Aus dem
Institut für Medizinische Informatik und Biomathematik
der Westfälischen Wilhelms-Universität Münster
- Direktor: Prof. Dr. W. Köpcke -

Design und Implementierung einer grafischen Benutzeroberfläche zur Verwaltung einer Datenbank für multimediale medizinische Objekte auf Basis XML-codierter Inhalte

INAUGURAL - DISSERTATION
zur Erlangung des
doctor rerum medicinalium

der Medizinischen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von Markus Ruppel
aus Münster
2003

Gedruckt mit Genehmigung der Medizinischen Fakultät der Westfälischen Wilhelms-Universität Münster

Dekan: Prof. Dr. H. Jürgens

1. Berichterstatter: Prof. Dr. W. Köpcke

2. Berichterstatter: PD Dr. N. Roeder

Tag der mündlichen Prüfung: 19.08.2003

Aus dem
Institut für Medizinische Informatik und Biomathematik
der Westfälischen Wilhelms-Universität Münster
- Direktor: Prof. Dr. W. Köpcke -
Referent: Prof. Dr. W. Köpcke
Koreferent: PD Dr. N. Roeder

Zusammenfassung

Design und Implementierung einer grafischen Benutzeroberfläche zur Verwaltung einer Datenbank für multimediale medizinische Objekte auf Basis XML-codierter Inhalte

Markus Ruppel

Der zunehmende Einsatz von Computersystemen in der Medizin führt insbesondere in den Bereichen Forschung und Lehre zu einer vermehrten Verwendung von multimedialen Objekten. Diese werden in den meisten Fällen nur ein einziges Mal und für einen bestimmten Zweck verwendet, da keine übergreifende Archivierung vorgenommen wird. Um Redundanzen bzgl. der Herstellung solcher Objekte vermeiden zu können, wird am Institut für Medizinische Informatik und Biomathematik an der WWU Münster ein Datenbanksystem (MMO-System) zur Verwaltung multimedialer medizinischer Objekte (MOBs) entwickelt. Dieses soll als Archiv zur Speicherung, zum Austausch und zur multiplen Verwendung (z.B. in Printmedien, CD-Roms, Präsentationsfolien, ...) verschiedenartiger Multimediadateien dienen. Die Anforderungen an dieses System beinhalten u.a. die Möglichkeit zur effizienten Suche nach geeigneten Objekten, sowie Funktionen zur automatischen Erstellung interaktiver virtueller Lehreinheiten unter Verwendung der gespeicherten Inhalte. In diesem Zusammenhang wurden ein semantisches Netzwerk zur Verknüpfung der Multimediaobjekte sowie Funktionen zur Verschlagwortung der Objekte über den UMLS Meta-Thesaurus entwickelt.
Um dem medizinischen Anwender ein sinnvolles Arbeiten mit dem MMO-System zu ermöglichen, muß eine komfortable Benutzeroberfläche bereitgestellt werden, über die die verschiedenen Funktionen des Systems aufgerufen werden können. In dieser Arbeit wird das Design und die Implementierung einer solchen Oberfläche (MMO-Client) beschrieben. Diese wurde als eigenständige Applikation für Microsoft Windows Betriebssysteme entwickelt und kommuniziert über eine Internetschnittstelle mit einem Backend-System, welches für die zentralisierte, physikalische Verwaltung der Daten zuständig ist. Dabei werden die Daten in der Objektbeschreibungssprache XML übertragen. Nach einer eingehenden, allgemeinen Beschreibung der Funktionen des MMO-Clients, werden die für die Verarbeitung der Multimediaobjekte verwendeten XML-Strukturen erläutert. Anschließend wird der MMO-Client an sich vorgestellt, wobei erörtert wird, wie die zuvor beschriebenen Funktionen konkret umgesetzt wurden.

Inhalt

1. Einleitung

Die rasante Entwicklung im Bereich der Computertechnologie während der letzten Jahre hat dazu geführt, daß Computer sowohl im privaten als auch im beruflichen Umfeld mehr und mehr zum Einsatz kommen. Die stetige Steigerung der Leistungsfähigkeit und der Benutzerfreundlichkeit der Systeme, die kontinuierlich fallenden Preise und die zunehmende Akzeptanz inter- und intranetbasierter Dienste führen zu einer zunehmenden Informationsverbreitung auf Basis multimedialer Daten. Diese Tatsache spiegelt sich auch – oder besser: besonders – im Bereich der Medizin wider. Sowohl in der klinischen Praxis als auch in Bereichen wie Forschung und Lehre nimmt die Nutzung von Multimedia zu.

Besonders für die Bereiche Forschung und Lehre stellt sich in diesem Zusammenhang die Frage, welche multimedialen Objekte für die Zusammenstellung von Lehr- und Forschungsmaterialien zur Verfügung stehen. Die meisten dieser Objekte werden bisher durch Mitarbeiter der jeweiligen Institutionen erstellt und in einem lokalen Dateisystem abgespeichert, dessen Zugriff im allgemeinen nur wenigen ausgewählten Institutsangehörigen vorbehalten ist. Auf diese Weise entstehen viele separate Multimediasammlungen, die jeweils nur einem stark eingeschränkten Personenkreis zur Verfügung stehen und oft nur ein einziges Mal und für einen bestimmten Zweck verwendet werden, da keine übergreifende Archivierung vorgenommen wird. Dies hat zur Folge, daß oft aufwendige Arbeitsschritte zur Entwicklung multimedialer Inhalte wiederholt werden müssen, obwohl möglicherweise bereits irgendwo geeignete Objekte vorhanden sind.
Derartige Redundanzen können durch die Entwicklung einer zentralen Plattform zur Verwaltung multimedialer Daten vermieden werden. Über eine solche Plattform können Multimediadateien einem definierbaren Personenkreis zur Verfügung gestellt werden, um diese in verschiedenen Medien (z.B. Printmedien, CD-Roms, Präsentationsfolien, ...) wiederverwerten zu können.

Grundsätzlich bietet das Internet eine geeignete technische Basis für die Entwicklung eines solchen Systems. Insbesondere im World Wide Web, dem bekanntesten aller Internetdienste, werden bereits einige medizinische Multimediadatenbanken bereitgestellt. Diese bieten zwar die Möglichkeit, multimediale Objekte (zumeist Bilddateien) herunterzuladen, allerdings ist es i.a. nicht möglich, eigene Objekte in diesen Systemen zu verwalten.
Darüber hinaus ist es schwierig, die verschiedenen Datenbanken in der Informationsflut des Internets überhaupt zu finden. Und selbst wenn man sie gefunden hat, so stellt sich dem Anwender die Aufgabe, sich mit den verschiedenen, individuell konstruierten Benutzeroberflächen der einzelnen Systeme zurechtzufinden. Dieses Problem wird verstärkt durch eine oftmals unzureichende Benutzerfreundlichkeit der Oberflächen, die auf technische Einschränkungen der gängigen Webbrowser zurückzuführen ist.
Des weiteren wird bei den meisten Systemen auf die Verwendung eines vorgegebenen medizinischen Vokabulars (z.B. ICD, SNOMED, UMLS, ...) zur Verschlagwortung der multimedialen Inhalte verzichtet. Hieraus ergeben sich Schwierigkeiten beim Absuchen der erfaßten Datenbestände, da der Benutzer entweder gar keine Möglichkeit zur Suche nach medizinischen Fachbegriffen hat, oder eine Stichwortsuche nur dann erfolgreich ist, wenn die Suchbegriffe mit den exakten Formulierungen des Autors der Multimediadatei übereinstimmen.

Vor diesem Hintergrund wird am Institut für Medizinische Informatik und Biomathematik an der WWU Münster ein Datenbanksystem zur Verwaltung multimedialer medizinischer Objekte (MMO-Datenbank) entwickelt, welches als Archiv zur Speicherung, zum Austausch und zur multiplen Verwendung verschiedenartiger Multimediaobjekte dienen soll. Diese Datenbank besitzt eine Internetanbindung, über die der Datenzugriff durch externe Software ermöglicht wird. Dabei werden die Ein- und Ausgaben in der layoutunabhängigen Objektbeschreibungssprache XML übertragen.

Zur Verwaltung, Recherche und Wiederverwendung multimedialer medizinischer Objekte mit diesem System, muß dem medizinischen Anwender eine benutzerfreundliche Oberfläche zur Verfügung gestellt werden, dessen Funktionen die Erfüllung der folgenden Zielsetzungen sicherstellen:

- Gewährleistung einer übersichtlichen Darstellung und Verwaltung der Objekte und ihrer Meta-Informationen
- Personenbezogene Kontrolle des Zugriffs auf die Objekte
- Effiziente Mechanismen zum Auffinden relevanter Objekte
- Mechanismen zur Verwaltung gefundener Objekte
- Möglichkeit zur flexiblen Erzeugung interaktiver, virtueller Lehreinheiten

Das Design und die Implementierung einer solchen Oberfläche ist das Ziel dieser Arbeit.

2. Grundlagen

In diesem Kapitel werden grundlegende Technologien und Konzepte erörtert, die für das weitere Verständnis dieser Arbeit notwendig sind. Dabei werden sowohl das Unified Medical Language System als auch ein Prototyp des MMO-Datenbanksystems vorgestellt.

2.1. Das Unified Medical Language System

Das Unified Medical Language System (UMLS) [9,33,58,59] ist eine umfassende elektronische „Wissensquelle“, die seit 1986 als interdisziplinäres Projekt unter Leitung der US National Library of Medicine entwickelt wird und in drei (ehemals vier) Bereiche unterteilt werden kann. Zwei dieser Bereiche sind der „UMLS Meta-Thesaurus“ und das „UMLS Semantic Network“.

2.1.1. Der Meta-Thesaurus und seine Einteilung in Konzepte

Der UMLS Meta-Thesaurus ist ein Zusammenschluß von über 60 medizinischen Wörterbüchern und Klassifikationen, der inzwischen ca. 1,9 Millionen Begriffe umfaßt. Zusätzlich zu den eigentlichen Wortbeständen werden Informationen bzgl. der Beziehungen und Hierarchien aus den Quell-Vokabularen übernommen. Durch diese Zusammenführung ist es unvermeidlich, daß Begriffe erfaßt werden, die zwar unterschiedlich geschrieben werden, aber inhaltlich das gleiche ausdrücken. Da kontrollierte medizinische Vokabulare dazu verwendet werden, medizinische Entitäten durch eine standardisierte linguistische Basis zu beschreiben, ist es notwendig, diese begrifflichen Überschneidungen adäquat zu berücksichtigen.
Im UMLS Meta-Thesaurus wird dies durch die Organisation der Begriffe in Form von Konzepten (Concepts) realisiert [10]. Hierbei enthält jedes Konzept eine Menge von Begriffen, die inhaltlich die gleiche Bedeutung haben. Dabei werden Synonyme ebenso erfaßt, wie lexikographische Varianten und verschiedene Sprachversionen eines Begriffs. Jeweils einer der in einem Konzept enthaltenen Begriffe wird als bevorzugte Bezeichnung des jeweiligen Konzepts definiert.

Derzeit sind ca. 800.000 verschiedene Konzepte im Meta-Thesaurus erfaßt, wobei jedes durch eine eindeutige Kennziffer, einem sog. „Concept Unique Identifier“ (CUI), identifiziert werden kann. Solch ein CUI besteht aus dem Zeichen „C“ gefolgt von einer siebenstelligen Ziffernfolge. Diese Konzepte können u.a. dazu verwendet werden, Objekte in Multimediadatenbanken mit repräsentativen Beschreibungen zu versehen. Da Multimediadateien im allgemeinen nicht aus reinem Text bestehen, kann eine Datenbanksuche nur anhand zugewiesener Schlagworte vorgenommen werden. Eine Verschlagwortung auf Freitextbasis führt jedoch nur zum Erfolg, wenn die Suchbegriffe exakt mit den Formulierungen der Schlagworte übereinstimmen [25], weshalb sich die Verschlagwortung auf Basis eines kontrollierten Vokabulars anbietet [3,28,38]. Durch die Verwendung der CUIs des UMLS Meta-Thesaurus anstelle von Freitext-Schlagworten kann nach dem tatsächlichen semantischen Sinn eines Begriffs gesucht werden, wobei von der exakten Schreibweise abstrahiert werden kann. Dieses Prinzip soll anhand eines kleinen Beispiels, das in Abb. 1 skizziert wird, erläutert werden.

Der Arzt A möchte eine angiografische Aufnahme einer pathologisch verengten Speiseröhre in der institutseigenen Multimediadatenbank ablegen, um sie auch seinen Kollegen zur Verfügung zu stellen. Das Datenbanksystem sieht eine Verschlagwortung auf Freitextbasis vor, so daß A seinem Bild die Begriffe „Stenose“ und „Oesophagus“ zuweist.

Die Kollegen B und C, die nichts von der Existenz von A’s Bild wissen, suchen nach einer solchen Abbildung, um sie in ihr Vorlesungsmanuskript einzubinden.

Auf der Suche nach geeignetem Bildmaterial befragen B und C auch die Multimediadatenbank. Da B, ein Gastdozent aus dem Ausland, der deutschen Sprache nicht mächtig ist, sucht er nach dem englischen Begriff „Stenosis“. Hingegen verwendet Kollege C bevorzugt deutschsprachige Begriffe und gibt daher „Speiseröhre“ als Suchwort ein. Enttäuscht stellen B und C fest, daß ihre Suche zu keinem Ergebnis führt.

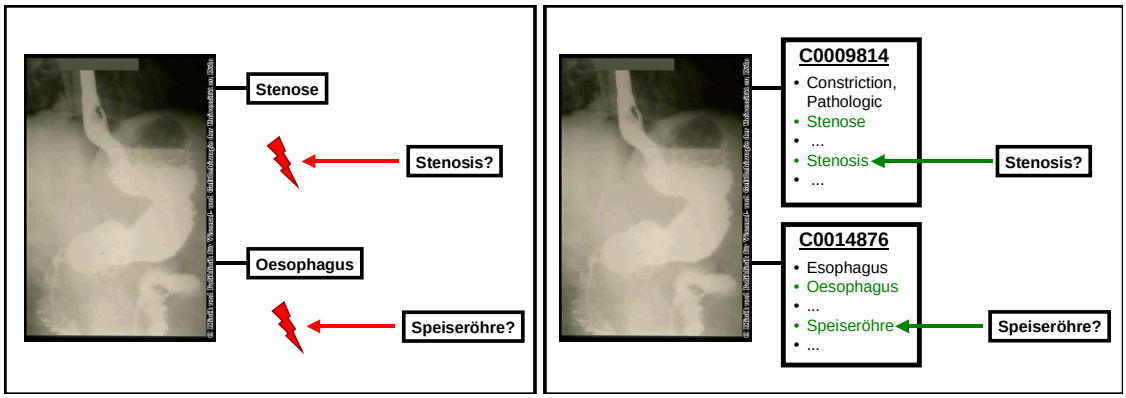


Abb. 1: Freitext- vs. Konzeptsuche
Links: Die Suche nach Freitext führt oft nicht zum Erfolg.
Rechts: Durch die Suche nach Konzepten kann von Sprache und exakten Formulierungen abstrahiert werden.

Obwohl das Bild von A durchaus relevant gewesen wäre, konnte es nicht gefunden werden, da der Erfolg der Suche in diesem Fall von der **Formulierung des Suchbegriffs** („Stenose“ ≠ „Stenosis“, „Oesophagus“ ≠ „Speiseröhre“) und nicht von seiner **semantischen Bedeutung** abhängt (vgl. Abb. 1, links).

Dieses Dilemma kann durch die Verwendung von UMLS Konzepten anstelle von Freitext-Schlagworten vermieden werden, wie die folgende Erweiterung des Beispiels zeigt.

> *Durch eine Erweiterung des Datenbanksystems, die der Student S während eines Praktikums entwickelt hat, ist es möglich, den Multimediadateien UMLS Konzepte anstelle der bisher üblichen Freitext-Schlagworte zuzuordnen. Über einen von S bereitgestellten Mechanismus zur Suche im UMLS Meta-Thesaurus, findet A schnell heraus, daß die Konzepte zu den Begriffen „Stenose“ und „Oesophagus“ im Meta-Thesaurus durch die CUIs C0009814 und C0014876 repräsentiert werden, die er seinem Bild anstelle der bisherigen Freitexte zuweist.*
>
> *Aufgrund der von S entwickelten technischen Neuerung, die sich im Institut schnell herumspricht, entschließen sich B und C zu einer*

erneuten Datenbanksuche. In der Hoffnung, doch noch fündig zu werden, suchen sie zunächst im UMLS nach den Konzepten, die ihren Suchbegriffen „Stenosis“ und „Speiseröhre“ entsprechen.
Da in einem Konzept alle Begriffe erfaßt werden, die inhaltlich die gleiche Bedeutung haben, stoßen auch B und C bei ihrer Konzeptsuche auf die CUIs C0009814 und C0014876. Diese CUIs werden nun für die eigentliche Suche in der Multimediadatenbank verwendet. Aufgrund der Übereinstimmung der Suchkonzepte mit den CUIs, die A seinem Bild zugewiesen hat, führt die Suche in diesem Fall zum Erfolg, so daß B und C die Abbildung in ihrem Skript verwenden können (vgl. Abb. 1, rechts).

Dieses Beispiel zeigt, wie mit der Verwendung von UMLS Konzepten Suchmechanismen entwickelt werden können, die sich nicht an der exakten Formulierung, sondern an der tatsächlichen semantischen Bedeutung eines Suchbegriffs orientieren.

2.1.2. Semantische Vernetzungen im UMLS

Wie bereits erwähnt, ist das „UMLS Semantic Network“ eines der drei Bereiche innerhalb der gesamten UMLS „Wissensquelle“. Diese Klassifizierung legt die Vermutung nahe, daß es im UMLS genau ein, bzw. **das** semantische Netzwerk gibt. Diese Aussage ist in der Form jedoch nicht korrekt, da es eigentlich zwei voneinander unabhängige semantische Netze gibt. Neben dem oben genannten „UMLS Semantic Network“, das im folgenden als das **erweiterte semantische Netzwerk** bezeichnet wird, gibt es ein einfacheres semantisches Netz innerhalb des UMLS Meta-Thesaurus. Letzteres wird im weiteren Verlauf dieser Arbeit als das **einfache semantische Netzwerk** bezeichnet. Abb. 2 zeigt die Zusammenhänge und Strukturen der semantischen Netze des UMLS.

Im vorhergehenden Abschnitt wurde erklärt, daß die Begriffe im UMLS Meta-Thesaurus als semantische Entitäten in Form von Konzepten organisiert werden. Diese Konzepte bilden die Knoten des **einfachen semantischen**

Netzwerks, die mit Hilfe verschiedener Relationen miteinander verknüpft werden (vgl. Abb. 2 unten). Diese Relationen dienen zur Beschreibung der verwandtschaftlichen Beziehung zwischen den Konzepten und wurden zu einem großen Teil bei der Zusammenführung der verschiedenen Quell-Vokabulare in den Meta-Thesaurus übernommen. Zusätzlich wurden weitere Relationen und Verknüpfungen von den Entwicklern des UMLS in das System integriert. Im einfachen semantischen Netz stehen elf Relationen zur Verfügung:

- *Broader (RB)* „... hat eine erweitere Beziehung zu ...“
- *Narrower (RN)* „... hat eine engere Beziehung zu ...“
- *Other related (RO)* „... hat Beziehung, die nicht RB oder RO ist ...“
- *Like (RL)* „... ist gleich oder sehr ähnlich ...“
- *RQ* „... ist verwandt und vermutlich synonym ...“
- *SY* „... is source asserted synonym ...“
- *Parent (PAR)* „... ist hierarchisch übergeordnet ...“
- *Child (CHD)* „... ist hierarchisch untergeordnet ...“
- *Sibling (SIB)* „... ist hierarchisch auf gleicher Ebene unter gemeinsamen Oberknoten ...“
- *AQ* „... is an allowed qualifier for a concept in a Metathesaurus source vocabulary ...“
- *QB* „... can be qualified by a concept in a Metathesaurus source vocabulary ...“

Optional können die Verknüpfungen mit einem zusätzlichen Verknüpfungsattribut versehen werden, über das die Beziehung zwischen den Konzepten konkreter beschrieben werden kann. Dabei stammt ein Verknüpfungsattribut entweder aus den „semantic relationships“ des erweiterten semantischen Netzwerks (s.u.) oder aus einer vorgegebenen Liste weiterer ausgewählter Relationen aus den Quell-Vokabularen des Meta-Thesaurus. Bisher wurde in den meisten Fällen auf die Verwendung von

Verknüpfungsattributen verzichtet, allerdings sollen diese in Zukunft häufiger zum Einsatz kommen.

Im **erweiterten semantischen Netzwerk** werden verschiedene semantische Kategorien (sog. „semantic types“, z.Zt. 134 Stück) und semantische Relationen (sog. „semantic relationships“, z.Zt. 54 Stück) definiert (vgl. Abb. 2, oben). Die Kategorien sind die Knoten in diesem semantischen Netz, die über die „semantic relationships“ miteinander verknüpft werden. Dabei ist eine der 54 im System erfaßten Relationen von besonderer Bedeutung: Über die „ist ein“-Beziehung wird eine grundsätzliche Organisation der Kategorien in einer hierarchischen Baumstruktur realisiert.

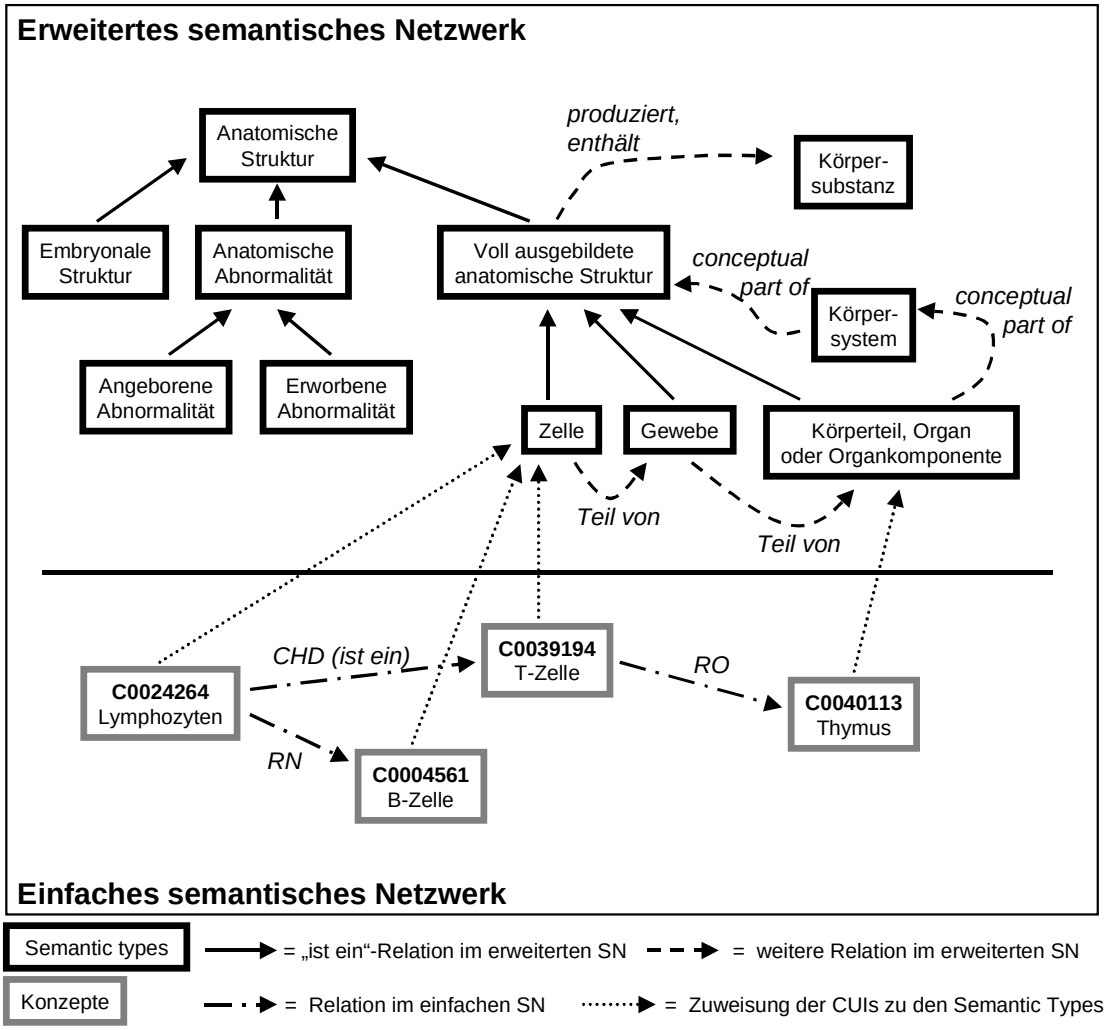


Abb. 2: Auszug aus den semantischen Netzwerken des UMLS

Jenseits dieser Hierarchie können die Kategorien über weitere Relationen, wie z.B. „ist Teil von“, „produziert“ etc., verknüpft werden.
Jedes Konzept des Meta-Thesaurus wird mindestens einer dieser Kategorien zugeordnet, wobei stets versucht wird, eine Zuweisung zu einer möglichst speziellen Kategorie innerhalb des Hierarchiebaums zu realisieren. So wird beispielsweise das Konzept „T-Zelle“ (C0039194) nicht der semantischen Kategorie „Voll ausgebildete anatomische Struktur“, sondern der spezielleren untergeordneten Kategorie „Zelle“ zugewiesen (vgl. Abb. 2).
Darüber hinaus sollte erwähnt werden, daß eine verwandtschaftliche Beziehung zwischen zwei semantic types A und B nicht automatisch bedeutet, daß diese Beziehung auch zwischen allen Konzepten aus A und B besteht. Beispielsweise sind die Kategorien „Anzeichen und Symptome“ und „Attribute des Organismus“ über die Relation „ist Bewertung von“ verknüpft. Zwar sind Anzeichen „Übergewicht“ (C0028754) und „Fieber“ (C0015967) Bewertungen der Organismusattribute „Körpergewicht“ (C0005910) und „Körpertemperatur“ (C0005903), jedoch kann mit dem Konzept „Fieber“ nicht das „Körpergewicht“ bewertet werden.

