

[X Series 3] Valerie Quercia, Tim O'Reilly, "The X Window System Series Volume 3: X Window System User's Guide", O'Reilly & Associates, Inc., Sebastapol, California, 1990

[X Series 4] Adrian Nye, Tim O'Reilly, "The X Window System Series Volume 4: X Toolkit Intrinsics Programming Manual", O'Reilly & Associates, Inc., Sebastapol, California, 1990

[X Series 5] "The X Window System Series Volume 5: X Toolkit Intrinsics Reference Manual", O'Reilly & Associates, Inc., Sebastapol, California, 1990

[X.200] "Reference Model of Open Systems Interconnection for CCITT Applications", CCITT Recommendation X.200, Melbourne, 1988 (Blue Book)

[X.208] "Specification of Abstract Syntax Notation One (ASN.1)", CCITT Recommendation X.208, Melbourne, 1988 (Blue Book)

[X.209] "Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)", CCITT Recommendation X.209, Melbourne, 1988 (Blue Book)

[X.217] "Association Control Service Definitions for Open Systems Interconnection for CCITT Applications", CCITT Recommendation X.217, Melbourne, 1988 (Blue Book)

[X.218] "Reliable Transfer: Model and Service Definition", CCITT Recommendation X.218, Melbourne, 1988 (Blue Book)

[X.219] "Remote Operations: Model, Notations and Service Definition", CCITT Recommendation X.219, Melbourne, 1988 (Blue Book)

[X.227] "Association Control Protocol Specification for Open Systems Interconnection for CCITT Applications", CCITT Recommendation X.227, Melbourne, 1988 (Blue Book)

[X.228] "Reliable Transfer: Protocol Specification", CCITT Recommendation X.229, Melbourne, 1988 (Blue Book)

[X.229] "Remote Operations: Protocol Specification", CCITT Recommendation X.229, Melbourne, 1988 (Blue Book)

12 Glossar

ACSE — Das *Association Control Service Element*, ein Service Element der Anwendungsschicht, dient dem Auf- und Abbau einer Anwendungs-Verbindung zwischen zwei OSI-Applikationen (einem *Initiator* und einem *Responder*), die über eine Schicht-6-Verbindung miteinander kommunizieren. Folglich enthält jede OSI-Applikation ein ACSE, da nur auf diese Weise eine Verbindung (*Association*) zwischen den *Application Entities (AEs)* hergestellt werden kann.

Andrew — Das Andrew System ist ein an der Carnegie Mellon University (CMU) entwickeltes System, das eine Programmier- und Arbeitsumgebung implementiert, die dem Benutzer ein reichhaltiges Repertoire an verfügbaren Komponenten zur Verfügung stellen soll. Das System besteht aus verschiedenen Elementen, einem Filesystem (AFS), einem Multimedia-Mailsystem (AMS), sowie einem Toolkit (ATK), das auf dem X Window System aufbaut.

Andrew Toolkit — Das Andrew Toolkit ist die Komponente des Andrew Systems, die es ermöglicht, auf bestehenden Klassen aufbauend auf verhältnismässig einfache Weise Multimedia-Applikationen zu gestalten. Es baut auf der Xlib auf und bietet eine objektorientierte Programmierumgebung, in der die vordefinierten Klassen verwendet oder durch eigene Definitionen ergänzt werden können.

ASN.1 — Die *Abstract Syntax Notation One*, die in [X.208] bzw. [ISO 8824] festgelegt ist, ermöglicht die Notation von komplexen Datentypen, ohne ihre Repräsentation festzulegen. Datentypen sind entweder *simple types* (wie Integers oder Strings) oder aus diesen zusammengesetzte *complex types*. Die Codierung von ASN.1-Objekten wird mit Hilfe von *Encoding Rules* durchgeführt, in [X.209] bzw. [ISO 8825] ist eine Menge von *Basic Encoding Rules* standardisiert.