Die beiden beschriebenen semantischen Netzwerke des UMLS können dazu verwendet werden, konzeptbasierte Suchmechanismen, wie sie in Abschnitt 2.1.1 vorgestellt wurden, mit einer Unschärfe zu versehen. Das bedeutet, daß neben den eigentlichen Suchkonzepten auch verwandte Begriffe automatisch in die Suche einbezogen werden können [28].
Als Beispiel soll das Konzept „T-Zelle“ (C0039194) als Suchkonzept betrachtet werden. Eine unscharfe Suche könnte realisiert werden, indem alle Konzepte, die im einfachen semantischen Netz vermöge der Relation „Child“ (CHD) mit diesem Konzept verknüpft sind, ebenfalls als Suchkonzepte betrachtet werden.
Auf diese Weise würden automatisch auch alle Suchergebnisse für die Konzepte „T-Killer-Zelle“ (C0022686), „T-Helfer-Zelle“ (C0018894) etc. angezeigt werden.

2.2. Ein Java-basierter Prototyp des MMO-Datenbanksystems

Am Institut für Medizinische Informatik und Biomathematik der WWU Münster wird derzeit ein Datenbanksystem entwickelt wird, das als Grundlage zur Entwicklung einer Lösung für die Zielsetzungen dieser Arbeit dient. Ein erster, funktional eingeschränkter Prototyp wurde in [25,27] vorgestellt. Dieses System ermöglicht es, multimediale Objekte oder deren Internetadressen in einer Datenbank zu verwalten.

Aus technischer Sicht wurde das System vollständig in Java in Verbindung mit einer ORACLE 8 Datenbank umgesetzt. Konkret wurden ein Java-Applet und ein Java-Servlet programmiert, wobei das Applet, das auf dem Webbrowser des Anwenders ausgeführt wird, die grafische Benutzerschnittstelle zum Absuchen und Auslesen der Datenbank darstellt.

Für jedes Objekt in der Datenbank können zusätzliche Meta-Informationen sowie Stichworte aus einem kontrollierten, in das System integrierte Vokabular angegeben werden. Dieses Vokabular wurde mit einem semantischen Netz versehen, welches eine übersichtliche Darstellung der Suchergebnisse innerhalb des Applets ermöglicht.

Das Absuchen der Datenbank wurde folgendermaßen umgesetzt: Zunächst kann der Benutzer ein Stichwort in ein Textfeld eingeben und es (über das Servlet, s.u.) an die Datenbank schicken. Daraufhin werden die im Thesaurus gefundenen Begriffe in einer hierarchischen Baumstruktur im Applet dargestellt. Hierfür wurde eine Java-Komponente verwendet, dessen Aufbau an den eines Dateibaums, wie er im Windows Explorer der Microsoft-Betriebssysteme verwendet wird, erinnert. Durch entsprechende Symbole vor den Einträgen ist es möglich, gezielt beliebige Pfade ein- und auszublenden. Auf diese Weise wird eine übersichtliche, komfortable Navigation im semantischen Netz ermöglicht. Durch Auswahl eines geeigneten Eintrags in der Baumstruktur werden nähere Informationen zu dem damit verbundenen Objekt, sowie das Objekt selbst aus der Datenbank geladen und im Webbrowser dargestellt.

Die zweite Komponente des Systems, das Servlet, nimmt die Anfragen des Applets über eine http-Verbindung entgegen und stellt die eigentliche

Verbindung zur Datenbank her. Lediglich in diesem Servlet werden die notwendigen SQL-Statements sowie die für den Datenbankzugriff nötigen Anmeldedaten verwaltet und auf die Datenbank angewendet. Das Servlet fungiert als eine Art „black box“, die über entsprechende Parameter Befehle entgegennehmen und verarbeiten kann. Die Ergebnisse werden entweder in XML [24] oder, sofern es sich um binäre Daten handelt, als Byte-Array an das Applet zurückgeliefert.

Durch diese Vorgehensweise ist es möglich, die Entwicklung von Benutzerschnittstelle und Datenbankapplikation zu entkoppeln. Insbesondere könnten mehrere alternative Benutzerschnittstellen entwickelt werden, die alle über die definierte Schnittstelle auf das Servlet zugreifen.

3. Methoden

In diesem Kapitel wird zunächst die Grundstruktur des Gesamtsystems (bestehend aus Client-Software, Backend-System und Datenbank) vorgestellt, um den Zusammenhang der zu entwickelnden Client-Software mit den anderen Komponenten des Systems darzustellen.
Daraufhin werden die verschiedenen Funktionen, die durch das System bereitgestellt werden, beschrieben.
Anschließend wird erörtert, wie die Daten, die im Zusammenhang mit den o.a. Funktionen anfallen, mit der Objektbeschreibungssprache XML strukturiert und zur Weiterverarbeitung aufbereitet werden.

3.1. Die Struktur des Gesamtsystems im Überblick

3.1.1. Die Komponenten und ihre Zusammenhänge

Das Gesamtsystem läßt sich grob in zwei Komponenten unterteilen: Ein Backend-System und eine Client-Applikation (siehe Abb. 3).

Das Backend ist für die zentralisierte, physikalische Verwaltung der Daten zuständig. Hierfür steht eine Datenbank zur Verfügung, in der sowohl Meta-Informationen als auch multimediale Objekte abgelegt werden können. Des weiteren enthält das Backend Schnittstellen zum Internet (bzw. zu einem Intranet), die durch diverse Serverkomponenten bereitgestellt werden. Über diese Schnittstellen können verschiedene Funktionen, die das Backend zur Verwaltung der Datenbestände anbietet, durch externe Programme aufgerufen werden.

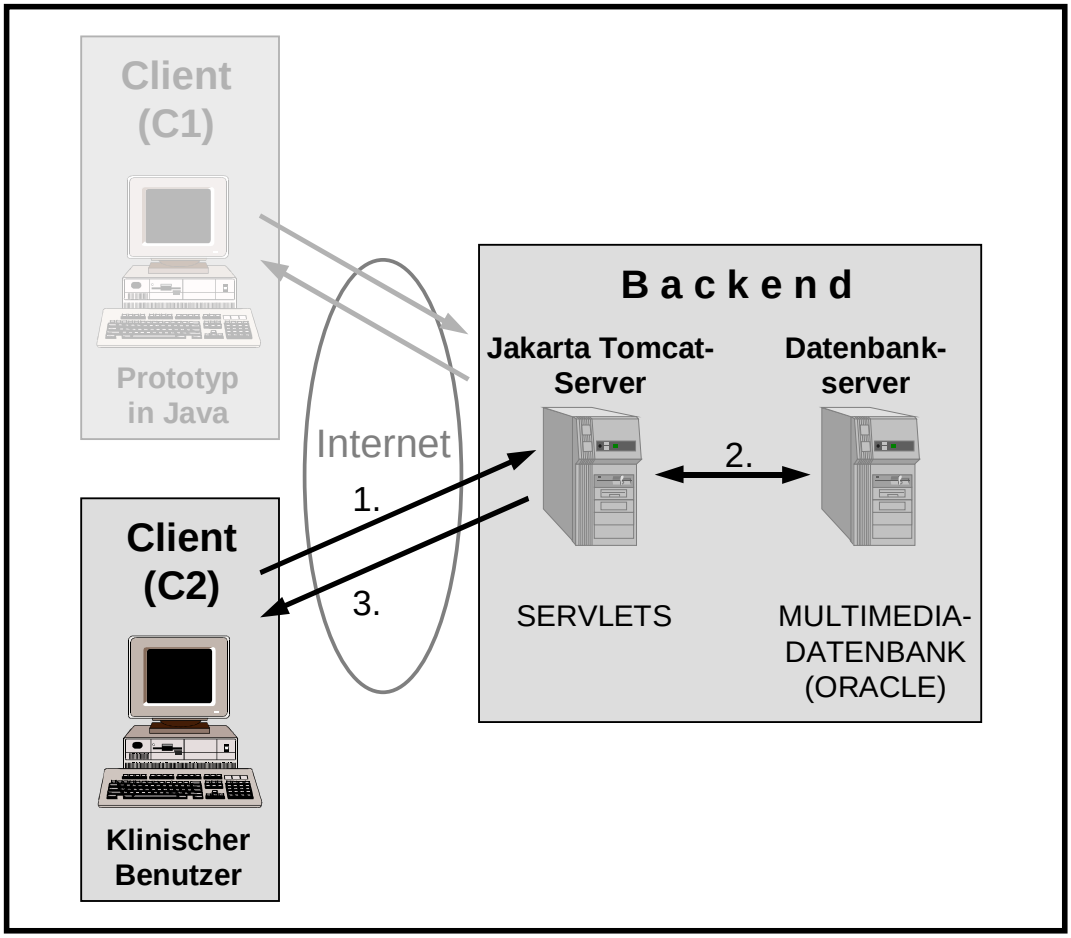


Abb. 3: Schematischer Aufbau des Gesamtsystems
C1: Der im vorherigen Kapitel vorgestellte Test-Client,
der als Java-Applet umgesetzt wurde.
C2: Ein komfortabler, benutzerfreundlicher Client,
dessen Entwicklung Ziel dieser Arbeit ist.
1. Per http-GET bzw. –POST wird eine Funktion der Servlets aufgerufen.
2. Falls nötig führt das Servlet die notwendige Datenbankabfrage durch.
3. Die Ergebnisse werden dem Client in Form von XML übermittelt.

Die Client-Software wird auf dem Rechner des Anwender ausgeführt und bietet eine grafische Benutzeroberfläche, über die verschiedene Funktionen bereitgestellt werden. Dabei kommuniziert die Software über die o.a. Schnittstelle mit dem Backend. Ein erster Client in Form eines Java-Applets wurde bereits in Abschnitt 2.2 vorgestellt. Dieser Prototyp, in Abb. 3 mit C1 bezeichnet, diente allerdings lediglich zu Testzwecken und ist zur Erfüllung der in der Einleitung formulierten Zielsetzungen unzureichend.

Daher soll im Rahmen dieser Arbeit eine alltagstaugliche Benutzeroberfläche (C2 in Abb. 3) für den klinischen Anwender entwickelt werden, mit der diese Ziele in adäquater Form erreicht werden.

3.1.2. Das Backend-System und seine Schnittstellen

Grundlage für das Backend des Systems bildet das in [25,27] vorgestellte System, dessen Serverkomponenten im Rahmen der Integration zusätzlicher Funktionen erweitert wurden. Dieses Backend besteht aus zwei Teilsystemen.

Zum einen gibt es verschiedene Serverkomponenten, die in Form von Java-Servlets umgesetzt wurden. Sie bilden die Internet-Schnittstelle des Backends für den Datenzugriff durch externe Programme, wie z.B. den Benutzerclient. Der Zugriff erfolgt dabei über das im World Wide Web übliche *Hypertext Transfer Protocol* (*http*, vgl. Abb.3, 1-3). Über die http-Befehle *GET* und *POST* können die verschiedenen Servlets angesteuert werden. Dabei werden die Angaben bezüglich der angeforderten Funktionen sowie die zugehörigen Parameter entweder in XML codiert oder als normale Variablen über die http-Befehle *GET* und *POST* übertragen. Das jeweilige Servlet nimmt diese Anfrage entgegen und verarbeitet sie.
Dabei greift es auf eine ORACLE 8.1.7 Datenbank zu, die das zweite der o.a. Teilsysteme des Backends darstellt. In dieser Datenbank werden die Informationen zu den Multimediadateien, sowie die Multimediadateien selbst gespeichert.
Die Ergebnisse der Anfrage werden in XML codiert und an die zugreifende Software (z.B. Client) zurückgeliefert.

3.1.3. Die Client-Software

Neben diesem Backend-System gibt es einen Benutzer-Client, dessen Entwicklung die Hauptaufgabe dieser Arbeit ist.
Ein weit verbreitetes Konzept zur Entwicklung von Software für Windows-Systeme liegt in der Verwendung des *Multiple Document Interface (MDI)*. MDI-

Applikationen bestehen aus einem Hauptfenster, in dem wiederum verschiedene Unterfenster geöffnet werden können. Auf diese Weise können mehrere Funktionen simultan durchgeführt werden.
Durch diese Parallelisierung können auch komplexe Abläufe, wie sie in den im folgenden beschriebenen Funktionen auftreten können, auf benutzerfreundliche Weise realisiert werden. Daher bietet sich die Umsetzung der Client-Software als MDI-Applikation an. In ihr werden die folgenden Funktionen bereitgestellt:

- Verwaltung und Darstellung der einzelnen Objekte und der zugehörigen Meta-Informationen als zusammenhängende Einheiten
- Zuweisung von Freitext-Kommentaren zu den Objekten
- Zuweisung von Attributen/Schlagworten zu den Objekten auf Basis des UMLS Meta-Thesaurus, inkl. Mechanismen zum Abspeichern und Wiederverwerten beliebiger Stichwortkombinationen
- Vergabe differenzierter, personenbezogener Zugriffsrechte
- Hervorhebung und Kommentierung interessanter Ausschnitte bei Bilddateien in Form von Annotationen
- Suche in der Datenbank unter Verwendung des UMLS Meta-Thesaurus
- Verwaltung beliebiger Datenbankobjekte in Galerien
- Mechanismen zur logischen Verknüpfung verschiedener Objekte
- Erstellung web-basierter, interaktiver Lehreinheiten in Form von „virtuellen Rundgängen“

3.2. Detaillierte Beschreibung der Client-Funktionen

3.2.1. Verwaltung von Objekten und Zusatzinformationen als zusammenhängende Einheit

Die wesentliche Aufgabe des Clients für die Multimediadatenbank (kurz MMO-Client) besteht darin, eine komfortable Verwaltung sowie eine übersichtliche Darstellung der gespeicherten Objekte sicherzustellen. Dabei reicht es nicht aus, nur die reinen Multimediadateien abzuspeichern. Vielmehr ist es nötig,

diese Dateien mit Zusatzinformationen, die im Zusammenhang mit den weiteren Funktionen des MMO-Clients generiert werden, zu versehen, um sich (und ggf. anderen) den Zugriff auf die Daten zu erleichtern und um Hintergrundinformationen vermitteln zu können. Eine benutzerfreundliche Verwaltung dieser Informationen ist nur dann möglich, wenn der Anwender die Zusammengehörigkeit der einzelnen Informationssegmente unmittelbar erkennen kann. Aus diesem Grund wurde der Client so konzipiert, daß zusammengehörende Informationen (Multimediadateien und Meta-Daten) stets als gekapselte Einheit dargestellt werden, die im folgenden als *Medical Object (MOB)* bezeichnet werden. Diese MOBs, deren Datenstruktur in Abschnitt 3.3.1 eingehend erläutert wird, können in der Datenbank des Backends oder alternativ im lokalen Dateisystem des Anwenders abgelegt werden. Dabei bietet sich die zweite Alternative an, wenn das jeweilige Objekt offline, d.h. unabhängig vom Netzwerk/Backend, verwendet werden soll. In diesem Fall ist der Umfang der zur Verfügung stehenden Funktionen allerdings eingeschränkt.

In den meisten Fällen wird es angebracht sein, jede einzelne Multimediadatei, wie z.B. ein Bild oder eine Videosequenz, in einem eigenen MOB zu erfassen. Allerdings kann es Situationen geben, in denen mehrere Multimediadateien nur einen Sinn ergeben, wenn Sie im unmittelbaren Zusammenhang betrachtet werden können. Beispiele für einen solchen Fall wären die Präsentationsfolien einer Lehrveranstaltung oder eine Serie von CT-Bildern. Falls diese nur als Abfolge einzelner Bilddateien vorliegen oder aus urheberrechtlichen Gründen nur als solche veröffentlicht werden sollen, wäre es sinnvoll, diese Dateien gebündelt als ein einziges Objekt zu interpretieren. Daher ist es möglich in einem MOB mehrere Multimediadateien zu erfassen. Dabei kann jede dieser Dateien mit einer individuellen Kurzbeschreibung in Form einer Unterschrift versehen werden.

3.2.2. Zuweisung von Freitextkommentaren

Eine Funktion des MMO-Clients ermöglicht dem Anwender, die MOBs mit Kommentaren im Freitext zu versehen. Neben Prosa-Beschreibungen der im

MOB erfaßten Multimeda-Dateien können auch Hinweise auf Urheberrechte, Kontaktinformationen zum Autor und alle weiteren Informationen, die nicht anderweitig im MOB erfaßt werden können, formuliert werden.

3.2.3. Zuweisung von Attributen über das UMLS

Bereits im vorangehenden Kapitel wurden die Vorteile diskutiert, die die Verschlagwortung multimedialer Objekte auf Basis eines definierten Vokabulars - insbesondere des UMLS - in Bezug auf die damit verbundenen Suchmöglichkeiten mit sich bringt. Für die MOBs in der MMO-Datenbank wurde dieses Prinzip erweitert, indem der in [25] vorgestellte Ansatz unter Verwendung des UMLS aufgegriffen wurde.
Hierbei handelt es sich um die Möglichkeit, den Zusammenhang, in dem die Schlagworte zugewiesen wurden, deutlicher herauszustellen. Dies wurde durch eine Funktion zur Integration frei definierbarer Attribute anstelle einfacher Schlagworte in die MOBs realisiert. Dabei besteht der Unterschied zwischen einem Schlagwort und einem Attribut darin, daß ein Attribut einen Namen und eine Ausprägung besitzt, während ein Schlagwort sich - im übertragenen Sinne - nur auf seine Ausprägung beschränkt.
Die Flexibilität, die die individuelle Festlegung der Meta-Information mit sich bringt, wird auch in [22] als notwendige und sinnvolle Funktionalität gedeutet, da bezüglich der Attribute „kein einheitliches Datenformat festlegbar (ist)“. Deshalb wurde in dem dort vorgestellten Programm *ImageCollector* „die Möglichkeit für den Benutzer geschaffen, eigene Bilddatenformate zu definieren“ [22].

Bei der Definition eines Attributs kann der Anwender sowohl den Namen als auch die Ausprägung selbst festlegen. Dabei steht es ihm frei, die jeweiligen Bezeichnungen entweder im Freitext oder unter Verwendung des UMLS-Thesaurus festzulegen.
Im zweiten Fall können die Benutzereingaben für eine Stichwortsuche in den in der UMLS-Datenbank erfaßten Begriffen verwendet werden. Zu jedem gefundenen Begriff wird dann jeder CUI ermittelt, dem der Begriff zugeordnet

ist. Daraufhin werden zu jedem gefundenen CUI sämtliche zugeordneten Synonyme aus der Datenbank gelesen, wodurch die semantische Bedeutung des CUIs abgeleitet werden kann [10]. Man erhält eine umfassende Zusammenfassung aller CUIs, die einen Bezug zu dem gewählten Begriff haben. Dieses Ergebnis wird vom Backend in XML umgewandelt und an den Client übermittelt, der diese Ergebnisse nach CUIs sortiert auf dem Bildschirm darstellt. Der Benutzer kann nun ein geeignetes Synonym zu dem CUI, der ihm am sinnvollsten erscheint, auswählen. Hierdurch wird der Attributsname bzw. –Wert über den ausgewählten CUI und nicht über den Freitext definiert.
Auf diese Weise können einem MOB sukzessiv beliebig viele Attribute zugewiesen werden, die im MOB übersichtlich aufgelistet werden.

Da es sehr wahrscheinlich ist, daß vielen MOBs eines Anwenders die gleichen oder zumindest ähnliche Attribute zugewiesen werden, besteht die Möglichkeit, die Attribute eines MOBs als ein sog. „Attributsprofil“ abzuspeichern. Sämtliche Attributsprofile werden als Liste im XML-Format in einer Systemdatei auf dem Rechner des Anwenders gespeichert. Diese Liste kann dazu verwendet werden, die Attribute eines MOBs mit geeigneten Voreinstellungen zu belegen, so daß nicht für jedes neue MOB sämtliche Einträge von Hand erstellt werden müssen. Hierfür steht dem Benutzer ein entsprechendes Auswahlfeld zur Verfügung, über das das gewünschte Profil aus der Liste selektiert und automatisch in das MOB integriert werden kann.

3.2.4. Integration personenbezogener Zugriffsrechte

Um es dem Benutzer zu ermöglichen, sein geistiges Eigentum nur einem bestimmten Personenkreis zugänglich zu machen, wurden Konzepte zur Verwaltung der Zugriffsrechte eines MOB entwickelt.
Diesbezüglich ist zu definieren

- welche Zugriffsarten grundsätzlich möglich sind und

- auf welche Benutzergruppen diese Zugriffsarten angewendet werden können.

Es muß zwischen drei verschiedenen Zugriffsmodi unterschieden werden:

1. *Kein Zugriff*
   Der Benutzer kann das MOB nicht öffnen. Insbesondere wird das MOB nicht in den Ergebnissen einer Datenbankrecherche aufgeführt, selbst wenn das Suchschema des Anwenders mit den Attributen des MOBs übereinstimmt.

2. *Lesezugriff*
   Der User kann das MOB in der Datenbank finden und öffnen, aber er kann es nicht verändern.

3. *Vollzugriff (Lesen und schreiben)*
   Benutzer, die einen Vollzugriff auf ein MOB besitzen, können dieses sowohl ansehen als auch verändern. Das bedeutet, daß der entsprechende Benutzer das MOB behandeln kann, als sei es sein eigenes, mit einer Ausnahme: Er kann zwar auch die Zugriffsrechte verändern, allerdings kann er nicht dem Eigentümer des MOBs den Zugriff verweigern. Der Benutzer, der das MOB angelegt hat, wird immer Eigentümer und somit schreibberechtigt sein.

Aus Sicht der Benutzergruppen werden vier Unterscheidungen vorgenommen:

1. *Eigentümer des MOBs*
   Der Benutzer, der das MOB erstellt hat, ist der Eigentümer und stellt eigentlich nur im weiteren Sinne eine Benutzergruppe dar. Er hat volle Schreibrechte und kann insbesondere die Zugriffsrechte für die weiteren Benutzergruppen festlegen.

2. *Mitglieder der Autorenliste*

In einem gesonderten Bereich des MOBs können beliebig viele User, die eine Zugangskennung zum System haben, in einer Autorenliste erfaßt werden. Hierfür stellt das System eine entsprechende Verwaltungsfunktion bereit. Alle Benutzer, die hier aufgeführt werden, bilden eine Gruppe, für die die Zugriffsrechte autark verwaltet werden können.

3. *Institutsangehörige*
   Es ist möglich, im Backend alle Anwender, die der gleichen universitären Einrichtung angehören, in jeweils einer Benutzergruppe zu erfassen. Für diese Gruppe können die Zugriffsrechte separat im MOB verwaltet werden.

4. *Alle anderen*
   Alle Nutzer, die nicht zu einer der o.a. Gruppen gehören, werden in der Gruppe *„Alle Anderen“* zusammengefaßt. Darin sind insbesondere alle die Anwender enthalten, die keine explizite Zugangskennung für das System besitzen.

Über entsprechende Auswahlfelder kann der Anwender, sofern er Schreibrechte für das jeweilige MOB besitzt, für jede dieser Benutzergruppen (außer der ersten, da hier grundsätzlich Schreibrechte vorliegen) die geeigneten Zugriffsrechte parametrieren.

3.2.5. Annotationen bei Bilddateien

Bilddateien spielen im Zusammenhang mit Multimedia in der Medizin eine besonders wichtige Rolle, so daß ihnen besondere Aufmerksamkeit in Hinblick auf die zu implementierenden Funktionen geschenkt werden sollte [22]. Eine wichtige Anforderung in der Medizin ist es, sog. „Regions of Interest“ (ROI) in Bildern festlegen und optisch hervorheben zu können [18,22].
Aus diesem Grund muß der MMO-Client die Möglichkeit bieten, Bilder mit Annotationen zu versehen. Die interessanten Ausschnitte können dazu über eine integrierte Zeichenfunktion mit Rechtecken und Ellipsen hervorgehoben und mit einem beschreibenden Text versehen werden. Dabei werden die

Hervorhebungen als separate Metadaten im MOB gespeichert und als Überlagerung des Bildes dargestellt, so daß die Multimediadatei als solche physikalisch unberührt bleibt. Hierdurch ist es insbesondere möglich, auch nach dem Abspeichern in der Datenbank Veränderungen (z.B. Verschieben, Löschen, ...) an den Annotationen vorzunehmen.

3.2.6. Suchen in der Datenbank

Über eine Funktion zur Suche in der Datenbank kann ein Anwender interessante MOBs im MMO-System ausfindig machen. Hierbei spielen die in Abschnitt 3.2.3 erläuterten frei definierbaren Attribute eine wichtige Rolle, da sie als Vergleichskriterien der gespeicherten MOBs mit den Präferenzen des Benutzers dienen und somit entscheidend für den Erfolg einer Suche sind.
Eine Datenbankabfrage wird in drei Schritten vollzogen. Jeder dieser Schritte führt zu einer Reaktion des System, die jeweils den nächsten Schritt einleitet:

1. *Benutzer:* Formulierung der Suchanfrage
   Reaktion des Systems: Darstellung der Attribute, hinter denen sich relevante MOBs verbergen
2. *Benutzer:* Auswahl eines Attributs
   Reaktion des Systems: Auflistung der zugehörigen relevanten MOBs
3. *Benutzer:* Auswahl der gewünschten MOBs
   Reaktion des Systems: MOBs werden geöffnet

3.2.6.1. Formulierung der Suchanfrage

Um eine Suche in der MMO-Datenbank durchzuführen, muß der Anwender zunächst angeben, nach welchen Begriffen gesucht werden soll. Diese können in ähnlicher Form wie die Attribute der MOBs angegeben werden, wobei dem Anwender die gleichen Eingabemechanismen wie in Abschnitt 3.2.3 zur Verfügung stehen. Insbesondere können sowohl die Namen als auch die Ausprägungen der Attribute, nach denen gesucht werden soll, frei gewählt

werden. Dabei können Name und Ausprägung als Freitext oder als UMLS-Konzept angegeben werden.
Darüber hinaus wird über eine Option, die den Wert „kann“ oder „muß“ annimmt, für jedes Suchattribut angegeben, ob es zwingend in den Suchergebnissen enthalten sein muß oder ob es optional vorkommen kann. Mit Hilfe der boolschen Operatoren „und“ und „oder“, werden die Suchbegriffe vom Backend zu einer logisch verknüpften Suchanfrage zusammengefaßt werden.
Konkret sind die Suchoptionen „kann“ und „muß“ der einzelnen Suchattribute folgendermaßen zu interpretieren. Es seien $A_1, \ldots, A_m$ die „muß“-Attribute und $A_{m+1}, \ldots, A_n$ die „kann“-Attribute einer Suchanfrage (mit $m<n$). Der resultierende boolsche Ausdruck für die Datenbankabfrage lautet dann:

$[A_1$ und A_2 und ... und $A_m]$
und
$[A_{m+1}$ oder A_{m+2} oder ... oder $A_n]$

Insbesondere liegt eine reine „und“-Verknüpfung (bzw. „oder“-Verknüpfung) vor, wenn ausschließlich „muß“-Attribute (bzw. „kann“-Attribute) verwendet werden.

Zusätzlich wird die gesamte Suchanfrage über ein entsprechendes Eingabefeld mit einem Unschärfegrad versehen, der sich auf die UMLS-basierten Suchattribute bezieht. Hierüber kann festgelegt werden, ob nach den exakten Konzepten gesucht werden soll, oder ob über die semantischen Netze des UMLS auch verwandte Konzepte in die Suche mit einbezogen werden sollen (vgl. Beispiel in 2.1.2). Der Unschärfegrad kann die Werte eins bis vier annehmen, wobei eins einer exakten Suche entspricht. Je größer dieser Wert gewählt wird, desto mehr verwandte Konzepte werden in die Suche einbezogen. Hierdurch kann die Zahl der gefundenen MOBs erhöht werden. Gleichzeitig sinkt allerdings die Genauigkeit der Suche.
Die konkrete Implementierung einer unscharfen Suche wird im Backend-System realisiert. Jedes der Suchattribute wird dabei durch eine Menge verwandter Suchbegriffe ersetzt, die um so umfangreicher ist, je größer der Unschärfegrad ist. Diese Menge könnte beispielsweise mit Hilfe der in Abschnitt

2.1.2 aufgeführten Verknüpfungen des einfachen semantischen Netzwerks des UMLS Meta-Thesaurus konstruiert werden. Abhängig vom betrachteten Suchattribut *A* und vom Unschärfegrad *k*, wäre beispielsweise die folgende Definition für diese Attributsmengen *M(A, k)* denkbar:

M(A,1) := { *A* }

M(A,2) := *M(A,1)* ∪ „Alle Attribute, die in Relation *RN* oder *RQ* zu *A* stehen“

M(A,3) := *M(A,2)* ∪ „Alle Attribute, die in Relation *RB*, *PAR* oder *CHD* zu *A* stehen“

M(A,4) := *M(A,3)* ∪ „Alle Attribute, die in Relation *SIB* oder *RO* zu *A* stehen“

Der o.a. boolsche Ausdruck für die Datenbankabfrage muß dann folgendermaßen angepaßt werden (sei $M(A_j, k) = \{ A_{j,1}, \ldots , A_{j,pj} \}$ für $j=1, \ldots , n$ mit $k \in \{ 1, .. , 4 \}$ fest vorgegeben):

```
[
     (A1,1 oder A1,2 oder ... oder A1,p1)
     und
     (A2,1 oder A2,2 oder ... oder A2,p2)
     und
     ...
     und
     (Am,1 oder Am,2 oder ... oder Am,pm)
]
und
[
     Am+1,1 oder Am+1,2 oder ... oder Am+1,p(m+1)
     oder
     Am+2,1 oder Am+2,2 oder ... oder Am+2,p(m+2)
     oder
     ...
     oder
     An,1 oder Am,2 oder ... oder An,pn
]
```

Man sieht leicht, daß die erste Version des boolschen Ausdrucks ein Spezialfall der zweiten Version ist, der im Falle einer exakten Suche (*k=1*) auftritt.

3.2.6.2. Verarbeitung der Suchanfrage und Darstellung der relevanten Attribute

Nachdem der Anwender die Suchanfrage formuliert hat, kann er diese an das Backend-System schicken, welches die Ergebnisse aus der Datenbank ausliest und zunächst zwischenspeichert. Dabei werden alle gefundenen MOBs mit einer Punktzahl (Score) versehen, mit der die Relevanz des jeweils gefundenen Objekts bezüglich der gestellten Suchanfrage gemessen werden kann. Bei der Berechnung dieses Scores müssen mehrere Faktoren berücksichtigt werden.
Zum einen spielt die Anzahl der Attribute, die sowohl im MOB als auch in der Suchanfrage (inklusive Unschärfe) enthaltenen sind, eine wichtige Rolle. Denn je mehr Übereinstimmungen gefunden werden, desto wahrscheinlicher ist es, daß das entsprechende Objekt für den Benutzer von Interesse ist.
Des weiteren wird berücksichtigt, wie exakt diese Übereinstimmungen sind. Dabei ist der Score-Beitrag für jede Übereinstimmung um so höher, je weniger Unschärfe notwendig ist, damit die Übereinstimmung zustande kommt. Oder mit anderen Worten: Es seien *A* das exakte Attribut aus der Suchanfrage, *k* der Unschärfegrad der Suche und *Q* das Attribut des MOBs, das (ggf. unter Unschärfe) mit *A* übereinstimmt. Dann ist der Beitrag, den die Übereinstimmung von *A* und *Q* für den Score des jeweiligen MOBs leistet, um so größer, je kleiner der Wert von

$$\min \{ j \in \mathbb{N} ; Q \in M(A, j), 1 \leq j \leq k \}$$

ist.

Nachdem alle relevanten MOBs aus der Datenbank ausgelesen und mit einer entsprechenden Punktzahl versehen wurden, erstellt das Backend zunächst eine Übersicht über alle Attribute der MOBs, die einen Beitrag zum Score eines gefundenen MOBs geleistet haben. Diese Übersicht wird dem MMO-Client als Reaktion auf die Suchanfrage zurückgeliefert. Der Client stellt diese Informationen in Form einer Baumstruktur am Bildschirm des Benutzers dar. Dabei werden alle vorkommenden Attributsnamen in der ersten Ebene des Baums aufgeführt. Unterhalb der einzelnen Namen werden dann alle

zugehörigen Attributswerte aufgeführt. Die Reihenfolge dieser Attribute ergibt sich aus der Summe der Scores der jeweils zugehörigen gefundenen MOBs. Auf diese Weise werden die wichtigsten Attribute stets als erstes angezeigt.

3.2.6.3. Auflistung der MOBs zu einem ausgewählten Attribut

Mit Hilfe der o.a. Baumdarstellung kann der Anwender auf komfortable Weise in den relevanten Attributen der gefundenen MOBs navigieren. Durch Auswahl eines geeigneten Attributs werden alle MOBs, die sich hinter diesem Attribut verbergen, in Form einer Liste am Bildschirm dargestellt. Dabei werden die Identifikationsnummer, unter der das MOB in der Datenbank gespeichert wurde, der Titel des MOBs und ein Vermerk, ob auf das MOB nur lesend oder auch schreiben zugegriffen werden darf, angegeben. Sofern es sich um eine Bilddatei handelt, wird darüber hinaus eine Miniaturansicht dieses Bildes angezeigt.

Ausgehend von dieser Liste kann der Anwender den letzten Schritt zu einer erfolgreichen Datenbanksuche durchführen, indem er beliebige MOBs aus der Liste auswählt und sie durch Aufruf einer entsprechenden Funktion öffnet.

3.2.7. Speichern von MOBs in Galerien

Um auch später noch auf interessante Suchergebnisse zurückgreifen zu können, ohne die Datenbank jedesmal von Neuem absuchen zu müssen, muß eine geeignete Speicherfunktion bereitgestellt werden. Zu diesem Zweck bietet es sich an, die Ergebnisse in sog. Galerien abzulegen. Wie der Name vermuten läßt, handelt es sich bei einer Galerie um eine Kollektion ausgewählter Objekte, die vom Benutzer verwaltet werden kann. Dabei hat der Anwender insbesondere die Möglichkeit, MOBs hinzuzufügen und zu entfernen und ihre Reihenfolge zu verändern. Jede Galerie kann auf der Festplatte des Benutzers gespeichert und später jederzeit über die Client-Software wieder geöffnet werden. Des weiteren ist es möglich, eine Galeriedatei, genau wie eine Multimediadatei, in einem MOB zu erfassen und sie von dort aus wieder zu

öffnen. Auf diese Weise können Galerien in der Datenbank gespeichert und über die Zugriffsrechte beliebigen Anwendern zur Verfügung gestellt werden.

Aus datentechnischer Sicht beinhalten Galerien Informationen über den Titel der jeweiligen Kollektion sowie eine lineare Liste, deren Einträge jeweils einem MOB entsprechen. Um Redundanzen zu vermeiden, werden dabei nicht die gesamten MOBs in dieser Liste gespeichert, sondern lediglich deren Identifikationsnummern aus der Datenbank. Dies bedeutet insbesondere, daß nur Objekte aufgenommen werden können, die bereits in der Datenbank erfaßt sind. MOBs, die neu erzeugt oder von der lokalen Festplatte geladen wurden, können nicht aufgenommen werden. Des weiteren können Galerien nicht „offline“ benutzt werden, da sämtliche Informationen, die zur Darstellung benötigt werden, aus der Datenbank geladen werden müssen.

3.2.8. Verknüpfung von MOBs

Die Funktionen zum Suchen nach relevanten, im System erfaßten MOBs bieten aufgrund der Integration des UMLS äußerst flexible Mechanismen zur Recherche in der Datenbank. Allerdings wäre es auch hilfreich für den Anwender, sich die Erfahrungen und Empfehlungen anderer Benutzer des Systems zunutze machen zu können. Einen Ansatz, um einen derartigen Erfahrungsaustausch zu ermöglichen, bieten die Funktionen zum Verknüpfen von MOBs.

Diese ermöglichen es dem Benutzer, Referenzen auf andere im System erfaßte Objekte in das eigene MOB zu integrieren. Dabei wird der logische Zusammenhang zwischen den Objekten durch die Bezeichnung der gewählten Relation repräsentiert (z.B. „ist Vergrößerung von“, „hat ähnliche Bildinformationen wie“, ...). Die zur Verfügung stehenden Relationen werden im Backend-System verwaltet und dem Client als Liste in XML-basierter Form übermittelt, so daß der Anwender die verwandtschaftlichen Beziehung zwischen den MOBs aus dieser Liste auswählen kann.

Auf diese Weise ist es dem Benutzer möglich, anderen Anwendern, die sein MOB in der Datenbank gefunden haben, über diese Referenzen Hinweise auf weitere thematisch verwandte Datenbankobjekte zu geben.

3.2.8.1. Voraussetzungen zum Erstellen von Verknüpfungen

Um verschiedene MOBs miteinander zu verknüpfen werden mindestens drei Komponenten benötigt:

1. Genau ein MOB, von dem aus auf ein oder mehrere MOBs verwiesen werden soll („Quell-MOB“)
2. Mindestens ein MOB, auf welches verwiesen werden soll („Ziel-MOB(s)“)
3. Informationen bezüglich der Art der verwandtschaftlichen Beziehungen, repräsentiert durch die Bezeichnung mindestens einer Relation

Um etwaige Verknüpfungen auf sinnvolle Weise definieren zu können, ist es nötig, daß sowohl Quell- als auch Ziel-MOB gewisse Bedingungen erfüllen.
Da die Verknüpfungsinformationen in das MOB integriert werden müssen, ist es notwendig, daß der Benutzer, der die Verknüpfung anlegen möchte, Schreibrechte für das Quell-MOB besitzt. Dies kann auf verschiedene Weise erreicht werden:

- Das MOB wird direkt aus der Datenbank geöffnet und die Zugriffsrechte des MOBs wurden entsprechend eingestellt.
- Das MOB wurde aus dem lokalen Dateisystem geöffnet.
- Das MOB wurde neu erstellt und noch nicht in der Datenbank gespeichert.

In den beiden letzten Fällen steht das MOB in keinem unmittelbaren Zusammenhang zur Datenbank, weshalb auf derartige MOBs stets schreibend zugegriffen werden darf.

Auch für die Ziel-MOBs müssen einige Bedingungen erfüllt sein, damit von einem anderen MOB aus auf sie verwiesen werden kann.
Ähnlich wie bei den Galerien werden die referenzierten MOBs nicht vollständig in das Quell-MOB integriert. Vielmehr werden lediglich die Identifikationsnummern der Datenbank gespeichert. Aus diesem Grund können nur diejenigen Objekte Ziel-MOBs sein, die bereits in der Datenbank gespeichert wurden und somit eine derartige Identifikationsnummer besitzen.
Da der Anwender, der die Relation erstellen möchte, die Ziel-MOBs zunächst inhaltlich sichten muß, ist es notwendig, daß er mindestens lesend auf die Ziel-MOBs zugreifen kann.

3.2.8.2. Kontext-bezogene Verknüpfungen

Um den Zusammenhang, in dem eine Verknüpfung zu interpretieren ist, genauer herausstellen zu können, kann jede Verknüpfung in einem bestimmten Kontext vergeben werden. Dies ist möglich, indem den drei Komponenten einer Verknüpfung (Quell-MOB, Ziel-MOBs, Relationen) eine weitere, optionale Komponente hinzugefügt wird. Hierbei handelt es sich um den Kontext, über den der Zusammenhang, in dem die jeweilige Verknüpfung gelten soll, eingeschränkt werden kann. Dabei können alle zur Verfügung stehenden Kontexte in einer Textdatei, dessen Position über die Systemeinstellungen des Programms festgelegt werden kann, vom Benutzer selbst verwaltet werden. Auf diese Weise wird für den Anwender eine größtmögliche Flexibilität in Bezug auf die Wahl der Kontexte sichergestellt.
Dieses Konzept soll anhand eines kleinen Beispiels erläutert werden.

> *Es sei A ein MOB mit einer Abbildung einer HIV-infizierten T-Helfer-Zelle. Des weiteren seien B und C MOBs, in denen jeweils ein Word-Dokument enthalten ist. In B gibt es einen Text zur Wirkweise von Retro-Viren, in C liegt ein Dokument zur Beschreibung von CD4+-Lymphozyten vor. Sowohl B als auch C weisen einen Zusammenhang zu A auf, weshalb A mit B und C unter der Relation „wird textuell beschrieben durch“ verknüpft wird. Da in diesem Fall nicht unmittelbar*

ersichtlich ist, was jeweils textuell beschrieben wird, könnten die Verknüpfungen folgendermaßen kontext-bezogen vergeben werden:

- *A → B, Relation: „wird textuell beschrieben durch“, Kontext: „Infos zu Retro-Viren“*
- *A → C, Relation: „wird textuell beschrieben durch“, Kontext: „Infos zur Zelle“*

3.2.8.3. Annotationen im Zusammenhang mit Verknüpfungen

Annotationen zu Bilddateien spielen auch in Verbindung mit Verknüpfungen eine wichtige Rolle. Es wurden Funktionen in den MMO-Client integriert, durch die auch etwaige Annotationen eines Quell-MOBs mit Verweisen zu anderen MOBs versehen werden können. Das folgende Beispiel soll diesen Zusammenhang veranschaulichen.

Es sei A ein MOB mit einer Bilddatei, in der mehrere Blutzellen dargestellt werden. Dabei scheint eine dieser Zellen medizinisch auffällig zu sein. Des weiteren sei B ein MOB mit einem Bild, welches diese Zelle in vergrößerter Form abbildet. In diesen Fall wäre es sinnvoll, die auffällige Zelle in A mit einer elliptischen Hervorhebung zu versehen, und dann eine Verknüpfung von dieser Hervorhebung auf MOB B unter der Beziehung „ist Verkleinerung von“ zu erstellen.

Da jede Annotation auch Bestandteil des MOBs ist, wird jede Relation, die sich auf eine Annotation bezieht, gleichzeitig auch global für das gesamte MOB angelegt. Das bedeutet, daß die Vereinigung aller auf Annotationen bezogenen Verweise eine Teilmenge der Verweise des gesamten MOBs darstellen.
Dies zieht folgende Konsequenzen nach sich:

- Wird eine Relation aus einer Annotation gelöscht, so bleibt sie trotzdem als Relation des gesamten MOBs erhalten. Dies ist darauf zurückzuführen, daß

sich nach Erstellung der Verknüpfung nicht mehr nachvollziehen läßt, ob die Verknüpfung nicht auch unabhängig von der Annotation als Verknüpfung des gesamten MOBs angelegt wurde.

- Wird eine auf das gesamte MOB bezogene Relation entfernt, so wird sie automatisch auch aus allen Annotationen, in denen sie vorhanden ist, entfernt.

3.2.9. Interaktive Lehreinheiten in Form von „virtuellen Touren“

Als Plattform für die Bereitstellung interaktiver Lehreinheiten bietet sich die Verwendung des World Wide Webs an [22,41]. Aus diesem Grund können ausgewählte MOBs mit dem MMO-Client in das HTML-Format exportiert und so im Internet veröffentlicht werden.

Um diese HTML-Seiten mit einer geeigneten Navigationsstruktur versehen zu können, ist es notwendig, diese Struktur in den MOBs, die zu jeweils einer Lehreinheit gehören, abzubilden. Hierfür wurde der MMO-Client um eine Funktion erweitert, mit der beliebige MOBs zu sogenannten „virtuellen Touren“ bzw. „virtuellen Rundgängen“ zusammengefaßt werden können.

Diese Touren eignen sich besonders gut zur Konstruktion virtueller Führungen durch Gebäude, die im Internet vorgestellt werden sollen. Aber auch medizinische Lehrmaterialien lassen sich unmittelbar durch dieses Konzept strukturieren und darstellen.

Im folgenden wird zunächst definiert, welche Arten von virtuellen Rundgängen zu unterscheiden sind. Anschließend wird erörtert, wie diese Konstrukte durch die Funktionalitäten des MMO-Clients umgesetzt werden können. Dabei wird insbesondere eine Erweiterung der in Abschnitt 3.2.8 vorgestellten Verknüpfungsfunktion erläutert, die für die Konstruktion von Touren notwendig ist.

3.2.9.1. Theoretische Definition von virtuellen Rundgängen

Ein virtueller Rundgang ist ein Zusammenschluß verschiedener Stationen, die jeweils eine Informationseinheit bilden. Dabei wird jede Station durch eine HTML-Seite repräsentiert, auf der die Informationen in Form eines Fotos bzw. eines Films und eines beschreibenden Textes dargestellt werden. Außerdem wird auf der Seite ein Navigationsbereich zur Erschließung der weiteren zur Tour gehörenden Stationen bereitgestellt.
Grundsätzlich lassen sich zwei Arten von virtuellen Rundgängen unterscheiden: Geführte und ungeführte Touren.

Geführte Touren bestehen aus einer Liste verschiedener Stationen, die inhaltlich nach einer vorgegebenen Reihenfolge sortiert sind. Diese Reihenfolge wird im Navigationsbereich jeder Seite in Form einer Auflistung der Stationen abgebildet.
Innerhalb der geführten Touren kann man zwischen zwei Klassen unterscheiden: Streng geführte Touren und freie geführte Touren.
Streng geführte Touren zeichnen sich dadurch aus, daß die o.a. inhaltliche Reihenfolge der Stationen bei der Erschließung der Inhalte eingehalten werden muß. Dies wird dadurch realisiert, daß die Stationsliste im Navigationsbereich nur aus einfachem Text besteht. Die Navigation ergibt sich aus Hyperlinks unterhalb dieser Liste mit Bezeichnungen der Form „zur nächsten Station“ bzw. „eine Station zurück“.
Im Gegensatz dazu erlauben **freie geführte Touren** eine freiere Navigation innerhalb der Stationen des Rundgangs. Dabei kann die vorgesehene Reihenfolge noch immer anhand der Liste im Navigationsbereich nachvollzogen werden, jedoch ist es dem Benutzer nun möglich, diese Reihenfolge zu durchbrechen und die Stationen nach eigenem Ermessen aufzurufen. Um dies zu ermöglichen, bestehen die Einträge in der Navigationsliste nicht wie bei den streng geführten Touren aus einfachem Text, sondern aus Hyperlinks, die unmittelbar auf die jeweiligen Stationen verweisen. Die o.a. Links in der Form „zur nächsten Station“ und „eine Station zurück“ können daher entfallen.

Im Gegensatz zu den geführten Rundgängen werden die Stationen eines **ungeführten Rundgangs** nicht in Listenform erfaßt. Vielmehr ist möglich, von einer Station aus auf beliebig viele weitere Stationen zu verweisen, wodurch ein deutlich höheres Maß an Flexibilität bzgl. der Gestaltung der Navigationsstrukturen bereitgestellt wird. Um diese mitunter große Zahl von Verweisen übersichtlich gestalten zu können, ist es möglich, diese in verschiedenen Kategorien thematisch zu unterteilen.
Im Zusammenhang mit Gebäuden, die im Internet vorgestellt werden sollen, eignen sich ungeführte Touren besonders gut dafür, dem Besucher eine Simulation der räumlichen Gebäudestruktur zur Verfügung zu stellen. Dabei kann der Navigationsbereich in Kategorien wie „geradeaus“, „nach links“, „zurück“ etc. unterteilt werden. Innerhalb der Kategorien werden dann Hyperlinks zu den jeweils benachbarten Stationen aufgeführt. Auf diese Weise kann sich der Besucher die räumliche Struktur des Gebäudes erschließen und die entsprechenden Informationen zu den einzelnen Bereichen abrufen.

3.2.9.2. Knotentypen zur Verwaltung verschiedener Inhalte

Für die Konstruktion virtueller Rundgänge werden verschiedene Knoten-Typen unterschieden. Dabei beschreibt jeder Knotentyp eine eigene Kategorie von Inhalten. Jede Instanz einer dieser Knotentypen wird durch ein eigenes MOB repräsentiert, in dem eine einzelne Informationseinheit gekapselt wird. Aus der geschickten Verknüpfung dieser MOBs ergeben sich die logischen Strukturen der verschiedenen Rundgänge.
Konkret werden folgende Knoten-Typen unterschieden:

1. *Positionsknoten*

   Positionsknoten bestehen aus einem normalen jpg-Bild, einem Panorama-jpg-Bild, das über das Java-Applet *„PTViewer“* [17] als interaktive Panoramaansicht dargestellt werden kann, oder einem Flash-Film. Diese Knoten repräsentieren die verschiedenen Stationen innerhalb der Rundgänge.