Client — Im Zusammenhang mit Client-Server-Architekturen ist der Client diejenige Komponente, welche einen vom Server angebotenen Dienst in Anspruch nimmt. Diese Dienstanforderung besteht aus der Spezifikation eines konkreten Dienstes, den der Server entweder erbringen oder zurückweisen kann. Die Art der Kommunikation zwischen beiden Komponenten ist dabei unerheblich, die Schnittstelle zwischen beiden ist abstrakt definiert und erlaubt somit Implementierungen, die auf unterschiedlichen Mechanismen aufbauen können.

CSCW — Mit *Computer Supported Cooperative Work* wird das Forschungsgebiet bezeichnet, das sich mit der Untersuchung und Entwicklung von computerunter- und gestützten Hilfsmitteln für die Arbeit in Gruppen beschäftigt. CSCW ist eine interdisziplinäre Forschungsrichtung, in der Informatiker, Soziologen, Psychologen und Wissenschaftler weiterer Fachrichtungen in enger Zusammenarbeit versuchen, den Einsatz von Computern für Gruppen sinnvoll zu gestalten. Die Programme, die für CSCW-Zwecke erstellt werden, werden unter dem Begriff *Groupware* zusammengefasst.

E-Client — Der MultimETH-Editor ist – analog zum MultimETH-Modell – in zwei Komponenten unterteilt, die die Verteilung des Editors realisieren. Der E-Client implementiert die Benutzerschnittstelle des Editors und ist für die Koordination zwischen den Eingaben des lokalen Benutzers und den vom E-Server erhaltenen Miiteilungen zuständig. Im Gesamtsystem existieren so viele E-Clients, wie Benutzer an einem gemeinsamen Dokument editieren, jeder E-Client ist einem K-Client zugeordnet. Die Kombination aus E-Client und K-Client wird M-Client genannt.

E-K-Protokoll — Zwischen dem E-Server und dem K-Server ist ein Protokoll definiert, das den Austausch von Informationen ermöglicht, die für beide Systeme notwendig sind, wie z.B. Dokumenteninhalte oder Zugriffsberechtigungen für Benutzer. Das E-K-Protokoll wird innerhalb des M-Server verwendet und dient der Trennung von E- und K-Server.

E-Protokoll — Das Protokoll, das für die Kommunikation zwischen den verschiedenen Komponenten des verteilten Editors benutzt wird, trägt die Bezeichnung E-Protokoll. Es dient dem Zweck, die Konsistenz zwischen dem Datenbestand der verschiedenen Clients zu erhalten und Aenderungsanfragen, die von diesen gestellt werden, an die Teilnehmer des gemeinsamen Editiervorganges verteilen zu können. Das E-Protokoll ist eine Teilmenge des M-Protokolls, das dem gesamten Konferenzablauf festlegt und regelt.

E-Server — Der MultimETH-Editor ist – analog zum MultimETH-Modell – in zwei Komponenten unterteilt, die die Verteilung des Editors realisieren. Der E-Server ist die zentrale Instanz, die die parallele Arbeit mehrerer Benutzer an einem Dokument koordiniert und so dafür Sorge trägt, dass die Konsistenz des Dokumentes nicht verletzt wird. Jeder Benutzer wird für den E-Server durch einen E-Client repräsentiert, mit dem er über das E-Protokoll kommuniziert. Der E-Server ist eng mit dem K-Server verbunden, beide zusammen bilden den M-Server.

K-Client — Ein K-Client ist diejenige Komponente, die einen Benutzer innerhalb des MultimETH-Konferenzsystems repräsentiert. Er ist somit die Schnittstelle des Benutzers zum Konferenzsystem und kommuniziert direkt mit dem K-Server (und damit indirekt mit den anderen K-Clients), um die Konferenzfunktionalität anbieten zu können. Im Falle eines verteilten Editierprozesses existiert neben dem K-Client ein E-Client, der die Schnittstelle zwischen Benutzer und verteiltem Editor realisiert. Die Kombination aus K-Client und (optionalem) E-Client wird M-Client genannt.