2. *Textknoten*

   Hierbei handelt es sich um die Textdateien mit den Beschreibungen zu den Stationen. Es können für jede Station mehrere solcher Textknoten angelegt werden, die den Stationen in Abhängigkeit von den jeweiligen Rundgängen zugeordnet werden können.

3. *Lageplanknoten*

   In einem Lageplanknoten kann eine jpg-Datei erfaßt werden, in der eine Gesamtübersicht des vollständigen Rundgangs abgebildet wird. Dieses Bild kann mit Annotationen versehen werden, denen Positionsknoten zugeordnet werden können. In den späteren HTML-Dateien kann sich der Besucher anhand dieses Lageplans eine Übersicht verschaffen und ggf. direkt zu der gewünschten Position springen.

   Im Zusammenhang mit der Darstellung von Gebäuden bietet es sich an, in einem Lageplanknoten den Grundriß zu erfassen. Sofern das Gebäude aus mehreren Stockwerken besteht, ist es möglich, in einem Lageplanknoten mehrere jpg-Dateien zu erfassen, die jeweils ein Stockwerk repräsentieren.

Um einen oder mehrere virtuelle Rundgänge erstellen zu können, ist es notwendig, zunächst die umzusetzenden Inhalte in verschiedenen Positions-, Text- und ggf. Lageplan-MOBs zu erfassen und in der Datenbank zu speichern. Die eigentliche Struktur der Rundgänge ergibt sich dann durch eine geeignete Verknüpfung dieser MOBs.

Dabei wird einem Positions-MOB ein Text zugeordnet, indem er mittels der Relation „hat eingebettete Ressource“ mit dem entsprechenden Text-MOB verknüpft wird.

Die Auswahl und die Reihenfolge der Positionsknoten einer geführten Tour wird festgelegt, indem die entsprechenden Positions-MOBs der Reihe nach über die Relation „zur nächsten Station“ miteinander verknüpft werden. Anhand dieser Verkettung kann der Aufbau der Navigationsstruktur der Stationen unmittelbar ermittelt werden.

Im Gegensatz dazu kann für die Erstellung der Navigation eines ungeführten Rundgangs jede beliebige im System bereitgestellte Relation verwendet werden. Einzige Ausnahme bildet die für die Textzuordnung reservierte Relation „hat eingebettete Ressource“. Bei der Erzeugung der HTML-Seiten bildet jede verwendete Relation eine Kategorie innerhalb des Navigationsbereichs. In dieser Kategorie werden automatisch Hyperlinks zu allen Stationen aufgeführt, die bezüglich der jeweiligen Relation mit dem aktuellen Positionsknoten verknüpft wurden.

Eine flexible Gestaltung mehrerer virtueller Rundgänge ist nur möglich, wenn jeder Positionsknoten in beliebig vielen Rundgängen verwendet werden kann. Dabei ist es notwendig, einem Positions-MOB in jedem Rundgang einen jeweils anderen Textknoten zuordnen zu können. Des weiteren werden sich die jeweils benachbarten Positionsknoten sowie die Relationen, unter denen sie mit dem aktuellen Positions-MOB verknüpft wurden, von Rundgang zu Rundgang unterscheiden.
Aus diesem Grund müssen die o.a. Verknüpfungen stets einem bestimmten Kontext zugeordnet werden (siehe Abschnitt 3.2.8.2). Dabei sollte der jeweilige Kontext dem Titel des zu erstellenden Rundgangs entsprechen. Auf diese Weise kann eindeutig bestimmt werden, welche MOBs in welcher Funktion zu welchem Rundgang gehören.

3.3. Datenmodellierung durch XML

Die Verarbeitung der verschiedenen Konstrukte (z.B. MOBs, Galerien, ...) sowie die Ausführung der verschiedenen Funktionen des MMO-Clients setzen eine strukturierte Aufbereitung der damit verbundenen Datenbestände voraus. Auf Basis dieser Strukturen werden die Daten einerseits im MMO-Client verarbeitet, und andererseits zwischen dem MMO-Client und dem Backend-System ausgetauscht.
Da letzteres eine plattformunabhängige Beschreibung der Daten erfordert, erfolgt die Strukturierung der Daten mit Hilfe der textbasierten Objekt-

Beschreibungssprache XML. Über verschiedene Dokumententyp Definitionen (DTDs), die im Anhang eingesehen werden können, werden die zugrundeliegenden Datenmodelle formal beschrieben.
Im folgenden werden diese Datenmodelle anhand ihrer XML-Repräsentationen vorgestellt. Dabei handelt es sich konkret um:

- Die Modellierung von MOBs
- Die Modellierung von Galerien
- Die Modellierung der Daten, die im Zusammenhang mit einer Suche in der MMO-Datenbank anfallen

3.3.1. Modellierung von MOBs

Für die strukturierte Erfassung der Informationen zu einem MOB wird eine Unterteilung des zugehörigen XML-Codes in die folgenden Bereiche vorgenommen:

- Identifikationsangaben zum Autor / bearbeitenden Benutzer
- Titel des MOBs
- Informationen zu den Zugriffsrechten, inkl. Autorenliste (vgl. 3.2.4)
- Verzeichnis der Attribute (vgl. 3.2.3)
- Freitextkommentar (vgl. 3.2.2)
- Angaben zu verwandten MOBs (vgl. 3.2.8)
- Abbildung der Multimediadateien sowie etwaiger Annotationen (vgl. 3.2.5)

3.3.1.1. Grundgerüst des XML-Codes

Die XML-Dokumente, die zur Abbildung von MOBs verwendet werden, werden von einem Wurzel-Element namens *medobject* eingeschlossen. Diesem Element werden ggf. die XML-Attribute *id* und *creationdate* hinzugefügt. Dabei enthält *creationdate* die genaue Zeit, zu der das MOB erstmalig in der Datenbank gespeichert wurde.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">
    <!-- Inhalt des MOBs -->
</medobject>
```

XML-Beispiel 1: Grundgerüst zur Abbildung von MOBs

Das Attribut *id* repräsentiert die Identifikationsnummer des MOBs in der Datenbank und wird nur dann mit Inhalt versehen, wenn das MOB zuvor aus der Datenbank geladen wurde. MOBs, die neu angelegt werden, erhalten beim Speichern in der Datenbank ein leeres *id*-Attribut, um dem Backend-System zu signalisieren, daß es sich um ein neues Objekt handelt. Gleiches gilt für MOBs, die von der Festplatte geöffnet wurden, und zwar auch dann, wenn sie ursprünglich aus der Datenbank geladen und auf der Festplatte gespeichert wurden. Dies ist darauf zurückzuführen, daß eine große Zeitspanne zwischen dem Abspeichern auf der Festplatte und dem erneuten Hochladen in die Datenbank nicht ausgeschlossen werden kann. In dieser Zeit können Änderungen durch weitere schreibberechtigte Benutzer an der in der Datenbank gespeicherten Version des MOBs vorgenommen werden. Da diese Änderungen durch ein erneutes Hochladen der älteren Festplattenversion überschrieben würden, können MOBs, die von der Festplatte geladen und dann in der MMO-Datenbank gespeichert werden, nur als neue MOBs in der Datenbank erfaßt werden.

3.3.1.2. Benutzeridentifikation

Wie bereits erwähnt ist das Abspeichern von MOBs in der Datenbank an differenzierte Zugriffsrechte gebunden. In diesem Zusammenhang ist es notwendig, die Identität des Benutzers, der das MOB hochläd, anhand seiner Kennung und eines „Session keys“ zu überprüfen. Der Session key ist eine zufällige Zeichenfolge mit zeitlich begrenzter Gültigkeit, die dem Benutzer automatisch nach erfolgreicher Systemanmeldung zugeordnet wird. Aus Sicherheitsgründen wird dieser Schlüssel bei jeder Transaktion anstelle des Benutzerpaßwort übermittelt.

Im XML-Quelltext werden diese Angaben durch das *editor*-Element integriert, welches folgende Struktur hat:

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">
  <editor login="ruppelm" sessionkey="992868600761" />
  <!-- ... Weitere Inhalte ... -->
</medobject>
```

XML-Beispiel 2: Benutzerinformationen im editor-Tag

Dieses Element hat keine Kinderelemente, die Benutzerkennung und der Session key werden über die Attribute *login* und *sessionkey* eingebunden.

Beim Herunterladen eines MOBs dient das *editor*-Element dazu, Informationen bzgl. des Eigentümers des MOBs in Form seiner Benutzerkennung an den Client zu übertragen. In diesem Fall entfällt das *sessionkey*-Attribut.

3.3.1.3. Titel des MOBs

Neben den frei definierbaren Attributen besitzt jedes MOB einen Titel. Da dieser für die Darstellung von Suchergebnissen und Galerien benötigt wird, wird er sowohl in der Datenbank als auch im MMO-Client separat verwaltet. Daher wird ihm innerhalb der XML-Struktur ein eigener Abschnitt gewidmet, der von dem *title*-Tag eingeschlossen wird, vgl. XML-Beispiel 3.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">
  <!-- ... Weitere Inhalte ... -->
  <title>Aneurysma, A. iliaca  mit V. cava Beteiligung</title>
  <!-- ... Weitere Inhalte ... -->
</medobject>
```

XML-Beispiel 3: Das title-Tag zur Abbildung des MOB-Titels

3.3.1.4. Zugriffsrechte und die Autorenliste

Die Zugriffsrechte, die mit einem MOB assoziiert sind, werden in einem Element namens *accesslist* abgebildet. Dabei gilt beim Öffnen des MOBs aus der Datenbank die Konvention, daß diese Zugriffsliste nur dann in den XML-Code integriert wird, wenn der Benutzer Schreibrechte für das Objekt besitzt. Denn nur in diesem Fall ist es notwendig, diese Informationen mit dem MMO-Client zu verarbeiten. Außerdem wäre es aus Sicht der Datensicherheit nicht unbedenklich, die Zugriffsrechte an einen nicht schreibberechtigten User zu übertragen. Unabhängig davon, daß die Client-Software diesen Teil des XML-Dokuments ohne vorliegende Schreibrechte ignoriert, könnte ein technisch geschickter Benutzer die XML-Informationen, die das Backend an seinen Client schickt, abfangen und so Informationen über die Zugriffsrechte erlangen.
Eine fehlende *accesslist* dient insbesondere als Signal für den MMO-Client, daß der MOB nur für den Lesezugriff geöffnet werden darf.

Innerhalb des *accesslist*-Tags werden die Zugriffsrechte für die Mitglieder der Autorenliste, die Gruppe der Institutsangehörigen und alle weiteren Benutzer über jeweils ein leeres Element namens *access* dargestellt (vgl. XML-Beispiel 4). Dabei wird über das Attribut *group* festgelegt, auf welche der drei Gruppen sich die Information bezieht. Dabei kann *group* die Werte *authors*, *institute* und *world* annehmen.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <accesslist>
    <access group="authors" value="write"/>
    <access group="institute" value="read"/>
    <access group="world" value="none"/>
  </accesslist>

  <!-- ... Weitere Inhalte ... -->

</medobject>
```

XML-Beispiel 4: Abbildung der Zugriffsrechte in XML

Über das Attribut *value* wird die Ausprägung der Zugriffsrechte der in *group* gewählten Gruppe festgelegt. Hierfür können die Werte *none* für keinen Zugriff, *read* für Lesezugriff und *write* für Vollzugriff verwendet werden.
Die erste der oben aufgeführten Gruppen, die Autorenliste, wird in einem eigenen, durch das Tag *authorlist* eingegrenzten Bereich aufgeführt. Da diese Liste nicht nur für die Zugriffsrechte wichtig ist, sondern auch interessante Hinweise bezüglich der Urheber eines MOBs bereitstellt, wird sie auch dann übertragen, wenn der Benutzer nur Leserechte besitzt.

Innerhalb des *authorlist*-Elements werden die verschiedenen Autoren der Reihe nach durch jeweils ein Tag namens *author* repräsentiert (vgl. XML-Beispiel 5). Durch verschiedene XML-Attribute, wie z.B. *name*, *adress* etc., werden die Informationen des jeweiligen Autors in das *author*-Tag integriert.
Sämtliche Autoren, die in diese Liste eingefügt werden können, werden zentral im Backend-System verwaltet und können über eine Suchfunktion im MMO-Client ausgewählt und dem MOB hinzugefügt werden. Dabei wird insbesondere die datenbank-interne Identifikationsnummer des jeweiligen Autors übertragen. Diese wird durch das Attribut *a_id* im *author*-Tag repräsentiert. Beim Speichern des MOBs in der Datenbank reicht es aus, lediglich die das Attribut *a_id* in das *author*-Tag zu integrieren, da alle weiteren Informationen des entsprechenden Autors unmittelbar über diese Nummer ermittelt werden können. Lediglich beim öffnen eines MOBs sowie beim Abspeichern auf der Festplatte werden die in *name*, *adress* etc. gespeicherten Daten verwendet.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <authorlist>

    <author a_id="2" name="Filler" firstname="Timm J."
            salutation="PD Dr. med."
            institution="Institut für Anatomie, WWU Münster"
            adress="Vesaliusweg 2-4"
            zip="48149" town="Münster" country="Deutschland"
            email="filler@uni-muenster.de" fon="+49 251 8355226"
            fax="+49 251 8355241" />

    <author a_id="3" name="Ruppel" firstname="Markus"
```

```
            salutation="Dipl.-Math."
            institution="Institut für Medizinische Informatik
                        und Biomathematik, WWU Münster"
            adress="Domagkstr. 9"
            zip="48129" town="Münster" country="Deutschland"
            email="ruppelm@uni-muenster.de" fon="+49 251 8358407"
            fax="+49 251 8352259" />

  </authorlist>

  <!-- ... Weitere Inhalte ... -->

</medobject>
```

XML-Beispiel 5: Die Autorenliste in XML

3.3.1.5. Verzeichnis der Attribute

Die Attribute des MOBs werden durch das Element *attriblist* in die XML-Strukturen integriert (vgl. XML-Beispiel 6). Innerhalb dieses Tags wird jedes Attribut durch jeweils ein *attrib*-Tag repräsentiert. Wie bereits erwähnt, besteht jedes Attribut aus einem Namen und einer Ausprägung. Diese Informationen werden durch die Elemente *name* und *value* innerhalb des *attrib*-Tags eingeschlossen. Dabei ist es möglich, sowohl den Attributsnamen als auch den Attributswert durch die Zuweisung eines geeigneten UMLS-CUIs genauer zu spezifizieren. Dieser CUI wird durch ein XML-Attribut der Form *cui="Cxxxxxxx"* im *name*- bzw. *value*-Tag erfaßt. Wird dieses XML-Attribut weggelassen, so ist der jeweilige Attributsname bzw. –wert als Freitext zu interpretieren.
In XML-Beispiel 6 wird eine solche Attributsliste aufgeführt. Dabei wurden die ersten beiden Attribute vollständig über UMLS-Konzepte definiert. Lediglich die Ausprägung des dritten Attributs wurde als reiner Freitext angegeben.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <attriblist>
    <attrib>
      <name cui="C0178784">Organ</name>
      <value cui="C0018787">Herz</value>
    </attrib>
    <attrib>
      <name cui="C0348026">Diagnose</name>
      <value cui="C0021308">Infarkt</value>
    </attrib>
    <attrib>
```

```
      <name cui="C0029981">Eigentumsrecht</name>
      <value>Dr. med. Mustermann</value>
    </attrib>
  </attriblist>

  <!-- ... Weitere Inhalte ... -->

</medobject>
```

XML-Beispiel 6: Abbildung der zugewiesenen Attribute

3.3.1.6. Freitextkommentare

In Abschnitt 3.2.2 wurde die Integration eines Freitextkommentars in das MOB erörtert. Dieser wird über das XML-Tag *comment* innerhalb des Wurzelknotens *medobject* realisiert, vgl. XML-Beispiel 7.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <comment>
    Die Entwicklung der Arteriosklerose ist ein jahrzehntelanger
    Vorgang, der meist schon im 2. und 3. Lebensjahrzehnt beginnt,
    sich aber erst in der 2. Lebenshälfte als Krankheit manifestiert.
    In der schematischen Abbildung sind drei Phasen der Entwicklung
    dargestellt, die im Einzelfall mit unterschiedlicher
    Geschwindigkeit voranschreiten.
  </comment>

  <!-- ... Weitere Inhalte ... -->

</medobject>
```

XML-Beispiel 7: Einbindung von Freitextkommentaren

3.3.1.7. Verknüpfungen zu anderen MOBs

Wie bereits erwähnt gibt es zwei Arten von Verknüpfungen zu anderen MOBs. Zum einen kann eine Verknüpfung so angelegt werden, daß sie sich auf den gesamten Quell-MOB bezieht, zum anderen kann sie sich auf eine beliebige Annotation innerhalb des Quell-MOBs beziehen. Dabei wird die erstellte Relation im zweiten Fall zusätzlich auch in den Verknüpfungen, die sich auf den gesamten MOB beziehen, aufgeführt (vgl. Abschnitt 3.2.8.3).

Die auf den Gesamt-MOB bezogenen Relationen werden in einem Element namens *sourcejoinedlist* innerhalb von *medobject* aufgelistet (vgl. XML-Beispiel 8). Darin befinden sich verschiedene Tags mit dem Namen *rel*, die jeweils eine

der verwendeten Relationsbezeichnung repräsentieren. Da diese im Backend-System verwaltet werden, wird im *rel*-Tag nicht die explizite Bezeichnung der Relation angegeben, sondern dessen Identifikationsnummer aus der Datenbank. Diese wird durch das XML-Attribut *id* eingebunden. In XML-Beispiel 8 werden z.B. die Relationen „ist Vergrößerung von“ und „zur nächsten Station“ verwendet, die innerhalb der Datenbank die Identifikationsnummern 3 und 13 haben.

Innerhalb der *rel*-Tags werden die Ziel-MOBs aufgeführt, auf die bezüglich der jeweiligen Relation verwiesen wird. Jeder Ziel-MOB wird durch ein Tag namens *mob* eingebunden. Über das XML-Attribut *id* wird die Identifikationsnummer des MOBs innerhalb des Backends angegeben. In XML-Beispiel 8 wird z.B. bzgl. der Relation „ist Vergrößerung von“ auf die MOBs 260 und 262 referenziert.

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <sourcejoinedlist>
    <rel id="3">
      <mob id="260"/>
      <mob id="262"/>
    </rel>
    <rel id="13">
      <mob id="243">
        <context name="Grundkurs 'Arterielles Verschlussleiden'" />
      </mob>
    </rel>
  </sourcejoinedlist>

  <!-- ... Weitere Inhalte ... -->

</medobject>
```

XML-Beispiel 8: Auf den MOB bezogene Verknüpfungen

In Abschnitt 3.2.8.2 wurde erörtert, daß die Zusammenhänge, in denen Verknüpfung gelten sollen, durch die Integration von Kontexten eingeschränkt werden kann. Diese optionalen Zusatzinformationen werden durch Elemente names *context* unterhalb des jeweiligen *mob*-Elements eingebunden. Der Name des jeweiligen Kontexts wird dabei durch das Attribut *name* einbezogen. Da die verschiedenen Kontexte nicht im Backend, sondern in einer Textdatei auf dem Rechner des Anwenders verwaltet werden, enthält das Attribut *name* in diesem

Fall keine Datenbank-Identifikationsnummer, sondern die tatsächliche Bezeichnung.
In XML-Beispiel 8 wurde beispielsweise ein Kontext im Zusammenhang mit der Erstellung virtueller Rundgänge vergeben (vgl. 3.2.9.2). Dabei wurde das aktuelle MOB innerhalb des Rundgangs / Kontextes „Grundkurs 'Arterielles Verschlussleiden'“ mit dem MOB Nr. 243 unter der Relation „zur nächsten Station“ verknüpft.

Neben dieser *sourcejoinedlist* wird beim Herunterladen eines MOBs auch eine *targetjoinedlist* angegeben, die die gleiche Struktur wie die *sourcejoinedlist* hat, mit dem Unterschied, daß etwaige Kontexte nicht berücksichtigt werden. Während die *sourcejoinedlist* eine Übersicht über alle MOBs liefert, auf die vom aktuellen MOB aus verwiesen wird, repräsentiert die *targetjoinedlist* eine Übersicht über alle MOBs, die auf das aktuelle MOB verweisen.
Da diese Informationen vom aktuellen Stand der Datenbank abhängen und vom Autor des MOBs nicht unmittelbar verändert werden können, wird die *targetjoinedlist* beim Speichern des MOBs in der Datenbank oder auf der lokalen Festplatte des Anwenders nicht in den XML-Code integriert.

3.3.1.8. Multimediadateien und Annotationen

Der letzte Abschnitt der XML-Dokumente beschäftigt sich mit den im MOB erfaßten Multimediadateien. Diese werden der Reihe nach durch jeweils ein Tag namens *mediapanel* repräsentiert. Dieses Tag enthält das Attribut *mimetype*, in dem das Dateiformat der jeweiligen Multimediadatei angegeben wird (z.B. *image/jpeg* für JPG-Bilder, *video/x-msvideo* für AVI-Videos etc.).
Innerhalb von *mediapanel* werden der Reihe nach folgende Informationen erfaßt (vgl. XML-Beispiel 9):

- Unterschrift zur Multimediadatei
- Annotationen inkl. etwaiger Relationen samt Kontext
- Die Multimediadatei an sich

```
<medobject id="3153" creationdate="13.05.2002 12:11:22">

  <!-- ... Weitere Inhalte ... -->

  <mediapanel mimetype="image/jpeg">
    <caption>Arterielles Verschlussleiden Stadium IV</caption>
    <shape type="ellipse" points="470,252,510,289">
      <annotation/>
    </shape>
    <shape type="rectangle" points="145,89,322,197">
      <annotation>
         Dieser Ausschnitt zeigt eine durch Arteriosklerose
         bedingte Verengung der Arterie.
      </annotation>
      <rel id="3">
        <mob id="3018"/>
      </rel>
      <rel id="13">
        <mob id="343">
          <context name="Grundkurs 'Arterielles Verschlussleiden'" />
        </mob>
      </rel>
    </shape>
    <binary type="code">
            /9j/4AAQSkZJRgABAgEFUAVQAAD/7RA2UGhvdG9zaG9wIDMuM
            AAAAABAFUAAAAAEAAgVQAAAAAQACOEJJTQPzAAAAAAAIAAAAA
                        <!-- ... Gekürzt ... -->
            wfyro8R+7Ga6I1k/Fc4zV455HGpErosav08RodrtE/MfBEgcA
            39r95Odvukax9I9k5LQ6Z1Grv4Sna0weQ0Rq06T/1aaxv/9k=
    </binary>
  </mediapanel>

</medobject>
```

XML-Beispiel 9: Ein „Mediapanel“ zur Integration der Multimediadateien

Innerhalb von *mediapanel* wird zunächst das *caption*-Tag aufgeführt. In diesem wird die Unterschrift zur jeweiligen Multimediadatei eingebettet.

Falls es sich bei der Datei um ein Bild handelt, so ist es über die in Abschnitt 3.2.5 beschriebene Funktion möglich, dieses mit Hervorhebungen in Form von Rechtecken oder Ellipsen zu versehen. Diese Hervorhebungen werden der Reihe nach durch jeweils ein Tag namens *shape* abgebildet. Dieses enthält die Attribute *type* und *points*. Dabei kann *type* den Wert *ellipse* oder *rectangle* annehmen und beschreibt die Form der Annotation. Das Attribut *points* enthält die Koordinaten der Hervorhebung. Diese werden durch vier durch Kommata getrennte natürliche Zahlen beschrieben. Dabei geben die ersten beiden Zahlen

die linke obere und die beiden letzten Zahlen die rechte untere Ecke an, gemessen in Pixeln ausgehend von der linken oberen Ecke des Bildes.[1]
Innerhalb des *shape*-Tags wird der Annotationstext, mit der die Hervorhebung vom Benutzer kommentiert wurde, in einem Tag namens *annotation* erfaßt.
Des weiteren können von den Hervorhebungen aus Verknüpfungen zu anderen MOBs erstellt werden (vgl. Abschnitt 3.2.8.3). Falls diese nicht vorliegen, so enthält das *shape*-Tag wie im ersten Fall von XML-Beispiel 9 lediglich das *annotation*-Tag als Kinderknoten. Ansonsten werden die Relationen genauso wie im oben beschriebenen *sourcejoinedlist*-Tag in das *shape*-Tag eingebunden. Dabei werden die Elemente *rel*, *mob* und ggf. *context* wie in Abschnitt 3.3.1.7 verschachtelt.

Als letztes werden Informationen zur eigentlichen Multimediadatei in das *mediapanel* eingebunden. Dies wird über das Tag *binary* realisiert. Dabei wird entweder die gesamte Datei oder konkrete Informationen zur Dateiposition eingebunden.
Da binäre Daten nicht ohne weiteres in Textdateien integriert werden können, ist es im ersten Fall notwendig, diese zuvor durch den *„MIME base 64“*-Algorithmus [51] in reinen Text umzuwandeln (vgl. Abschnitt 4.2.2.2). Beim Speichern des MOBs in der Datenbank wird immer diese Methode verwendet, da sie es erlaubt, den gesamten MOB in einer einzigen Datei zu erfassen und en bloc an das Backend zu übermitteln. Daher ist es in diesem Fall nicht nötig, zusätzliche Attribute mit Informationen zur verwendeten Übertragungstechnik in das *binary*-Tag zu integrieren.
Anders verhält es sich, wenn das MOBs von der lokalen Festplatte des Benutzers oder aus dem Backend geladen wird. In diesen Fällen gibt es mehrere Möglichkeiten, die Multimediadateien einzubinden. Daher wird das *binary*-Tag mit dem Attribut *type* ausgestattet, welches angibt, auf welche Weise die Multimediadateien zur Verfügung gestellt werden. Grundsätzlich kann *type* einen der folgenden Werte annehmen:

[1] Im Falle einer Ellipse werden die Ecken des kleinsten Rechtecks, das die Ellipse umfaßt, angegeben.

- *localfile* = die Datei befindet sich im lokalen Dateisystem unter dem im *binary*-Tag angegebenen Pfad
- *code* = die Datei wurde als MIME-codierte Zeichenkette in das *binary*-Tag eingebettet
- *id* = die Datei hat in der Datenbank die im *binary*-Tag angegebene Identifikationsnummer
- *url* = die Datei kann unmittelbar unter der im *binary*-Tag angegebenen Webadresse heruntergeladen werden

Beim Laden eines MOBs aus der Datenbank bindet das Backend-System die Dateiinformation immer über *type="id"* ein. Der Wert dieser *id* wird vom MMO-Client automatisch in eine dynamische Webadresse umgewandelt, unter der die Multimediadatei im Backend abgerufen werden kann. Mittels dieser Adresse kann der Client die Datei herunterladen und am Bildschirm darstellen.
Die Konstellation *type="url"* wird zwar vom Client voll unterstützt, findet aber zur Zeit noch keine Anwendung. Dieser Mechanismus könnte für potentielle Weiterentwicklungen des Backend-Systems interessant werden. Wenn es darum geht, MOBs zu erfassen, deren Multimediadateien nicht in der Datenbank des Backends, sondern im Web abgelegt werden, kann diese Variante verwendet werden, um die Position der Datei in das MOB zu integrieren. Eine derartige Vorgehensweise wäre beispielsweise denkbar, falls es zu einem Datenaustausch mit dem HEAL-Projekt [16,29,57] käme.

Durch das Abspeichern eines MOBs im lokalen Dateisystem des Benutzers soll ein netzwerkunabhängiger Umgang mit dem MOB ermöglicht werden. Daher wird beim Speichern auf der Festplatte entweder die Methode *type="localfile"* oder *type="code"* verwendet.
Die Variante *localfile* hat den Vorteil, daß die MOB-Datei relativ klein ist und sehr schnell vom Client geladen werden kann. Allerdings kann sie nur auf dem Rechner des Anwenders verwendet werden, da nur dort die im *binary*-Tag angegebene Multimediadatei vorhanden ist.

Wird die Methode *code* verwendet, so kann die MOB-Datei relativ groß werden. Des weiteren kann sich der Ladevorgang beim Öffnen eines MOBs mit großen Dateien verzögern, da die MIME-codierten Inhalte zunächst wieder in Binärdateien zurück verwandelt werden müssen. Der Vorteil dieser Methode liegt darin, daß sämtliche Informationen des MOBs in einer Datei gekapselt werden. Auf diese Weise ist es beispielsweise möglich, die MOB-Datei per Email an einen Kollegen zu schicken, der sie auf seinem eigenen Rechner mit dem MMO-Client öffnen kann.

3.3.2. Modellierung von Galerien

Wie in Abschnitt 3.2.7 bereits erwähnt, bestehen Galerien im wesentlichen aus einer Liste von MOBs. Diese kann im XML-Format auf der Festplatte des Anwenders und in der Datenbank gespeichert werden. Dabei werden nicht die gesamten Inhalte der MOBs erfaßt, sondern lediglich deren Identifikationsnummern aus der Datenbank. Des weiteren ist es möglich, der Galerie einen Titel zuzuweisen.

XML-Dokumente zur Repräsentation von Galerien werden durch das *gallery*-Tag eingeschlossen (vgl. XML-Beispiel 10). Innerhalb dieses Tags wird zunächst der Titel der Galerie über das XML-Element *title* eingebunden. Daraufhin werden die einzelnen MOBs der Reihe nach durch jeweils ein leeres Tag namens *item* repräsentiert. Dabei enthält jedes *item*-Tag ein Attribut namens *id*, in dem die Identifikationsnummer des MOBs aus der Datenbank erfaßt wird.

```
<gallery>

  <title>Bilder zum arteriellen Verschlussleiden</title>

  <item id="260" />
  <item id="343" />
  <item id="243" />

</gallery>
```

XML-Beispiel 10: Eine Galerie mit drei Einträgen

3.3.3. Datenmodellierung im Zusammenhang mit der Datenbanksuche

In Abschnitt 3.2.6 wurde erklärt, daß eine Suche in der MMO-Datenbank aus den drei Schritten *Suchanfrage erstellen*, *Auswahl eines geeigneten Attributs* und *Auswahl der MOBs zum gewählten Attribut* besteht. Der Aufbau der Daten, die während dieses Prozesses zwischen dem Client und dem Backend ausgetauscht werden, wird in den XML-Beispielen 11 bis 13 skizziert.

3.3.3.1. Aufbau der Suchanfrage

Die Zusammenstellung der MOB-Attribute, nach denen in der Datenbank gesucht werden soll, wird durch das Tag *searchrequest* eingeschlossen (vgl. XML-Beispiel 11).

Über die XML-Attribute *login* und *loginsession* werden die Authentifizierungsdaten des Anwenders übergeben. Ähnlich der in Abschnitt 3.3.1.2 dargestellten Situation, ist diese Legitimation für die Wahrung der in Abschnitt 3.2.4 beschrieben Zugriffsrechte notwendig. Da sich die Zugriffsrechte auf ein MOB von Benutzer zu Benutzer unterscheiden, muß die Identität des Suchenden bei der Zusammenstellung der Suchergebnisse berücksichtigt werden.

```
<searchrequest login="ruppelm" loginsession="992868600761"
               oldsearchsession="-1">

  <fuzzy level="2" />

  <attrib searchoption="muss">
    <name cui="C0178784">Organ</name>
    <value cui="C0022646">Niere</value>
  </attrib>

  <attrib searchoption="kann">
    <name cui="C0012634">Krankheit</name>
    <value cui="C0231179">Insufficiency</value>
  </attrib>

  <attrib searchoption="kann">
    <value>Farbfoto</value>
  </attrib>

</searchrequest>
```

XML-Beispiel 11: Formulierung einer Anfrage zur Datenbanksuche

Durch die Verwendung des *Multiple Document Interface* (vgl. Abschnitt 3.1.3) ist es möglich, mehrere Datenbankabfragen parallel mit dem MMO-Client durchzuführen. Da jede dieser Abfragen aus mehreren Kommunikationsschritten zwischen Client und Backend besteht, muß das Backend Buch darüber führen, welche Daten zu welcher Suchanfrage gehören. Aus diesem Grund wird jede Datenbanksuche vom Backend mit einer eindeutigen Identifikationsnummer („Suchsession Key“) versehen und in einer sog. *Suchsession* verwaltet. Beim Abschicken der Suchanfrage wird eine solche Session vom Backend erzeugt. Wird im Laufe der Recherche eine bereits vorhandene Suchanfrage verändert und erneut abgeschickt, so muß die bisher zugehörige Suchsession vom Backend verworfen und durch eine neue ersetzt werden. Aus diesem Grund wird das *searchrequest*-Tag mit dem Attribut *oldsearchsession* versehen, in dem die Identifikationsnummer der zu löschenden Suchsession angegeben wird. Falls die Suchanfrage erstmalig abgeschickt wird, so bekommt *oldsearchsession* den Wert *-1*.

Innerhalb des *searchrequest*-Tags wird zunächst das leere Tag *fuzzy* in den XML-Code eingebettet. Über das zugehörige XML-Attribut *level* wird der Unschärfegrad der Suche angegeben.
Danach werden sukzessiv die MOB-Attribute aufgelistet, nach denen gesucht werden soll. Jedes dieser Attribute wird durch ein Tag namens *attrib* repräsentiert. Diese haben im wesentlichen die gleiche Struktur, wie die *attrib*-Tags aus den Attributsverzeichnissen der MOBs (vgl. Abschnitt 3.3.1.5). Insbesondere werden die UMLS-Konzepte zur Charakterisierung der Attributsnamen bzw. –werte ggf. über XML-Attribute der Form *cui=“Cxxxxxxx“* in die Tags *name* und *value* integriert.
Darüber hinaus enthält jedes *searchrequest*-Tag ein *searchoption*-Attribut, das die Werte *„kann“* oder *„muss“* annehmen kann. Hierdurch wird die in Abschnitt 3.2.6.1 erörterte Unterscheidung von „kann“- und „muß“-Suchattributen abgebildet.

3.3.3.2. Aufbau der Attributsübersicht

Als Reaktion auf eine Suchanfrage liefert das Backend-System eine Übersicht über die Suchattribute, die in den gefundenen MOBs enthalten sind (vgl. Abschnitt 3.3.3.2). Diese wird in Form eines zweistufigen XML-Baums vom Backend an den Client übermittelt. Dieser Baum wird durch das Element *searchresult* eingeschlossen (vgl. XML-Beispiel 12), wobei die Identifikationsnummer der vom Backend neu erzeugen Suchsession über das XML-Attribut *searchsession* angegeben wird.

```
<searchresult searchsession="7584936345784">

  <item type="attribname" caption="organ" cui="C0178784">
    <item type="attribvalue" caption="Niere" cui="C0022646" />
    <item type="attribvalue" caption="Urogenital" cui="C0521380" />
  </item>

  <item type="attribname" caption="Bildart" cui="">
    <item type="attribvalue" caption="Farbfoto" cui="" />
  </item>

  <item type="attribname" caption="Medium" cui="">
    <item type="attribvalue" caption="Farbfoto" cui="" />
  </item>

</searchresult>
```

XML-Beispiel 12: Attributsübersicht als Ergebnis der Suche

Der Baum für die Darstellung der Attribute wird durch eine Sequenz verschachtelter *item*-Tags repräsentiert. Dabei gibt das XML-Attribut *type*, das die Werte *attribname* und *attribvalue* annehmen kann, an, ob es sich um einen Attributsnamen oder –wert handelt. Diese Eigenschaft wird zwar schon aus der hierarchischen Position des *item*-Tags unmittelbar ersichtlich (Name = erste Ebene im Baum, Wert = zweite Ebene im Baum), da aber in zukünftigen Versionen der Software u.U. auch komplexere Varianten dieses Baums auftreten können, wurde das *type*-Attribut schon jetzt integriert.

Die XML-Attribute *caption* und *cui* geben den Text und den CUI des UMLS-Konzepts des jeweiligen *items* an. Falls es sich dabei um reinen Freitext handelt, so ist das *cui*-Attribut leer.

3.3.3.3. Aufbau der MOB-Liste

Nachdem der Anwender ein Attribut aus dem o.a. Baum ausgewählt hat, wird vom Backend eine Liste mit den zugehörigen MOBs zusammengestellt. Diese wird in das *imagelist*-Tag eingebettet (vgl. XML-Beispiel 13). Alle gefundenen MOBs werden der Reihe nach in Form eines *image*-Tags aufgeführt.

Wie in Abschnitt 3.2.6.3 beschrieben, werden zu jedem MOB Informationen bzgl. der Datenbank-Identifikationsnummer, des Titels und der Schreibrechte eingebunden. Diese Informationen werden in den *image*-Tags durch die Attribute *id*, *title* und *permission* integriert.

Im *title*-Attribut wird der in Abschnitt 3.3.1.3 beschriebene Titel des MOBs aufgeführt.

```
<imagelist>

  <image title="Sagitalschnitt einer insuffizienten Niere" id="3017"
         permission="w" />
  <image title="Ultraschallbild der Niere" id="3024"
         permission="r"/>
  <image title="Das Urogenitalsystem im Überblick" id="3041"
         permission="r"/>

  <!-- Ggf. weitere image-Tags -->

</imagelist>
```

XML-Beispiel 13: Nach Auswahl eines Attributs werden die MOBs aufgelistet

Im Attribut *permission* werden die Zugriffsrechte des Benutzers bzgl. des jeweiligen MOBs abgebildet. In diesem Zusammenhang können die Werte *„r“* für „nur lesen“ und *„w“* für „lesen und schreiben“ auftreten.

Die Identifikationsnummer des MOBs, die im XML-Attribut *id* angegeben wird, wird insbesondere dazu verwendet, die Miniaturansicht des MOBs vom Backend herunterzuladen und in der MOB-Liste darzustellen. Außerdem dient sie als Schlüssel, der an das Backend übermittelt wird, wenn der Anwender ein gefundenes MOB öffnen möchte. In diesem Fall werden die in Abschnitt 3.3.1 erörterten Daten für das jeweilige MOB vom Backend an den Client übertragen.

4. Ergebnisse

In diesem Kapitel wird die eigentliche MMO-Client-Applikation vorgestellt, in der die im vorherigen Kapitel erörterten Funktionen und Modelle umgesetzt wurden. Neben einem Gesamtüberblick über die Oberfläche werden auch die Verwaltung und Verknüpfung von Mobs, die Suche in der Datenbank und die Speicherung von MOBs in Galerien erläutert. Anschließend werden die Funktion zum Export virtueller Rundgänge anhand des Projekts LOTSE erörtert.

4.1. Der MMO-Client im Überblick

Im Gegensatz zu dem in [27] vorgestellten Prototyp-Client, der in Form eines Java-Applets umgesetzt wurde (vgl. Abb. 3), handelt es sich beim MMO-Client um eine Applikation für die Betriebssysteme *Microsoft Windows 98* und *NT* sowie deren kompatible Nachfolgesysteme.
Für die Umsetzung wurde die auf der Programmiersprache *Object Pascal* basierende Entwicklungsumgebung *Borland Delphi 5 Professional* verwendet. Diese wurde um die im Internet verfügbaren Softwarekomponenten *Extended Document Object Model* (*XDOM)* [44] und *TGIFImage* [56] erweitert.
XDOM ist eine Erweiterungsbibliothek zur Verarbeitung von XML-Dokumenten, die auf dem W3C-Standard *Document Object Model (DOM)* [60] basiert. Das DOM ist „ein plattform- und sprachunabhängiges Interface, das es Programmen ... erlaubt, dynamisch auf den Inhalt, die Struktur und den Stil von (XML-) Dokumenten zuzugreifen“ [60].
Die *TGIFImage*-Komponente ermöglicht die Verarbeitung von GIF-Bildern mit *Delphi*. Da dieses Dateiformat im Web sehr weit verbreitet ist, sollte es besonders in Hinblick auf die Wiederverwertbarkeit der multimedialen Objekte vom MMO-Client unterstützt werden.

Wie bereits erwähnt, wurde das Multiple Document Interface (MDI) für die Umsetzung des MMO-Clients verwendet. MDI-Applikationen bestehen aus einem Hauptfenster, in dem mehrere Unterfenster, sog. MDI-Children-Fenster, geöffnet werden können. Hierdurch ist es möglich, mehrere Arbeitsabläufe

parallel durchzuführen. Insgesamt wurde der MMO-Client mit drei Arten von MDI-Children-Fenstern ausgestattet: Eine für die MOBs, eine für die Galerien und eine für die Suche in der Datenbank. Abb. 4 zeigt eine Gesamtübersicht über die Oberfläche des MMO-Clients, in dessen Hauptfenster jeweils eine Instanz dieser drei MDI-Children-Klassen zu sehen ist. Die meisten Funktionen des Clients können entweder innerhalb der MDI-Children-Fenster oder alternativ über das Hauptmenü aufgerufen werden. Um die Arbeitsabläufe zu vereinfachen, können diese Menüpunkte auch über vordefinierte Tastenkombinationen (sog. Shortcuts) oder über kleine Icons, die sich in einer Werkzeugleiste im Seitenkopf befinden, aufgerufen werden.

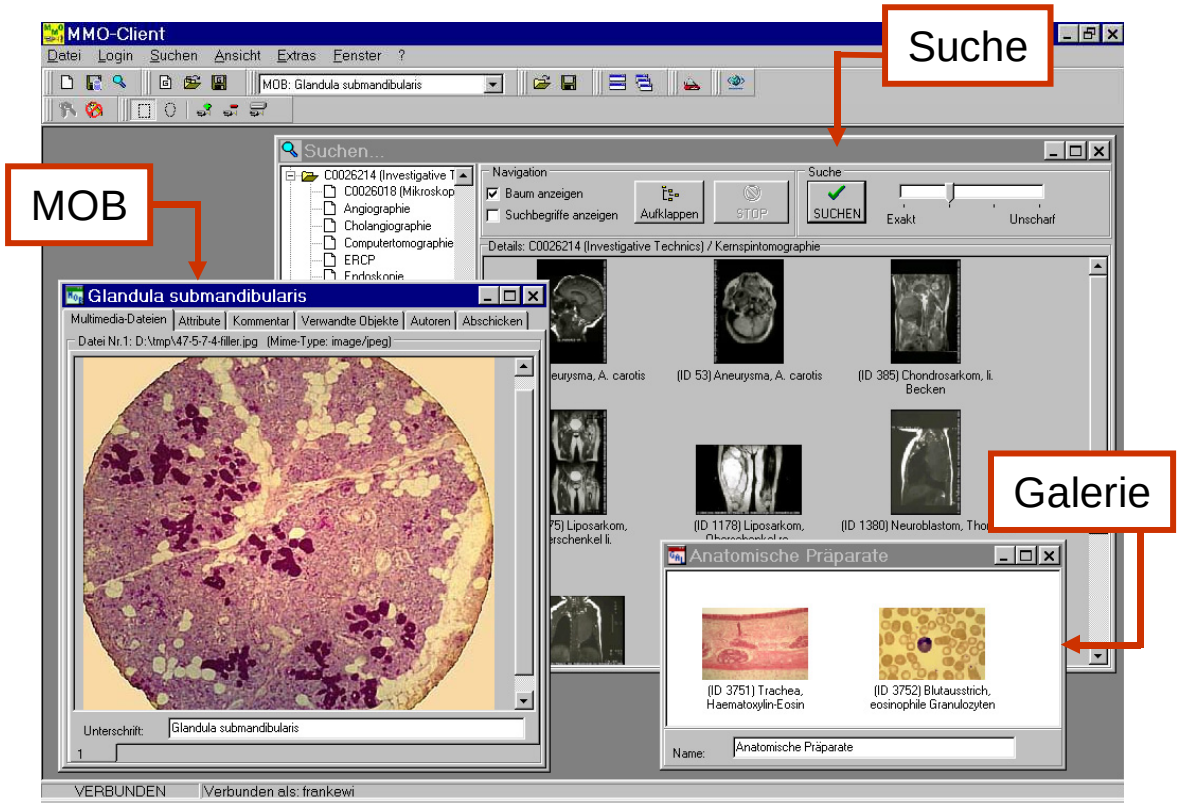


Abb. 4: Übersicht über den MMO-Client. Im Hauptfenster werden die verschiedenen MDI-Children-Fenster dargestellt.

Bevor der MMO-Client verwendet werden kann, muß er auf dem Rechner des Benutzers installiert werden. Um auch den Anwendern, die keinen administrativen Zugriff auf Ihren Arbeitsplatzrechner haben, das Arbeiten mit der Software zu ermöglichen, kann die Installation durch einfaches Kopieren

und Anpassen bestimmter Dateien vorgenommen werden. Hierfür sind weder Administratorenrechte noch Zugriffe auf zentrale Systemkomponenten notwendig.

4.2. Abbildung von MOBs im MMO-Client

4.2.1. Jedes MOB entspricht einem Fenster

Die Datenverwaltung im MMO-Client wurde so organisiert, daß alle Daten, die zusammen gehören, auch stets zusammenhängend dargestellt werden. Aus diesem Grund wird jedes MOB, inklusive seiner Meta-Daten, in einem eigenen MDI-Children-Fenster dargestellt. Jedes dieser Fenster wird durch virtuelle Karteireiter in thematische Unterbereiche unterteilt (vgl. Abb. 5), wodurch eine komfortable Verwaltung der Inhalte ermöglicht wird. Dabei werden folgende Bereiche unterschieden:

- *Multimediadateien:* Verwaltung der Bilder, Videodateien, Textdokumente etc., die im MOB erfaßt wurden (siehe Abschnitt 4.2.2)
- *Attribute:* Verwaltung der frei definierbaren Attribute (siehe Abschnitt 4.2.3)
- *Kommentar:* Verwaltung der in den Abschnitten 3.2.2 und 3.3.1.6 besprochenen Kommentare in einem Freitextfeld
- *Verwandte Objekte:* Darstellung der verknüpften MOBs (siehe Abschnitt 4.5.3)
- *Autoren:* Verwaltung der in den Abschnitten 3.2.4 und 3.3.1.4 beschriebenen Autorenliste
- *Abschicken:* Verwaltung der in den Abschnitten 3.2.4 und 3.3.1.4 erörterten Zugriffsrechte, sowie Möglichkeiten zum Speichern des MOBs in der Datenbank

4.2.2. Multimediadateien im MOB-Fenster verwalten

4.2.2.1. Organisation der Dateien im MOB

Im ersten der o.a. virtuellen Karteireiter werden die im MOB erfaßten Multimediadateien verwaltet. In diesem Zusammenhang ist es möglich, Dateien hinzuzufügen und zu entfernen, die Reihenfolge zu verändern und beliebige Multimediadateien auf der lokalen Festplatte des Anwenders abzuspeichern. Dabei ist die zuletzt genannte Funktion besonders wichtig für die Wiederverwertung der Objekte in den eigenen Lehr- und Forschungsmaterialien, da eine Datei auf diese Weise aus dem MOB extrahiert und mit anderen Programmen weiterverarbeitet werden kann. Der Umfang der vom MMO-Client unterstützten multimedialen Dateiformate erstreckt sich von Bildern (gif-, jpg- und bmp-Dateien) über Videos (avi- und mpg-Dateien), Animationen (swf-Dateien) und Sounds (wav- und mp3-Dateien) bis hin zu textbasierten Dokumenten (doc-, pdf-, rtf-, txt- und html-Dateien).
Die Navigation in den erfaßten Dateien eines MOBs erfolgt über untergeordnete, kleinere virtuelle Karteireiter, die sich im unteren Fensterbereich befinden (vgl. Abb. 5). Über die Beschriftungen dieser Reiter werden die Multimediadateien durchnumeriert. Die o.a. Funktionen für diese Dateien können sowohl über ein Kontextmenü, daß sich über ein Klick mit der rechten Maustaste auf den entsprechenden Karteireiter öffnet, als auch über das Hauptmenü (bzw. über die Werkzeugleiste / Shortcuts) aufgerufen werden.

Wie in Abschnitt 3.2.5 erwähnt, können Bilddateien mit Annotationen versehen werden. Hierfür wurde eine Zeichenfunktion in den MMO-Client integriert, mit der elliptische und rechteckige Hervorhebungen mit der Maus erstellt werden können. Da diese Hervorhebungen als separate Meta-Daten verwaltet und lediglich als Überlagerung der Bilddatei dargestellt werden, ist eine nachträgliche Bearbeitung der Annotationen möglich. So können diese beispielsweise mit der Maus verschoben und über ein Kontextmenü gelöscht oder mit einem Annotationstext versehen werden.

In Abb. 5 wurden je eine rechteckige und eine elliptische Hervorhebung integriert. Wie man sieht, sind die Hervorhebungen nicht einfarbig, sondern passen sich automatisch der jeweiligen Hintergrundfarbe an, so daß sie unabhängig von der Färbung der Bilddatei immer sichtbar sind.