K-Protokoll

K-Server — Der K-Server ist die Instanz des Konferenzsystems, die den Ablauf der Konferenzen steuert und als zentrale Komponente die Kommunikation zwischen allen K-Clients koordiniert. Zwischen K-Server und K-Clients wird das K-Protokoll verwendet, das den Ablauf der Kommunikation reglementiert. Findet neben der Konferenztätigkeit ebenso ein verteilter Editiervorgang statt, so existiert neben dem K-Server ein E-Server, der für die zentrale Instanz für den Editiervorgang darstellt. Die Kombination aus K-Server und (optionalem) E-Server wird M-Server genannt.

LittlemETH — Ausgehend von der Definition eines Konferenzsystems, wie sie durch MultimETH gegeben wird, existiert eine Implementierung einer Untermenge dieses Systems, der LittlemETH Prototyp. In der derzeitigen Version ist er eine teilweise Umsetzung des Konferenzsystems, ohne jedoch den Multimedia-Editor zu enthalten. Weitere Arbeiten im MultimETH-Projekt sollen zu einer stärkeren Annäherung zwischen der Definition und dem Prototypen führen.

M-Client

Mehrbenutzerbetrieb — Konventionelle Applikationen sind so gestaltet, dass sie zu einem Zeitpunkt von einem Benutzer verwendet werden können. Die gemeinsame Arbeit mit einer Applikation ist somit nicht möglich. Mehrbenutzerunterstützung bedeutet, dass mehrere Anwender gleichzeitig mit der Applikation arbeiten können und der nebenläufige Zugriff auf einen gemeinsamen Datenbestand so koordiniert wird, dass eine sinnvolle, rechnergestützte Gruppenarbeit zustande kommt.

M-Protokoll

M-Server

Multimedia — Unter dem Begriff Multimedia werden recht verschiedene Dinge verstanden, je nachdem, in welchem Rahmen er verwendet wird. Die Auffassungen reichen von unterschiedlichen Informationstypen (Text, Graphik, Rasterbildert, usw.), dies ist die in diesem Papier verwendete Definition, bis zu unterschiedlichen Medien-Typen, also visuell, akustisch, taktil oder anderes. Während die erste Auffassung im Bereich der Informationsverarbeitung bereits weite Verbreitung gefunden hat, sind Medientypen, die sich von visuellen oder akustischen Repräsentationen unterscheiden noch in der Anfangsphase ihrer Entwicklung.

MultimETH

ROSE — Das *Remote Operations Service Element*, ein Service Element der Anwendungsschicht, dient der Ausführung von Remote Operations (die als grundlage für einen *Remote Procedure Call* verwendet werden können) in einer OSI-Umgebung. Ein entfernter Service wird von einem *Invoker* aufgerufen, der als vom Kommunikationspartner, dem *Responder*, übermittelte Antwort entweder ein Resultat (*Result*), einen Fehler (*Error*) oder eine Zurückweisung (*Rejection*) erhält.

Server — Im Zusammenhang mit Client-Server-Architekturen ist der Server diejenige Komponente, welche einen vom Client angeforderten Dienst erbringt oder zurückweist. Der Server bietet einen abstrakten Dienst an, der von Clients in verschiedenen konkreten Ausprägungen in Anspruch genommen werden kann. Die Art der Kommunikation zwischen beiden Komponenten ist dabei unerheblich, die Schnittstelle zwischen beiden ist abstrakt definiert und erlaubt somit Implementierungen, die auf unterschiedlichen Mechanismen aufbauen können.

Toolkit — Ein Toolkit ist in der Terminologie von X eine Sammlung von Prozeduren, die auf der Xlib aufsetzen und dem Programmierer eine abstraktere Sicht ermöglichen als den direkten Umgang mit dem X Protocol. Neben diesem Vorteil liegt die Idee der Toolkits darin, dass die einheitliche Benutzung eines Toolkits durch mehrere Applikationen dazu führt, dass bestimmte Design-Prinzipien für X Clients gewahrt bleiben, wenn diese innerhalb des Toolkits implementiert sind.