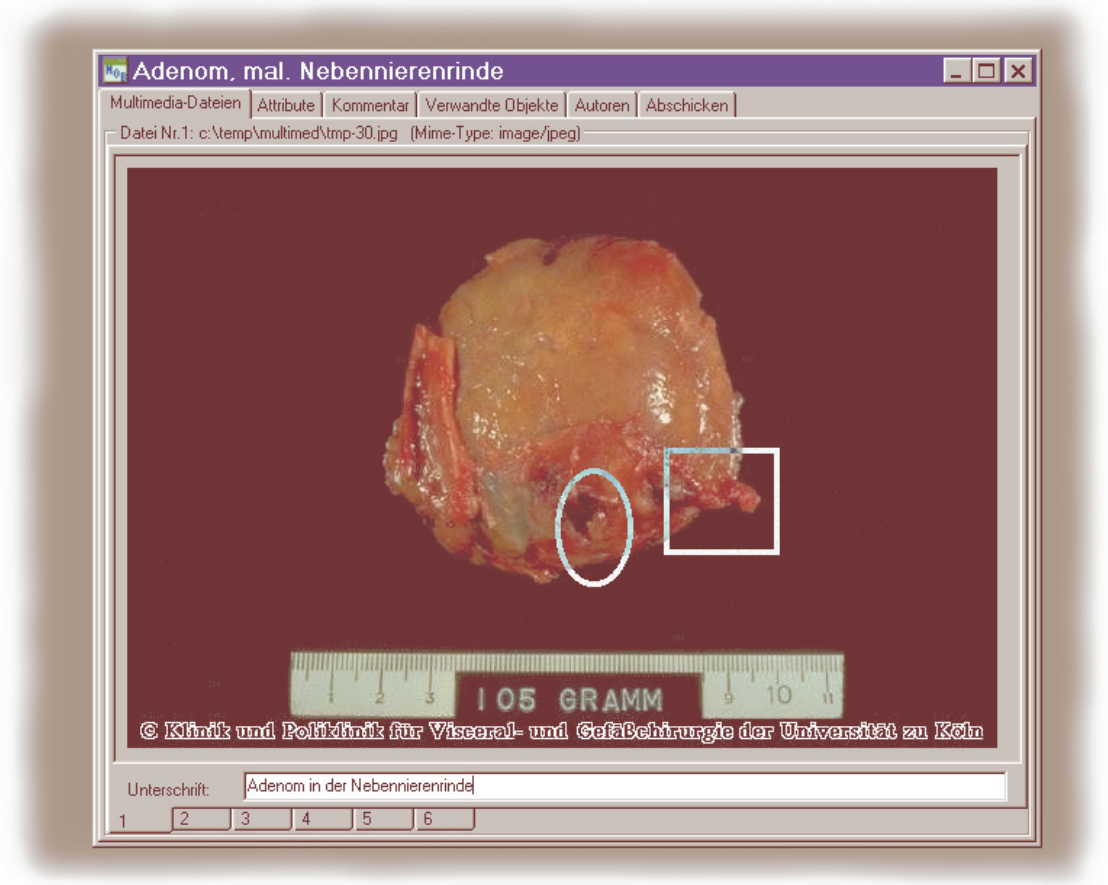


Abb. 5: Jedes MOB wird in einem MDI-Children-Fenster dargestellt. Im oberen Fensterbereich wird das MOB über virtuelle Karteireiter in verschiedene Bereiche unterteilt. Die verschiedenen Multimediadateien werden durchnumeriert und können im unteren Fensterbereich ausgewählt werden. Bilddateien können mit Annotationen versehen werden.

4.2.2.2. Umwandlung von Binärdaten in Textdaten

Um ein MOB in der Datenbank oder auf der Festplatte speichern zu können, müssen die binären Multimediadateien in den entsprechenden XML-Text eingebettet werden. Da binäre Informationen in der Regel nicht ohne weiteres als reiner Text interpretiert werden können, ist es notwendig, diese über geeignete Algorithmen entsprechend umzuwandeln. Im vorliegenden Fall

werden die Binärdaten nach dem „*MIME base 64“*-Standard [51] codiert. Dieser Algorithmus erlaubt es, den Inhalt beliebiger Dateien in eine Folge von Zeichen, die in jedem gängigen Zeichensatz vorhanden sind[2], zu verwandeln und auf eindeutige Weise zurück zu transformieren. In *Delphi‘s* Komponentenbibliothek gibt es eine Komponente namens *TNMUUProcessor*, die eine Implementierung dieser Routine bereitstellt.

Eine nähere Betrachtung des *MIME base 64*-Algorithmus zeigt, daß die Anzahl der mit der Codierung verbundenen Rechenoperationen und damit auch die benötigte Rechenzeit (nahezu) linear in Abhängigkeit von der Größe der zu verarbeitenden Datei verläuft [51]. Verschiedene Versuche mit der *TNMUUProcessor*-Komponente haben jedoch ergeben, daß die verbrauchte Rechenzeit deutlich überlinear in Bezug auf die Dateigröße wächst, was möglicherweise auf eine unsaubere Umsetzung dieser Komponente zurückzuführen ist. Diese Verzögerung kann besonders bei größeren Dateien zu empfindlichen Einbußen bezüglich der Benutzerfreundlichkeit führen, da der Anwender lange Wartezeiten hinnehmen muß.

Aus diesem Grund wurde für den MMO-Client eine Prozedur entwickelt, die es ermöglicht, die zu transformierende Datei in kleinere Segmente (*Slices*) aufzuteilen und diese sukzessiv mit einer Instanz der *TNMUUProcessor*-Komponente zu codieren. Dabei wurde die Länge der Slices so geschickt gewählt, daß die Aneinanderreihung der codierten Segmente exakt der Codierung der gesamten Datei entspricht. Auf diese Weise konnte der Transformationsprozeß deutlich beschleunigt werden.

Des weiteren wird die MIME-Transformation automatisch im Hintergrund durchgeführt, sobald die Multimediadatei in das MOB geladen wird[3]. Auf diese Weise kann der Benutzer wie gewohnt mit dem MMO-Client weiterarbeiten, während die Transformation durchgeführt wird. In Verbindung mit der o.a. Beschleunigung kann der Umwandlungsprozeß daher meistens abgeschlossen werden, bevor der Benutzer die weitere Bearbeitung des MOBs erledigt hat. So können unnötige Wartezeiten vermieden werden.

[2] Man nennt diese Zeichen auch das „MIME-Alphabet“.
[3] Dabei werden die Dateien der Reihe nach in einem eigenen *Thread* transformiert.

4.2.3. Attribute als Freitext oder über das UMLS definieren

Im Karteireiter „Attribute“ können die unter 3.2.3 vorgestellten Attribute des MOBs verwaltet werden. Dabei ist es möglich, sowohl die Attributsnamen, als auch die Ausprägungen frei zu definieren. Diese Angaben können entweder als Freitext eingegeben oder über das UMLS ausgewählt werden. In Abb. 6 wird die Vorgehensweise skizziert.

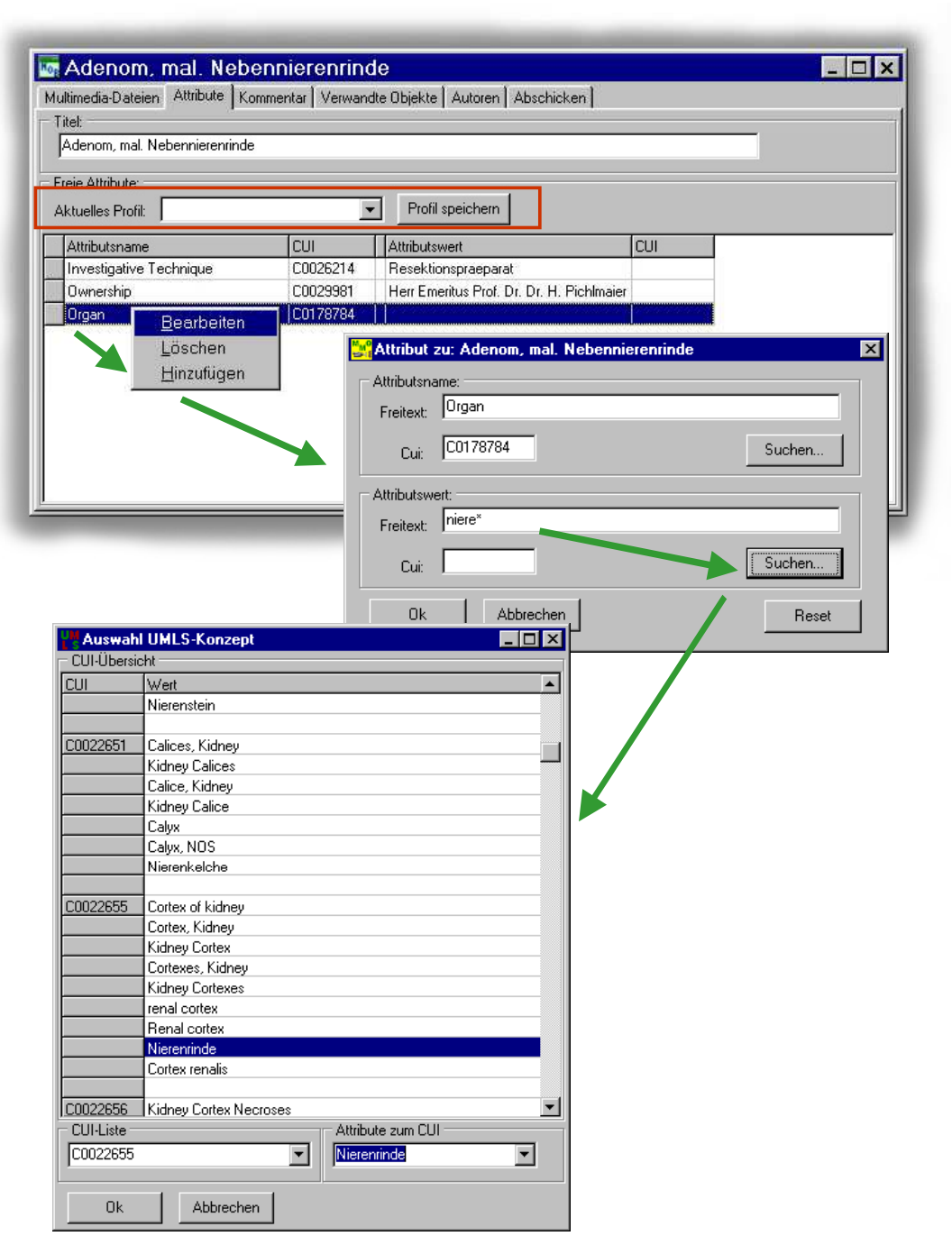


Abb. 6: Zuweisung eines Attributs über das UMLS

Bis auf den Titel des MOBs, der aufgrund seiner gesonderten Funktion (vgl. Abschnitt 3.3.1.3) in einem separaten Textfeld erfaßt wird, werden alle Attribute in einer Tabelle aufgelistet. Die Verwaltung der Inhalte dieser Tabelle erfolgt über ein Kontextmenü, über das Attribute gelöscht, hinzugefügt und bearbeitet werden können. Beim Aufruf der beiden letzten Funktionen öffnet sich ein zweigeteiltes Dialogfenster, in dem sowohl der Name, als auch die Ausprägung des Attributs festgelegt werden können. Dabei können die jeweiligen Angaben entweder direkt in die Tabelle übernommen und somit als Freitext interpretiert werden, oder sie werden für eine Suche im UMLS verwendet. Hierfür steht ein entsprechender Button zur Verfügung, über den sich ein weiteres Dialogfenster öffnet. In diesem Fenster werden die UMLS-Suchergebnisse wie in Abschnitt 3.2.3 beschrieben dargestellt, so daß der Anwender einen geeigneten CUI und einen bevorzugten Begriff auswählen kann, die dann für den Attributsnamen bzw. den -wert eingesetzt werden.

Falls ein Anwender mehrere MOBs aus ein und demselben medizinischen Fachgebiet im System ablegen möchte, so wird er diese vermutlich mit sehr ähnlichen Attributen versehen. Da es extrem aufwendig wäre, für jedes MOB die gleichen Attribute jedesmal von neuem eingeben zu müssen, wurde der MMO-Client mit einer Funktion zum Verwalten von sog. *„Attributsprofilen“* ausgestattet. Diese erlaubt es, die aktuellen Inhalte der Attributstabelle unter einem beliebigen Namen abzuspeichern und sie in anderen MOBs wiederzuverwenden. Hierbei braucht die Tabelle nicht vollständig zu sein, so daß sich der Benutzer Attributsvorlagen erstellen kann, die dann in jedes beliebige MOB eingelesen und individuell vervollständigt werden können. Die Verwaltung der Attributsprofile erfolgt über eine Auswahlliste oberhalb der Attributstabelle (vgl. Abb. 6, rot markierter Bereich).

4.3. Datenbanksuche mit dem MMO-Client

Für die Suche in der MMO-Datenbank steht dem Benutzer eine eigene Klasse von MDI-Children-Fenstern (im folgenden „Suchfenster“ genannt) zur Verfügung. Hierdurch ist es möglich, verschiedene Datenbankabfragen parallel durchzuführen, wobei jede dieser Abfragen im MMO-Client durch jeweils ein Suchfenster repräsentiert wird. Insbesondere gibt es zu jedem dieser Fenster genau eine assoziierte Suchsession im Backend-System (vgl. Abschnitt 3.3.3.1). Jedes Suchfenster ist in mehrere Bereiche unterteilt, die zur Durchführung des mehrstufigen Suchprozesses und zur Darstellung der jeweiligen Ergebnisse dienen (vgl. Abb. 7).

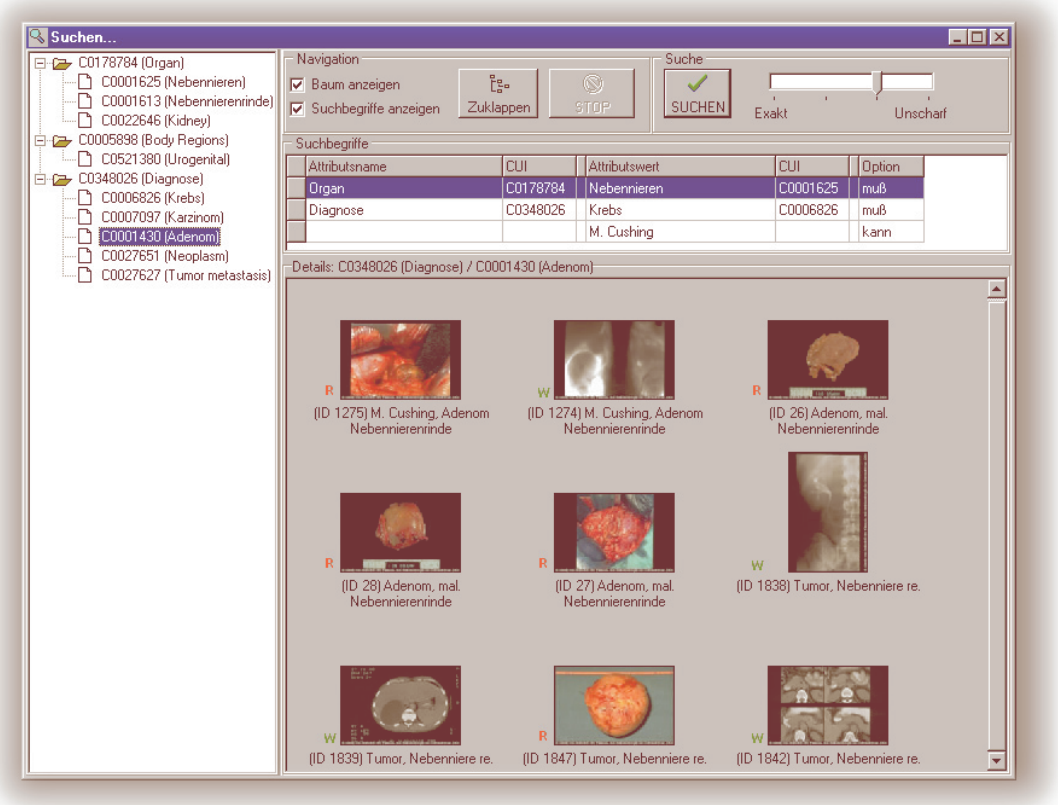


Abb. 7: Fenster für die Suche in der Datenbank

Wie in Abschnitt 3.2.6 beschrieben, müssen zunächst die Attribute, nach denen gesucht werden soll, festgelegt werden. Hierfür steht dem Anwender im Bereich „Suchbegriffe“ die gleiche Tabelle wie für die frei definierbaren Attribute der MOBs (vgl. Abb. 6) zur Verfügung. Insbesondere wurden die Funktionen zum

Hinzufügen, Entfernen und Verändern der Einträge übernommen. Darüber hinaus wurde die Spalte „Option“ hinzugefügt, die über das Kontextmenü zeilenweise mit den Werten „kann“ und „muß“ versehen werden kann. Hierdurch wird die in Abschnitt 3.2.6.1 erläuterte logische Verknüpfung der einzelnen Suchbegriffe festgelegt.

Nachdem die Suchanfrage formuliert wurde, kann der Benutzer über einen virtuellen Schieberegler im Bereich „Suche“ den Unschärfegrad der Datenbankabfrage festlegen. Dieser Regler kann vier verschiedene Positionen annehmen, wobei die Stellung ganz links einer exakten Suche (Unschärfegrad = 1) und die Stellung ganz rechts einer maximal unscharfen Suche (Unschärfegrad = 4) entspricht.

Wenn dieser Schritt abgeschlossen ist, stehen alle notwendigen Information für die Suche bereit, und die Suchanfrage kann über den Button „SUCHEN“ an das Backend-System geschickt werden. Daraufhin leitet das Backend den in Abschnitt 3.2.6.2 beschriebenen Suchprozeß ein und liefert eine Übersicht über die relevanten Attribute an den MMO-Client zurück. Diese wird im linken Bereich des Suchfensters in einer Baumkomponente, die der Darstellung der Ordnerhierarchie eines Dateisystems im Programm „Windows Explorer“ der Firma Microsoft ähnelt, integriert. Dabei bilden die Attributsnamen die erste Ebene des Baums und werden mit einem gelben „Ordner“-Symbol versehen. Den Attributswerten, die durch Aufklappen entsprechender Pfade unterhalb der Attributsnamen zu finden sind, wird ein weißes „Dokument“-Symbol zugeordnet. Auf diese Weise können die Attributsnamen und –werte auf den ersten Blick visuell unterschieden werden.

Mit Hilfe dieses Attributbaums kann der Anwender komfortabel in den gefundenen Attributen navigieren und per Mausklick eine geeignete Konstellation aus Attributsname und –wert auswählen. Daraufhin werden die gefundenen MOBs, die sich hinter dem entsprechenden Attribut verbergen, im Bereich „Details“ aufgelistet. Als Orientierungshilfe für den Anwender wird das ausgewählte Attribut zusätzlich in die Überschriftszeile dieses Bereichs neben dem Wort „Details“ integriert.

Jedes MOB in der Liste wird durch eine kleine Grafik repräsentiert. Sofern in dem jeweiligen MOB nur eine einzelne Bilddatei erfaßt wurde, wird diese in Form einer Miniaturansicht dargestellt (vgl. Abschnitt 3.2.6.3). Für alle anderen MOBs bietet das Backend ein breites Repertoire an vordefinierten Miniaturabbildungen, die nach Bedarf für die verschiedenen MOBs verwendet werden. Darüber hinaus werden die Identifikationsnummer und der Titel des MOBs sowie ein Hinweis auf die Zugriffsrechte angegeben. Dabei bedeutet ein rotes „R“, daß nur lesend auf das MOB zugegriffen werden darf, während ein grünes „W“ für Lese- und Schreibrechte steht.

Mit der Maus können beliebige MOBs aus der Liste der Ergebnisse ausgewählt und über ein Kontextmenü zur Ansicht oder Weiterverarbeitung geöffnet werden.

Um die Durchführung der Suche und die Darstellung der Ergebnisse zu vereinfachen, wurde der Bereich „Navigation“ eingerichtet, in welchem dem Benutzer verschiedene Hilfsfunktionen zur Verfügung gestellt werden. Beispielsweise ist es mit Hilfe des Buttons „Auf-/Zuklappen“ möglich, sämtliche Pfade des Attributbaums vollständig anzuzeigen bzw. zu verbergen. Über die Kontrollkästchen „Baum anzeigen“ und „Suchbegriffe anzeigen“ kann der Anwender den Attributbaum und die Tabelle der Suchbegriffe ein- und ausblenden. Nach Auswahl eines geeigneten Attributs kann so ein größtmöglicher Teil des Fensters für die Darstellung der u.U. sehr umfangreichen MOB-Liste verwendet werden. Der Aufbau dieser Liste kann über den „STOP“-Button jederzeit unterbrochen werden, z.B. wenn sich herausstellt, daß das gesucht MOB bereits in der Liste aufgeführt wurde und weitere Ergebnisse für den Benutzer nicht mehr von Interesse sind.

4.4. Verwaltung von MOBs in Galerien

Auch Galerien werden durch jeweils eine Instanz einer speziellen Klasse von *MDI-Children*-Fenstern repräsentiert. Diese bestehen aus einem „Container“, der den größten Teil des Fensters einnimmt, und einem Textfeld für den in Abschnitt 3.3.2 beschriebenen Titel der Galerie (Abb. 8). In dem Container

werden die MOBs verwaltet und dargestellt, wobei neben den Miniaturabbildungen der MOBs auch ihre Titel und Datenbank-IDs angezeigt werden. Um eine Galerie mit Inhalten zu bestücken, können ein oder mehrere MOBs

- aus der Ergebnisliste einer Datenbankabfrage oder
- aus einer anderen Galerie

mit der Maus markiert und per Drag-and-Drop über dem Container der Galerie „fallengelassen“ werden (Abb. 8).

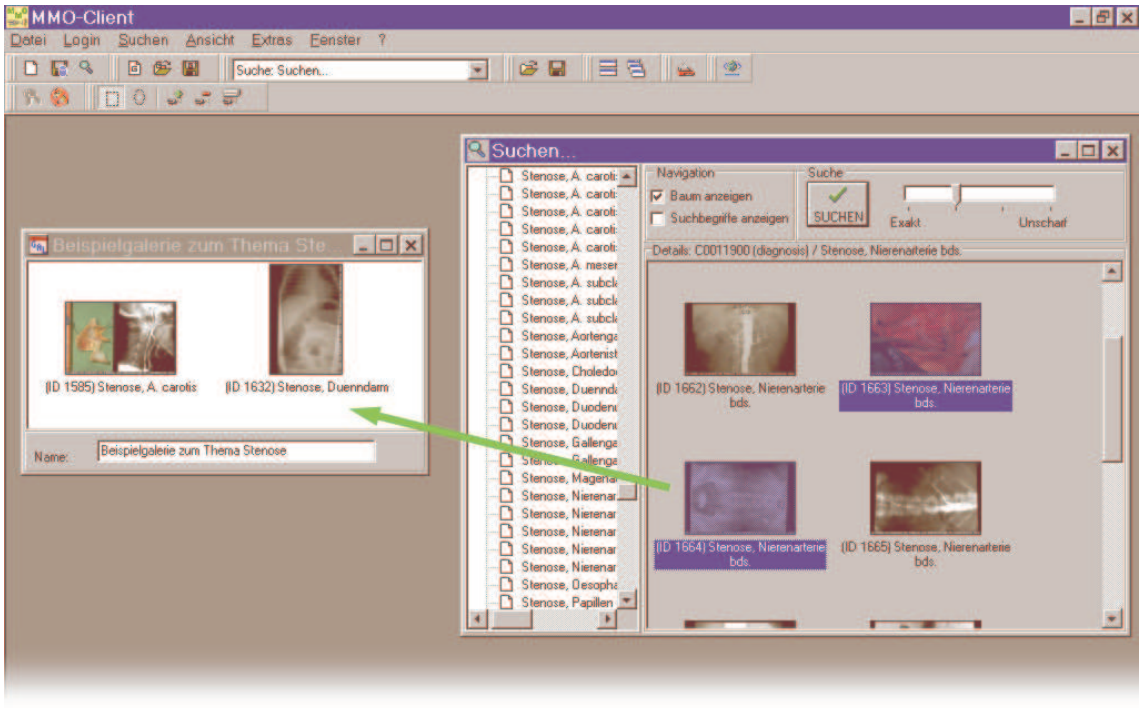


Abb. 8: Gefundene MOBs in Galerien sichern

Dieser Vorgang kann beliebig wiederholt werden, so daß bereits bestehende Galerien erweitert werden können. Dabei kontrolliert das System automatisch, daß bereits vorhandene MOBs nicht doppelt aufgenommen werden. Innerhalb der Galerie läßt sich die Reihenfolge der gespeicherten MOBs anpassen, indem einzelne Objekte per Drag-and-Drop an eine neue Position verschoben werden. Dabei ist das zu verschiebende Objekt über dem MOB abzulegen, hinter das es verschoben werden soll.

Per Doppelklick lassen sich einzelne MOBs öffnen. Weitere Funktionen, wie z.B. das Markieren oder Öffnen aller in der Galerie erfaßten Knoten, verbergen sich hinter einem Kontext-Menü, das sich mit der rechten Maustaste öffnen läßt. Hier findet der Anwender insbesondere eine Funktion zum Entfernen der ausgewählten MOBs aus der Galerie.