Widget Set — Innerhalb des X Toolkit, eines bestimmten, auf der Xlib aufbauenden Toolkit, gibt es verschiedene Widget Sets, die dazu verwendet werden, unterschiedliche Arten von Benutzerschnittstellen (verschiedene *Look and Feel*) zu implementieren. Ein Widget ist ein X Window mit vordefiniertem Verhalten, das – in Kombination mit anderen – als Baustein verwendet werden kann, um einen bestimmten Look and Feel zu erreichen. Die X Toolkit Widget Sets bauen auf den *Xt Intrinsics* auf, die die Basis für alle Widget Sets bilden. Bekannte Widget Sets sind das OSF/Motif Widget Set sowie das Athena Widget Set.

X Client — In der X Terminologie ist ein X Client eine Applikation, die den Dienst eines X Servers, des graphischen Terminals, in Anspruch nimmt. Zwischen beiden Komponenten wird das X Protocol verwendet, das es ermöglicht, Client und Server räumlich getrennt und durch ein Rechnernetz verbunden zu betreiben. Ein X Client kann Verbindungen zu mehreren Servern haben, um Ausgaben auf unterschiedlichen Terminals vorzunehmen.

Xlib — Die Xlib ist die am nächsten am X Protocol liegende Schnittstelle des X Window Systems, auf die eine Applikation entweder direkt, oder unter Verwendung eines Toolkits aufsetzen kann. Diese Schnittstelle ist für verschiedene Sprachen standardisiert, kann also von Applikationen auf unterschiedlichen Rechnertypen konsistent verwendet werden. Damit wird erreicht, dass die Portabilität von auf der Xlib aufsetzender Software sichergestellt ist.

X Server — Der X Server ist im X Window System diejenige Komponente, die es Clients gestattet, geräteunabhängig graphische Ausgaben vorzunehmen. Der Server wird über ein definiertes Protokoll, das *X Protocol*, angesprochen, womit gewährleistet ist, das der Zugriff auch über ein Kommunikationsnetz geschehen kann. Der Server interpretiert die Anfragen und setzt sie in gerätespezifische Anweisungen um, die das mit dem Server verbundene Ausgabegerät ansteuern. Die mit einem Server verbundenen Eingabegeräte (Tastatur, Maus) generieren *Events*, die als Meldungen von Benutzeraktionen an die entsprechenden Clients übertragen werden.

X Window System — Das X Window System, teilweise auch mit X11 bezeichnet, ist ein verteiltes Fenstersystem, das es gestattet, Applikationen und ihre Ausgaben auf einem graphischen Terminal räumlich zu trennen. Erreicht wird diese Trennung durch eine definierte Schnittstelle zwischen diesen Komponenten, das *X Protocol*. X ist unabhängig von bestimmten Ausgabegeräten und bildet deshalb eine gute Grundlage für ein verteiltes Fenstersystem in einer heterogenen Rechnerumgebung.

13 Index

Andrew 4, 21
Andrew Toolkit 2, 3, 21
    Informationstypen 21
    Insets 21
Association Control Service Element 19
Computer Graphics Metafile 17
EXPRES 23
Groupware 26
ISODE 7
LittlemETH 1
MacPaint 22
Motif 2, 3, 4, 21
    Compliance Conventions 2
MultimETH 1, 3, 4, 7, 8, 23
ODA 4, 5, 17, 23
    Hyper-ODA 5
    ODA/ATK Konversion 23
Open Systems Interconnection 7
    Association Control Service 7
    Reliable Transfer Service 7
    Remote Operations Service 7
PostScript 22
Reliable Transfer Service Element 20
Remote Operations Service Element 19
SunView 3, 4
User Interface 2
X Window System 2, 21
    Toolkits 21
        InterViews 21
        Xt Toolkit 2, 21
        XView 21
    X Bitmap 22
    X Protocol 21

X Window Dump 22
Xlib 2