Jede Galerie kann als „*.gal“-Datei auf der Festplatte des Anwenders gespeichert werden. Innerhalb dieser Datei werden die Informationen wie in Abschnitt 3.3.2 beschrieben im XML-Format abgelegt. Des weiteren ist es möglich, eine Galerie in der Datenbank zu speichern, indem sie wie eine Multimediadatei in einem MOB erfaßt wird. An der Stelle, an der üblicherweise die Bilder, Video-Clips etc. im MOB erscheinen, wird dann ein Button angezeigt, über den die Galerie per Mausklick geöffnet werden kann.

4.5. Verknüpfung von MOBs mit Hilfe des MMO-Clients

Die Verknüpfung von MOBs (vgl. Abschnitt 3.2.8) kann auf komfortable Weise per Drag-and-Drop vorgenommen werden. Dabei müssen stets die Ziel-MOBs mit der Maus festgehalten und über dem Quell-Mob „fallengelassen“ werden. Hierfür ist es notwendig, das Quell-MOB zur Bearbeitung zu öffnen. Des weiteren muß durch die Aktivierung eines Kontrollkästchens im Hauptmenüpunkt „Ansicht“ veranlaßt werden, daß in den MOB-Fenstern ein „Relationsanker“ angezeigt wird. Ein Relationsanker ist ein Querbalken, der im oberen Bereich der MOB-Fenster dargestellt wird und als Start- und Landeplatz für die mit der Erstellung von Verknüpfungen verbundenen Drag-and-Drop-Aktionen dienen kann (siehe Abb. 9 und Abb. 10).

Wie in Abschnitt 3.2.8 erläutert, können sich Verknüpfungen einerseits auf das gesamte MOB, oder auf einzelne Hervorhebungen beziehen. Dabei beinhaltet eine annotationsbezogene Verknüpfung automatisch auch eine MOB-bezogene Verknüpfung.

4.5.1. Bezug zwischen Quell- und Ziel-MOB herstellen

Um eine Verknüpfung zwischen den MOBs herzustellen, ist es nötig, die Ziel-MOBs per Drag-and-Drop auf dem Relationsanker des Quell-MOBs abzulegen. Dabei gibt es zwei Möglichkeiten, die Ziel-MOBs mit der Maus zu erfassen.

Wenn neben dem Quell-MOB auch das Ziel-MOB zur Bearbeitung/Ansicht geöffnet wurde, so ist es möglich, dieses über seinen Relationsanker mit der Maus zu greifen. Dabei müssen natürlich beide MOBs den in Abschnitt 3.2.8.1 genannten Bedingungen genügen. Diese kann der Benutzer anhand der Beschriftung der Relationsanker kontrollieren, die stets einem der folgenden drei Punkte entspricht:

- *„Nur Drag“*
  Der Benutzer besitzt keine Schreibrechte, aber das MOB wurde bereits in der Datenbank gespeichert. Deshalb darf auf dieses Objekt nur verwiesen werden.
- *„Nur Drop“*
  Der Benutzer hat Schreibrechte, aber das MOB hat (noch) keine Identifikationsnummer in der Datenbank. Deshalb darf nur von diesem Objekt auf andere verwiesen werden.
- *„Drag and Drop“*
  Ansonsten. Der Benutzer hat Schreibrechte und das MOB hat eine Identifikationsnummer in der Datenbank.

Die zweite Möglichkeit zur Ergreifung der Ziel-MOBs liegt in der Verwendung der Miniaturansichten in Galerien und Suchergebnissen. Da die MOBs, die dort angezeigt werden, bereits in der Datenbank erfaßt und für den Benutzer lesbar sind, erfüllen sie immer die Voraussetzungen für Ziel-MOBs.

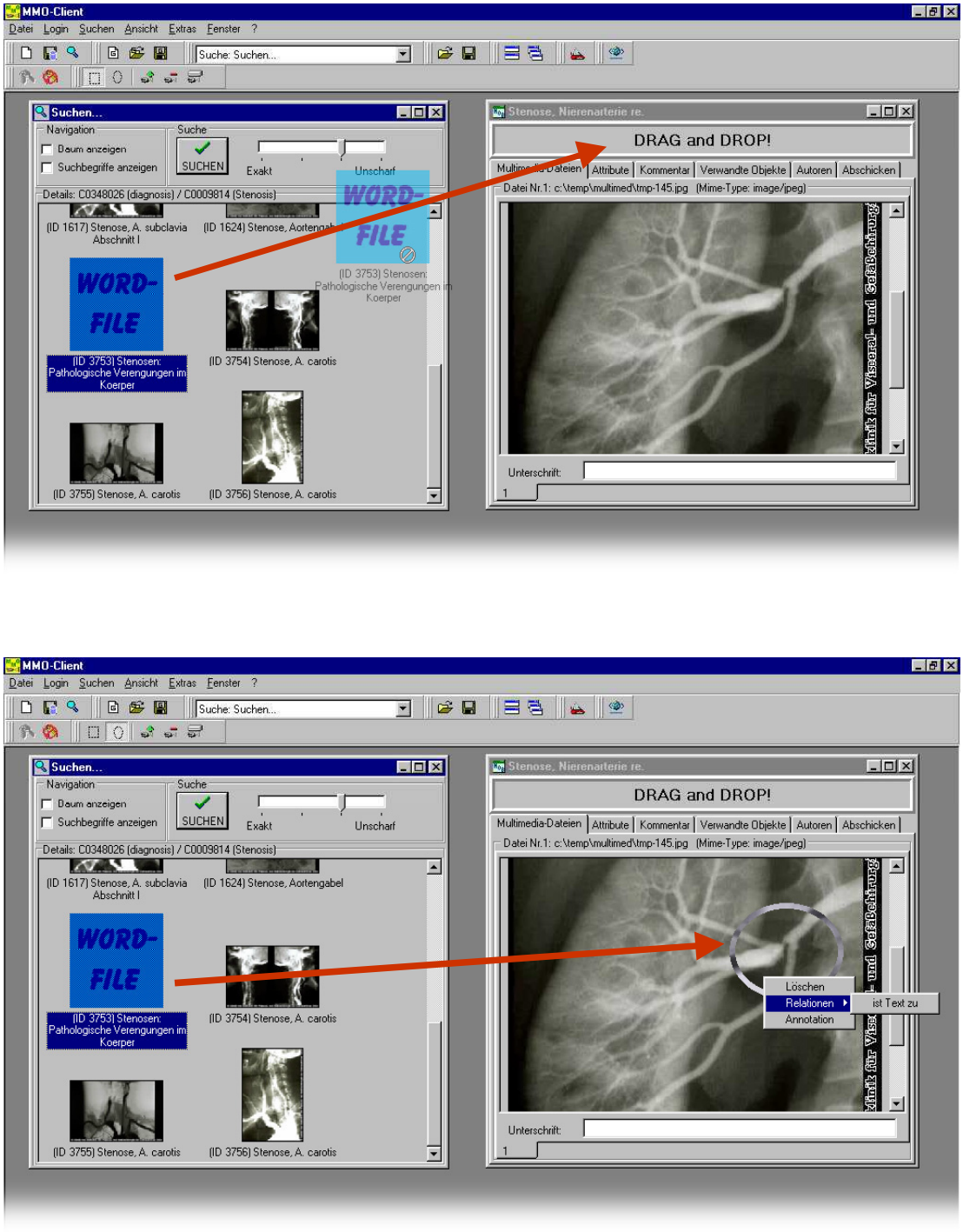


Abb. 9: Verknüpfung von MOBs (Teil 1)
Ziel-MOBs werden per Drag-and-Drop auf dem Relationsanker (oben) bzw. auf einer Annotation (unten) des Quell-MOBs angelegt.

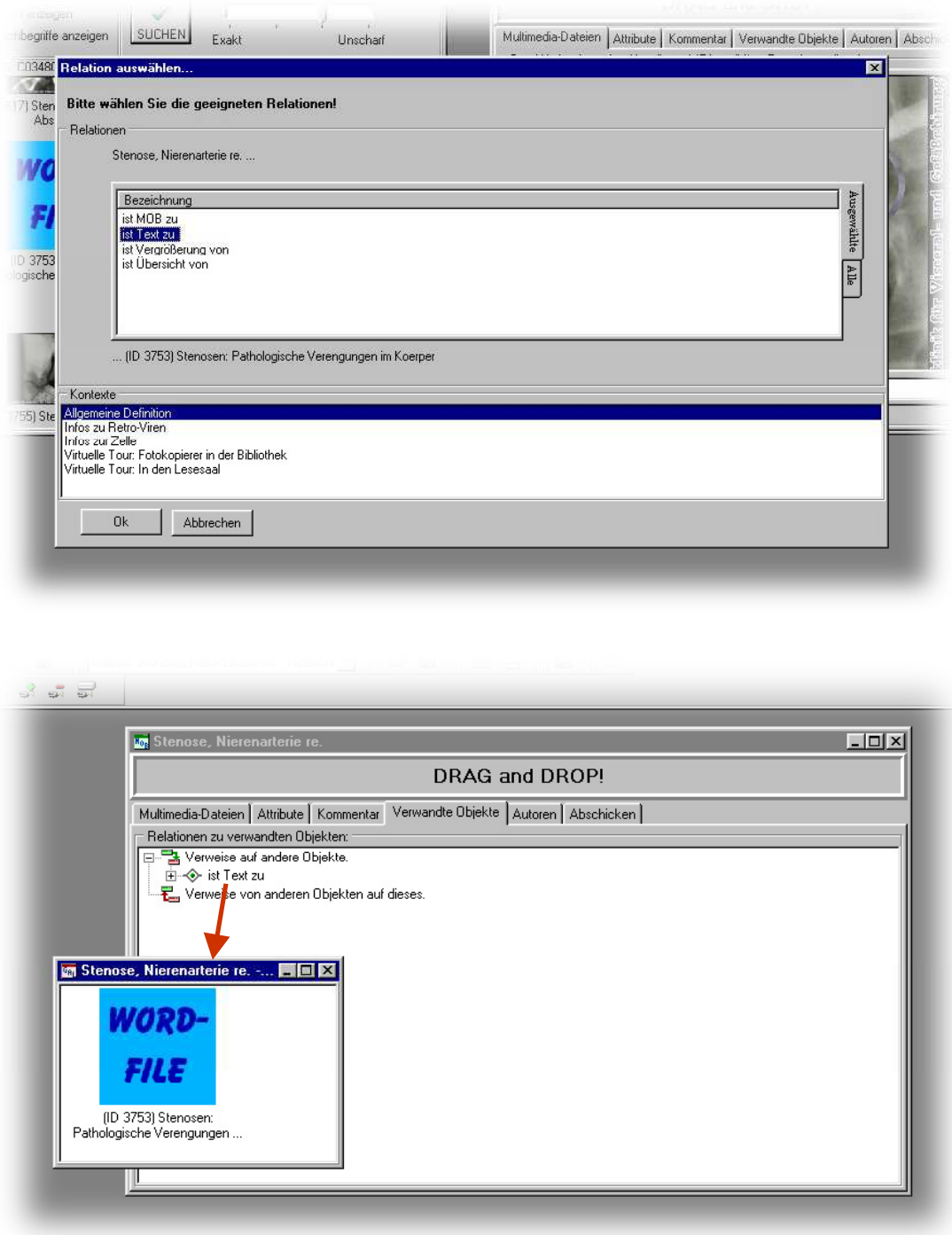


Abb. 10: Verknüpfung von MOBs (Teil 2)
Oben: Im Dialogfeld sind die gewünschten Relationen und Kontexte auszuwählen.
Unten: Im Reiter „Verwandte Objekte“ werden alle mit dem MOB assoziierten Relationen aufgeführt. Per Doppelklick bekommt man die zugehörigen Objekte.

Aus diesem Grund wurden Mechanismen in den MMO-Client eingebunden, die es ermöglichen, eine oder mehrere der Miniaturabbildungen mit der Maus zu markieren und diese per Drag-and-Drop auf dem Relationsanker des Quell-MOBs zu plazieren (Abb. 9, oben). Auf diese Weise wird das Quell-MOB mit allen ausgewählten Ziel-MOBs in Verbindung gesetzt.

Wie bereits erwähnt, gibt es neben den MOB-bezogenen Relationen auch solche, die sich auf Annotationen beziehen. Die Vorgehensweise zur Erstellung derartiger Verknüpfungen ist sehr ähnlich zu den oben beschriebenen „MOB-zu-MOB“-Verknüpfungen. Der einzige Unterschied besteht darin, daß die mit der Maus aufgegriffenen Ziel-MOBs nicht auf dem Relationsanker des Quell-MOBs abgelegt werden, sondern auf der gewünschten Hervorhebung im Bild (Abb. 9, unten).

4.5.2. Auswahl von Relation und optionalen Kontexten

Nachdem der Bezug zwischen Quell- und Ziel-MOB hergestellt wurde, öffnet sich ein Dialogfenster, in dem die relevanten Relationen (z.B. „ist Vergößerung von...“, „ist textuelle Beschreibung von...“ etc.; oberer Fensterteil) sowie die entsprechenden Kontexte (unterer Fensterteil) ausgewählt werden können (Abb. 10, oben).

Die Liste aller verfügbaren Relationen wird beim Start des MMO-Clients initial aus dem Backend in den Client importiert. Da diese Liste sehr lang sein kann, und ein Benutzer seine Verknüpfungen vermutlich immer nur aus einer ausgewählten Teilmenge aller zur Verfügung stehenden Relationen wählen möchte, kann der Anwender im o.a. Dialogfenster zwischen zwei Darstellungsmodi hin und her schalten.
Im ersten Modus werden alle im Backend erfaßten Relationen aufgelistet. Dabei ist jeder Listeneintrag mit einem Kontrollkästchen versehen, welches für den zweiten Darstellungsmodus relevant ist (Karteireiter „Alle“ in Abb. 10, oben).

Im zweiten Darstellungsmodus werden nur die Relationen angezeigt, deren Kontrollkästchen im ersten Darstellungsmodus aktiviert wurden. Insbesondere werden diese Kontrollkästchen hier nicht abgebildet, da sie für diese Darstellungsform keine zusätzlichen Informationen bergen (Karteireiter „Ausgewählte“ in Abb. 10, oben).
Durch diese beiden Darstellungsformen kann jeder Benutzer die für ihn wichtigen Relationen in einer Art Bibliothek verwalten. Dabei kann er den ersten Darstellungsmodus verwenden, um die gewünschten Relationen über die o.a. Kontrollkästchen in die Bibliothek einzufügen oder sie aus dieser zu entfernen. Die Ergebnisse seiner Selektion werden dann im zweiten Darstellungsmodus abgebildet. Diese Einstellungen werden nach Beendigung des Programms automatisch gespeichert, so daß die entsprechenden Relationen auch beim nächsten Start des MMO-Clients als präferiert gekennzeichnet sind.
Unabhängig vom gewählten Darstellungsmodus können in dieser Auswahlliste die Relationen ausgewählt werden, die den verwandtschaftlichen Zusammenhang zwischen den aktuellen Quell- und Ziel-MOBs beschreiben.

In der unteren Auswahlliste in Abb. 10 (oben) können optional die Kontexte ausgewählt werden, durch die die verwandtschaftliche Beziehung der MOBs genauer spezifiziert wird. Wie in Abschnitt 3.2.8.2 erörtert, können diese Kontexte in einer vom Benutzer vorgegebenen Textdatei verwaltet werden. Dabei entspricht jede Zeile dieser Datei, die beim Öffnen des Dialogfensters zeilenweise in die Auswahlliste eingefügt wird, genau einem Kontext.

4.5.3. Darstellung der verknüpften Objekte im MOB-Fenster

Für die Darstellung der Verknüpfungen im Quell-MOB wurden die MOB-Fensters mit einem eigens dafür vorgesehenen virtuellen Karteireiter ausgestattet. Dieser Bereich enthält eine Komponente, die es erlaubt, Symbole und Textelemente in einer hierarchischen Baumstruktur zu erfassen. Einzelne Pfade dieses Baums können über kleine „+“- und „-“-Symbole, die sich am Anfang jedes Eintrags befinden, auf- und zugeklappt werden (vgl. Abb. 10, unten und Abb. 11). Innerhalb dieser Baumkomponente befinden sich auf der

obersten Ebene zwei Pfade mit den Beschriftungen *„Verweise auf andere Objekte“* und *„Verweise von anderen Objekten auf dieses“*.

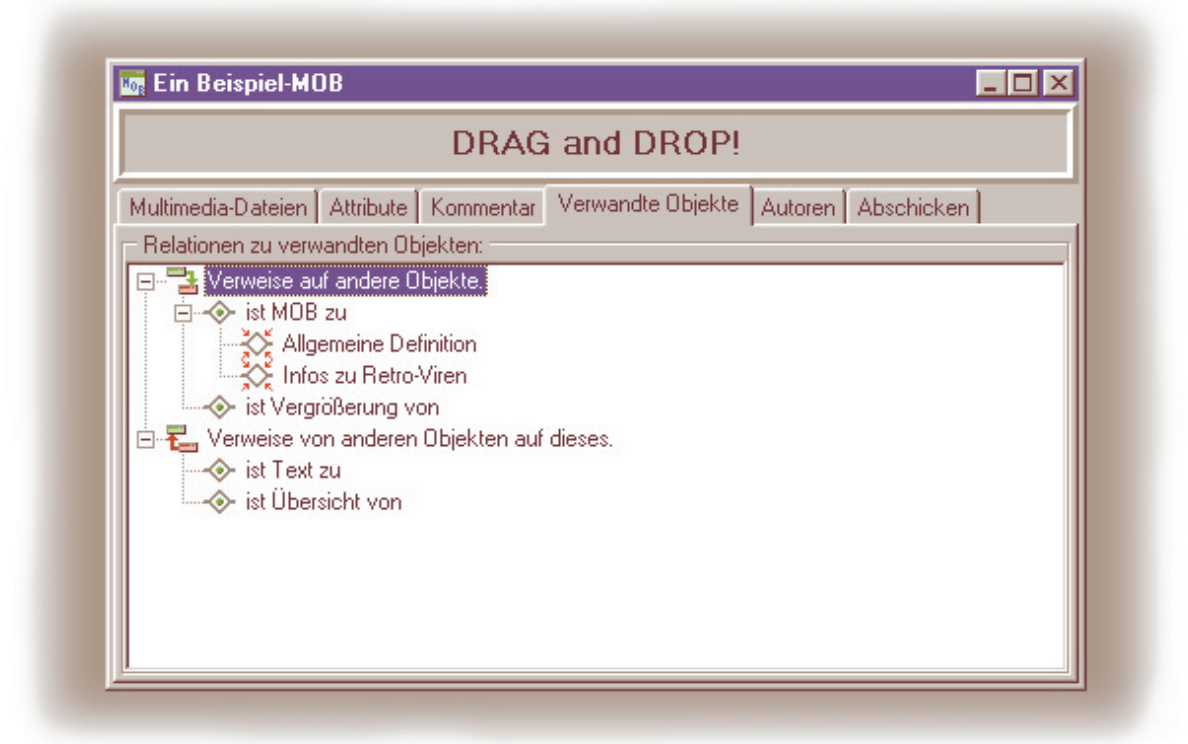


Abb. 11: Verwaltung verwandter MOBs in einem eigenen virtuellen Karteireiter

Wird nun eine der im vorherigen Abschnitt beschriebenen Verknüpfungen durchgeführt, so werden die damit verbundenen Information unterhalb des Knotens *„Verweise auf andere Objekte“* im Quell-MOB abgelegt. Dabei werden unterhalb dieses Knotens die Bezeichnungen der Relationen aufgelistet, hinter denen sich mindestens ein zugeordnetes MOB verbirgt. Über einen Doppelklick auf eine der Relationen öffnet sich ein abgewandeltes Galeriefenster, in dem alle MOBs (unabhängig von etwaigen Kontexten) aufgeführt werden, auf die bzgl. der ausgewählten Relation verwiesen wurde (vgl. Abb. 10, unten). Sofern diese Relation in Verbindung mit Kontexten vergeben wurde, werden innerhalb des Baums zusätzlich alle verwendeten Kontexte unterhalb der jeweiligen Relation angezeigt. Über einen Doppelklick auf einen Kontext wird die Anzeige der zugehörigen MOBs entsprechend eingeschränkt.

Im Pfad *„Verweise von anderen Objekten auf dieses“* werden die Objekte dargestellt, die auf das aktuelle MOB verweisen. Dabei werden auch hier die verwendeten Relationen der Reihe nach aufgelistet. Allerdings werden in diesem Fall die verwendeten Kontexte nicht berücksichtigt, da diese als Freitext angegeben werden können. Die damit verbundene Flexibilität könnte zu einer unübersichtlichen Baumstruktur führen.
Da die Verknüpfungsinformationen immer nur in den Quell-MOBs gespeichert werden, werden die Inhalte des Pfades *„Verweise von anderen Objekten auf dieses“* weder beim Speichern im lokalen Dateisystem, noch beim Speichern in der Datenbank in den XML-Code integriert. Lediglich beim Öffnen des MOBs aus der Datenbank werden diese Informationen automatisch vom Backend in den XML-Code eingebaut.

Die Verknüpfungen, die sich speziell auf bestimmte Bild-Annotation beziehen, können über das Kontextmenü der entsprechenden Hervorhebung analog zum Pfad *„Verweise auf andere Objekte“* angezeigt werden. Dabei wird die Baumstruktur durch verschachtelte Menüeinträge abgebildet (vgl. Abb. 9, unten).

4.6. Export von virtuellen Touren nach HTML

4.6.1. Spezieller Darstellungsmodus von MOB-Fenstern

Wie in Abschnitt 3.2.9 erklärt, können die verschiedenen virtuellen Rundgänge durch eine geschickte Interpretation von Relationen und Kontexten mit dem MMO-Client konstruiert werden. Dabei werden drei Arten von Knotentypen unterschieden: Positions-, Text- und Lageplanknoten. Bzgl. der Positionsknoten kann außerdem unterschieden werden, ob ein normales JPG-Bild, ein Panorama-Bild oder eine Flash-Animation dargestellt werden soll. Sofern ein MOB in einen virtuellen Rundgang integriert werden soll, muß zunächst der Knotentyp festgelegt werden. Des weiteren können für jeden Knotentyp verschiedene Parameter angegeben werden, die für den Export wichtig sind. Die Festlegung des Typs, sowie die Werte der entsprechenden Parameter

werden über spezielle, vorgegebene Attribute angegeben, die nicht über das UMLS, sondern als Freitext definiert werden.

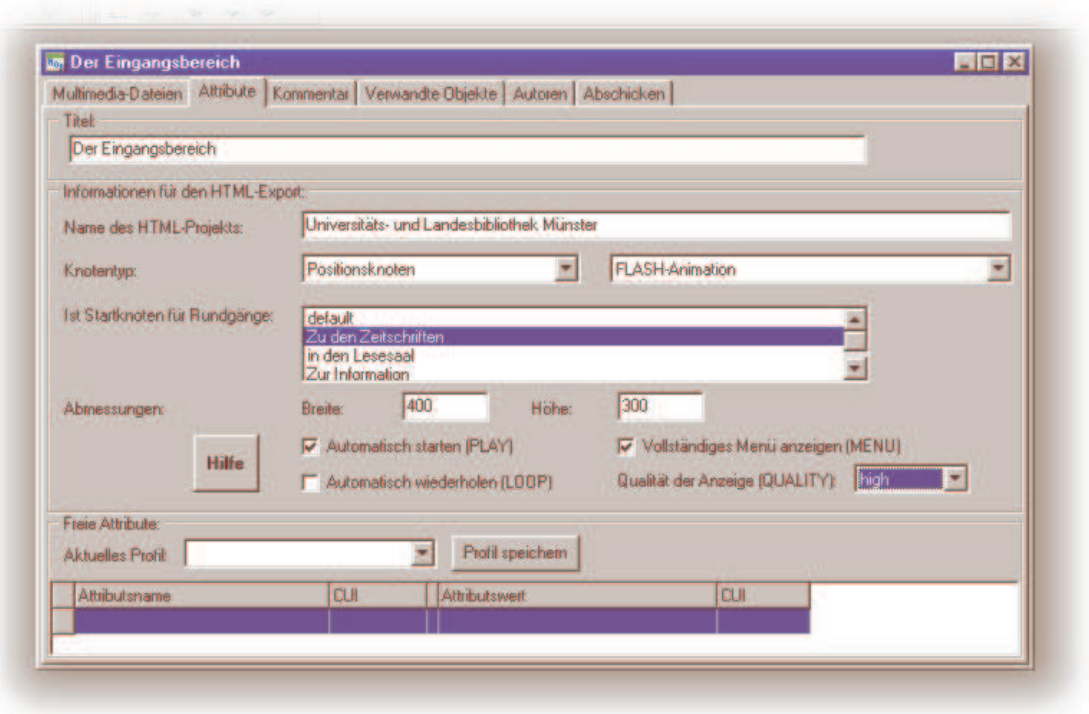


Abb. 12: Im HTML-Export-Modus werden die für die Rundgänge relevanten Attribute in einem separaten Bereich verwaltet.

Um dem Anwender die Erstellung virtueller Touren zu erleichtern, können die MOB-Fenster in einen speziellen Darstellungsmodus für den HTML-Export umgeschaltet werden. Dieser Modus sorgt dafür, daß die Attribute, die speziell für den HTML-Export benötigt werden, nicht mehr in der Attributstabelle, sondern in einem eigenen Bereich verwaltet werden (vgl. Abb. 12). Je nach ausgewähltem Knotentyp werden die jeweils relevanten Eingabefelder dynamisch ein- und ausgeblendet („Skip logic“, vgl. [4]).

Eine besondere Rolle für den Export spielen die Eingabefelder „Name des HTML-Projekts“ und „Ist Startknoten für Rundgänge“ (nur bei Positionsknoten). Wie bereits erwähnt, kann jeder Anwender beliebig viele virtuelle Rundgänge erstellen, die jeweils einen eigenen Titel (= Kontext der Verknüpfungen) besitzen. Um eine sinnvolle Navigation in den HTML-Seiten zu ermöglichen, ist es beim Export der Seiten nötig, zunächst eine Übersicht über alle Touren des

Benutzers zu erstellen und über Hyperlinks den Einstig in diese Touren zu ermöglichen. Aus diesem Grund muß für jede virtuelle Tour genau ein zugehöriger Positionsknoten über das Feld „Ist Startknoten für Rundgänge“ als Einstiegsknoten der Tour definiert werden.
Probleme ergeben sich jedoch, wenn verschiedene Benutzer die gleichen Bezeichnungen für ihre Touren verwenden. In diesem Fall könnte es beim Export der Seiten zu Verwechslungen kommen, da es in der Datenbank mehrere Startknoten zu einem Tournamen gäbe. Um dieses Problem zu vermeiden, ist eine inhaltliche Abgrenzung der MOBs notwendig. Diese kann durch das Eingabefeld „Name des HTML-Projekts“ vorgenommen werden. Der Projektname dient als Kennzeichner für die MOBs, die in den Rundgängen eines Anwenders oder einer Institution verwendet werden. Darüber hinaus kann er als Zusatzinformation in die zu exportierenden HTML-Seiten integriert werden. Aus diesem Grund sollte ein möglichst eindeutiger, aussagekräftiger Begriff gewählt werden, wie z.B. der Name des Autors oder der Institution.
Durch die Definition von HTML-Projekten ergibt sich eine weitere Hierarchieebene zur Organisation von virtuellen Rundgängen, die zur folgenden Gesamtstruktur führt:

- Für einen bestimmten Themenbereich kann ein HTML-Projekt definiert werden
- Innerhalb eines Projekts können mehrere virtuelle Touren erfaßt werden
- Innerhalb der Touren werden, wie bereits bekannt, verschiedene verknüpfte Positionen nebst Text erfaßt

4.6.2. Das Exportfenster

Um die virtuellen Rundgänge im Internet präsentieren zu können, müssen diese nach ihrer Konstruktion in HTML-Seiten umgewandelt werden. Hierfür steht dem Anwender ein Dialogfenster zur Verfügung, in dem die verschiedenen globalen Eckdaten für den Seitenexport angegeben werden können (Abb. 13).

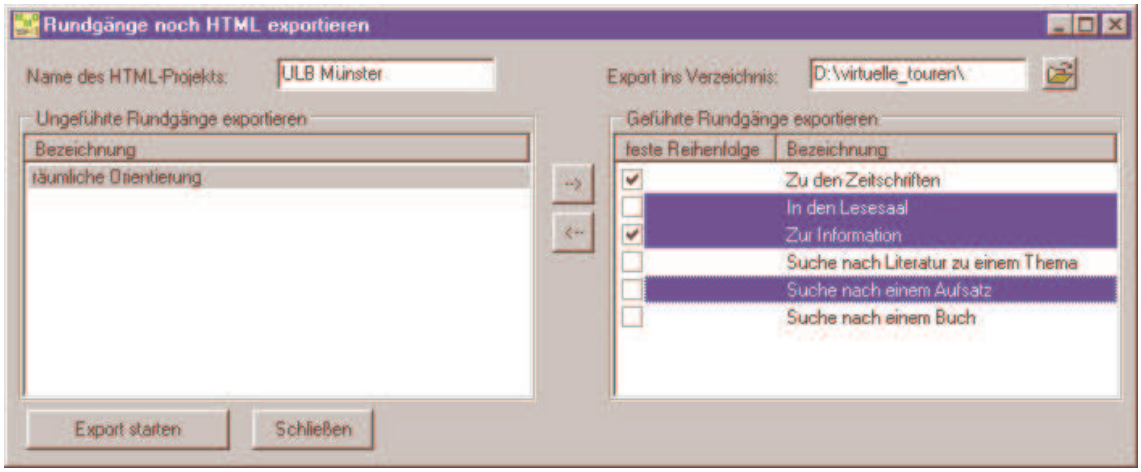


Abb. 13: In einem Dialogfenster können die Eckdaten für den HTML-Export der Rundgänge angegeben werden.

Dabei werden die Voreinstellungen dieses Fensters in den allgemeinen Programmoptionen des MMO-Clients verwaltet.

Im oberen Fensterbereich müssen der Name des zu exportierenden HTML-Projekts und das Verzeichnis, in dem die HTML-Dateien angelegt werden sollen, angegeben werden.
Des weiteren finden sich in dem Fenster zwei Auswahllisten, in denen die verfügbaren Kontexte aufgeführt werden. Wie in Abschnitt 3.2.9.2 erwähnt, sind einige dieser Kontexte als Titel der virtuellen Rundgänge zu interpretieren. Da diese Eigenschaft jedoch nicht explizit in den Systemeinstellungen abgespeichert wird, werden hier zunächst alle verfügbaren Kontexte angegeben, aus denen der Anwender die zu exportierenden Rundgänge auswählen kann.
Die Inhalte dieser Listen können über entsprechende Buttons hin und her verschoben werden. Auf diese Weise kann festgelegt werden, welche der Kontexte als ungeführte Touren (linke Liste) und welche als geführte Touren (rechte Liste) zu interpretieren sind. Bei Kontexten, die nicht als Rundgänge interpretiert werden sollen, spielt es keine Rolle, in welcher Liste sie geführt werden, da sie für den Export nicht beachtet werden.
Des weiteren kann in der Liste der geführten Touren über das Kontrollkästchen „feste Reihenfolge“ festgelegt werden, welche der Rundgänge als streng

geführt (Kästchen aktiviert) und welche als frei geführt (Kästchen deaktiviert) betrachtet werden sollen. Dies hat folgenden Hintergrund: Die Frage, ob eine geführte Tour frei oder streng geführt werden soll, wird nicht bei der Konstruktion, sondern erst beim Export der Rundgänge entschieden. Dies ist möglich, da sowohl freie als auch streng geführte Touren grundsätzlich die gleiche Struktur haben, nämlich die einer linearen Liste von Positionsknoten. Durch die Festlegung zum Exportzeitpunkt bleibt es dem Anwender vorbehalten, freie geführte Rundgänge auch kurzfristig als streng geführte Touren zu interpretieren (und umgekehrt), ohne Änderungen an der Konstruktion vornehmen zu müssen.

Bevor der HTML-Export gestartet werden kann, müssen zunächst die zu exportierenden Touren in den Auswahllisten markiert werden. Unfertige Touren und Kontexte, die keine Rundgänge repräsentieren, können vom Export ausgeschlossen werden, indem sie nicht markiert werden. Anschließend werden die resultieren HTML-Seiten durch Betätigung des Buttons „Export starten“ in das vom Anwender gewählte Verzeichnis geschrieben. Das Layout der Seiten kann durch Template-Dateien, deren Positionen in den allgemeinen Programmoptionen angegeben werden können, festgelegt werden. Diese Templates bestehen im wesentlichen aus HTML-Code, der an verschiedenen Stellen mit Platzhaltern versehen werden kann. Beim Export werden diese Platzhalter durch entsprechende Werte (z.B. Titel des Rundgangs, Name des HTML-Projekts, ...) ersetzt.
Des weiteren gibt es Bereiche in den Seiten, die nicht eins zu eins durch einen festen Wert ersetzt werden können, wie z.B. der Navigationsbereich mit den Verweisen zu anderen Positionen. Um auch diese Inhalte abbilden zu können, ist es möglich, kleine XML-Segmente in die Templates einzubauen. Diese Segmente können vom MMO-Client interpretiert und zum Einfügen komplexerer Seitenelemente verwendet werden.

4.6.3. Ein Anwendungsbeispiel: Das Projekt LOTSE

Die Funktionalität zur Erstellung virtueller Touren wurde im Rahmen des BMBF-Projekts LOTSE an der Universitäts- und Landesbibliothek Münster verwendet und evaluiert. Anhand dieses Projekts wird diese Funktionalität im folgenden näher illustriert.

4.6.3.1. Hintergrund

Das Akronym LOTSE steht für „Library online tour and selfpaced education“ und beschreibt ein webbasiertes Navigations- und Schulungssystem für elektronische und konventionelle Informationsmittel. Dieses soll Fachwissenschaftler und Studenten dabei unterstützen, fachliche Informationsressourcen zu finden, zu benutzen und zu bewerten. Derzeit werden exemplarisch die Inhalte für die Fächer Pädagogik und Medizin umgesetzt.

Für die Aufbereitung der Inhalte zur Darstellung im Internet, stehen dem LOTSE-Team zwei autarke Systeme zur Verfügung. Zum einen werden verschiedene Inhalte über ein XML-basiertes Content Management System veröffentlicht.

Zum anderen werden virtuelle Bibliothekstouren angeboten, die mit dem MMO-Client erstellt werden. In diesem Zusammenhang wurden bereits frühzeitig Entwicklungsversionen des MMO-Clients vom LOTSE-Team verwendet, um experimentell erste Test-Rundgänge zu erzeugen. Die dabei gesammelten Erfahrungen wurden in persönlichen Gesprächen erörtert und in die weitere Entwicklung des MMO-Clients integriert. Hierdurch wurden besonders die Gestaltung des HTML-Export-Modus der MOB-Fenster (vgl. Abschnitt 4.6.1) sowie die Anforderungen an den Integrationsgrad von Lageplanknoten in die Rundgänge beeinflußt.

4.6.3.2. Aufbau der exportierten Seiten

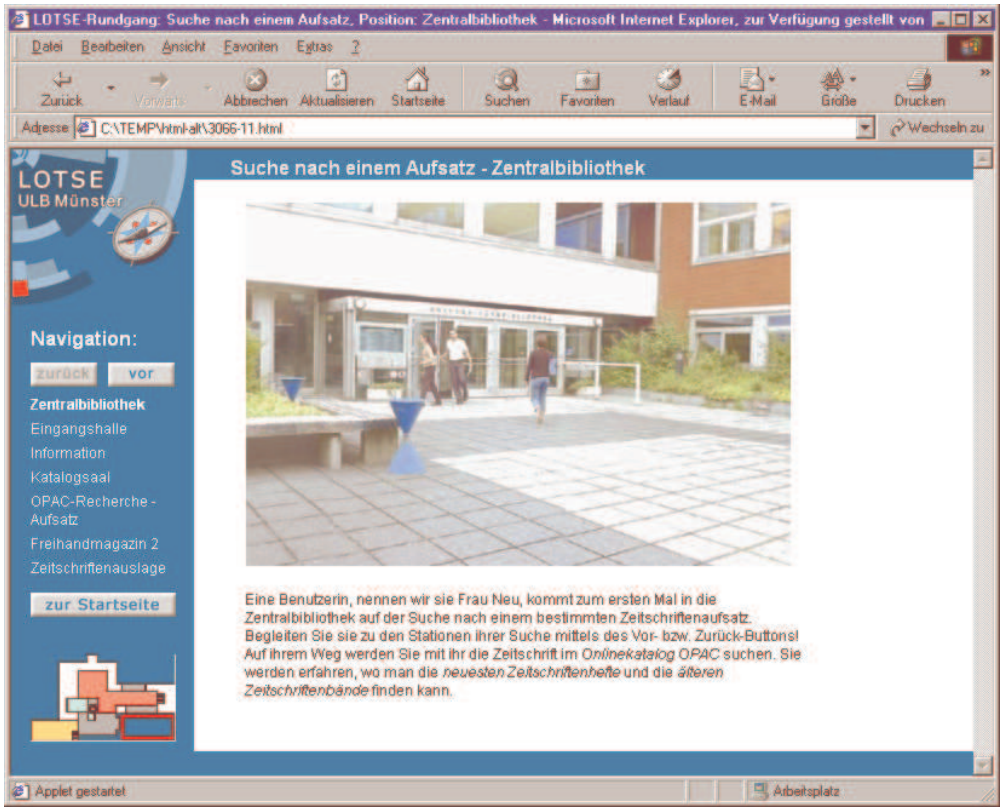
Mit Hilfe der beiden in Abb. 14 dargestellten Webseiten, die aus den o.a. LOTSE-Rundgängen stammen, kann der Aufbau der nach HTML exportierten Positionsknoten erläutert werden.

Das Kernstück dieser Seiten bildet die Abbildung bzw. Flash-Animation der Position sowie der zugehörige beschreibende Text. In Abb. 14 (a) wurde beispielsweise ein 360° Panoramabild eingefügt, welches den Bereich vor dem Bibliothekseingang zeigt. Mit der Maus kann der Besucher den Blickwinkel ändern und sich so einen Überblick über die Umgebung verschaffen. Der beschreibende Text entspricht dem Text-MOB, der dem Positions-MOB bzgl. des aktuellen Rundgangs (Kontext) zugewiesen wurde.

Dieser Seitenkern wird in das eigentliche Seitenlayout eingebettet. Dabei wird im oberen Bereich der Name des Rundgangs sowie der Titel des aktuellen Positions-MOBs angezeigt. Auf der linken Seite befindet sich der Navigationsbereich, über den die weiteren zum Rundgang gehörenden Positionen aufgerufen werden können. Je nach Rundgangstyp kann sich die Struktur dieses Bereichs ändern. Die in Abb. 14 abgebildeten Stationen gehören beispielsweise beide zu je einem geführten Rundgang, wobei es sich oben (a) um eine streng geführte und unten (b) um eine freie geführte Tour handelt. Dies kann man am Aufbau der Navigationsleiste erkennen. Im Abb. 14 (a) besteht dieser Bereich aus einfachem Text, wobei die aktuelle Position im Fettdruck dargestellt wird. Die Navigation wird über Grafiken am Listenanfang mit der Beschriftung „vor“ und „zurück“ realisiert. Diese sind mit entsprechenden Hyperlinks versehen. In Teil (b) der Abbildung besteht die Liste der Positionsknoten aus einzelnen Hyperlinks, so daß die verschiedenen Positionen unmittelbar aufgerufen werden können. Daher sind die in den strengen Touren verwendeten „vor“- und „zurück“-Links überflüssig.

Wie in Abschnitt 3.2.9.2 erwähnt, kann jedem virtuellen Rundgang auch ein Lageplan zugewiesen werden. Dieser kann aus mehreren Stockwerken bestehen, die im Lageplan-MOB durch jeweils eine JPG-Datei repräsentiert werden. Sofern ein Positionsknoten mit einer Hervorhebung innerhalb eines zum Rundgang gehörenden Lageplan-MOBs verknüpft wurde, wird unterhalb

des Navigationsbereichs eine Miniaturansicht des zugehörigen Stockwerks angezeigt. Per Mausklick kann hierüber ein Fenster mit einer Detailansicht des Lageplans (inklusive aller Stockwerke) geöffnet werden. Dabei wird die aktuelle Position im Rundgang durch eine farbige Hervorhebung in der entsprechenden Lageplangrafik angezeigt. Sofern es sich um eine freie geführte oder eine ungeführte Tour handelt, kann der Besucher durch Anklicken des Lageplans direkt an die jeweils zugehörige Position des Rundgangs springen.



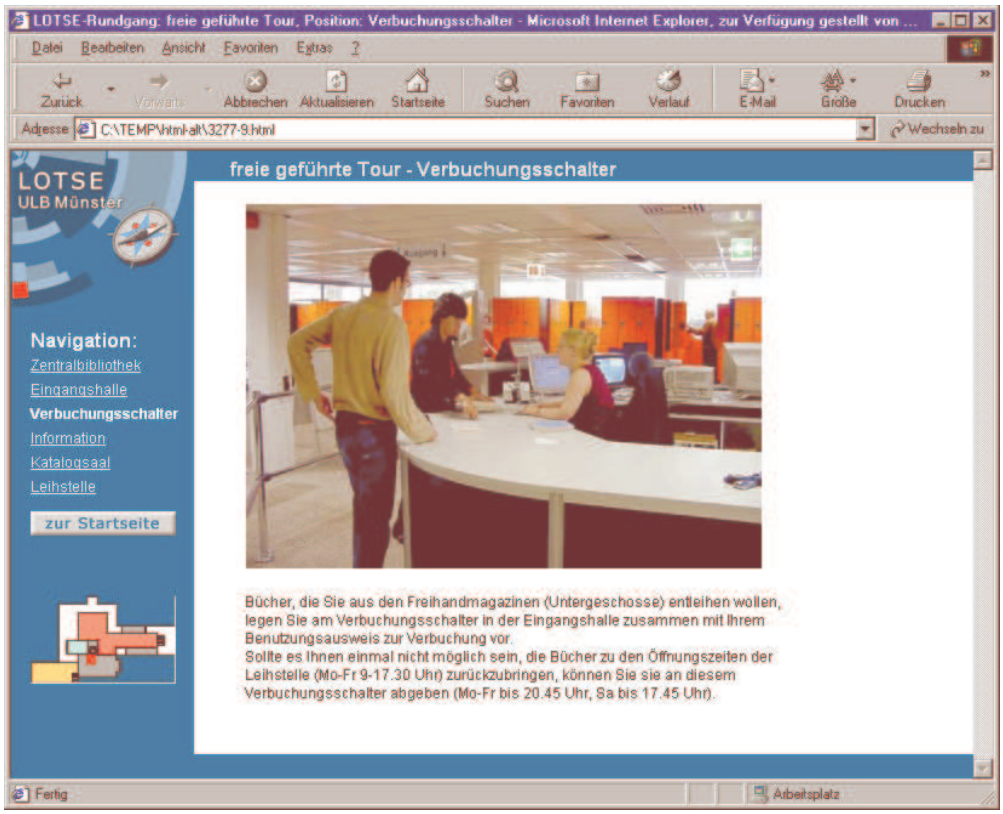


Abb. 14: Virtuelle Touren im Rahmen des Projekts LOTSE
Oben (a): Position „Zentralbibliothek“ im streng geführten Rundgang „Aufsatz“
Unten (b): Position „Verbrauchsschalter“ im Rundgang „freie geführte Tour“

5. Diskussion

In der vorliegenden Arbeit werden der Entwurf und die Implementierung einer Benutzeroberfläche (MMO-Client) zur komfortablen und effizienten Verwaltung einer Datenbank für multimediale, medizinische Objekte (MMO-Datenbank) beschrieben. Dabei wurden die Funktionen dieses Programms nicht auf die elementaren Datenbankmechanismen (Suchen, Speichern, ...) beschränkt, sondern um darüber hinausgehende Funktionalitäten (offline-Verwaltung im Dateisystem, HTML-Export, ...) erweitert.

5.1. Umsetzung des MMO-Clients als eigenständige Windows-Applikation

Für die Entwicklung moderner Benutzeroberflächen stehen zwei alternative Umsetzungsstrategien zur Verfügung. Zum einen ist es möglich, Applikationen zu entwickeln, die auf der Systemebene des Client-Rechners ausgeführt werden. Diese Anwendungen werden meistens für ein bestimmtes Betriebssystem kompiliert und können nur auf solchen Systemen ausgeführt werden. Zum anderen werden oft webbasierte Lösungsansätze verfolgt, deren Inhalte meistens vollständig im HTML-Format verarbeitet werden [28].

Webbasierte Systeme haben den Vorteil, daß sie plattformübergreifend in heterogenen Netzwerkumgebungen verwendet werden können, da lediglich ein Webbrowser als Benutzerclient benötigt wird [1]. Da ein solcher Browser heutzutage in allen gängigen Betriebssystemen bereits standardmäßig integriert ist, kann auch der Aufwand für die Installation der Benutzeroberflächen minimiert werden. Daher sieht Katz „WWW-Browser (als) exzellente Kandidaten für Benutzerschnittstellen für universelle Multimedia-Betrachtungsstationen“ [37].

Allerdings reichen die Funktionen einer „Betrachtungsstation“ nicht aus, um den Anforderungen an einen komfortablen Benutzerclient zur Verwaltung von multimedialen Objekten gerecht zu werden, so daß weiterführende Funktionalitäten notwendig sind. Aufgrund technischer Einschränkungen der

Webbrowser-Technologie [1] lassen sich diese Funktionen nicht auf HTML-Basis realisieren, da die einzigen Interaktionsmöglichkeiten, die in HTML zur Verfügung stehen, aus Hyperlinks und Formularen bestehen.
Daher sind in den letzten Jahren Entwicklungswerkzeuge und Programmiersprachen wie Active X, Flash und Java entstanden, die es ermöglichen, interaktive Bausteine in HTML-Seiten einzubauen. Um diese Bausteine nutzen zu können, ist es unter Umständen notwendig, entsprechende Plugins (kleine Zusatzprogramme) in den Browser zu integrieren, was mit zusätzlichem administrativen Aufwand verbunden ist.
Diese neuen Technologien bieten zwar vielversprechendes Verbesserungspotential [12], allerdings ist fraglich, ob dieses Potential ausreicht, um den Zielsetzungen dieser Arbeit gerecht zu werden. Insbesondere Java-Applets haben den Nachteil, daß sie in ihrem Funktionsumfang stark eingeschränkt und zuweilen fünf bis 50 mal langsamer sind, als Applikationen, die speziell für ein Betriebssystem kompiliert wurden [1]. Insgesamt scheinen „die derzeit verwendeten Techniken ... unzureichend bezüglich Zeit und Komfort“ zu sein, so daß „noch immer Arbeit erledigt werden muß, um die grafischen Benutzeroberflächen für die Navigation in WWW-basierten Bild- und Multimediadatenbanken zu optimieren“ [28].

Um die Probleme webbasierter Anwendungen vermeiden zu können, wurde der MMO-Client als eigenständige Applikation für Microsoft Windows Betriebssysteme umgesetzt, in der dem Anwender komfortable Mechanismen wie z.B. Drag-and-Drop und Zugriffe auf das lokale Dateisystem eingeräumt werden. Insbesondere wurde das „Multiple Document Interface“ (MDI) verwendet, welches bereits in verschiedenen medizinischen Anwendungen erfolgreich eingesetzt wurde (z.B. [15,22,47,48]). Durch das MDI können mehrere Unterfenster in einem Hauptfenster geöffnet werden, wodurch ein paralleler Ablauf mehrerer Arbeitsprozesse ermöglicht wird.
Die Installation der Software kann auf sehr einfache Weise durch Kopieren und Bearbeiten bestimmter Dateien vorgenommen werden. Dies hat den Vorteil, daß weder Administratorenrechte noch Zugriffe auf zentrale

Systemkomponenten für die Installation notwendig sind. Auf diese Weise kann ein Anwender die Software mit wenigen Handgriffen selbst installieren, so daß der administrative Aufwand ähnlich gering ist, wie bei einer webbasierten Lösung [37,40].

5.2. XML als Codierungsstandard für die Datenverwaltung

Sowohl die Kommunikation zwischen MMO-Client und Backend, als auch die Client-interne Datenhaltung und -speicherung basieren vollständig auf der Extensible Markup Language (XML) [24]. Wie in vielen Bereichen der Computertechnologie, kann diese Objektbeschreibungssprache auch in der Medizinischen Informatik als eine der zukunftsweisendsten Standards angesehen werden, dessen aktuelle Entwicklung mit großem Interesse beobachtet wird (z.B. DTD vs. XML Schema als Standard zur Beschreibung der Struktur von XML-Dokumenten, [54]). So gibt es in anderen wissenschaftlichen Projekten beispielsweise Ansätze zur Entwicklung XML-basierter elektronischer Patientenakten [5,50], sowie Transformationen bestehender Kommunikationsstandards nach XML [19-21,31,32].

Die Verwendung von XML für die Kommunikation zwischen den Komponenten des MMO-Systems hat den Vorteil, das sowohl der MMO-Client als auch das Backend-System vollkommen unabhängig voneinander entwickelt werden konnten, da lediglich eine Absprache bzgl. der zu übertragenden XML-Strukturen getroffen werden mußte.

Durch diese Entkopplung war es einerseits möglich, den MMO-Client ohne technische Detailkenntnisse über die Datenbankstruktur und die Backend-Software zu erstellen. Andererseits hat die Verwendung von XML den Vorteil, daß hierdurch offene Schnittstellen für den Datenaustausch zwischen Backend und anderen Systemen geschaffen werden können. Auf diese Weise könnte beispielsweise ein Austausch mit der Datenbank der „Health Education Assets Library“ (HEAL) [30] entstehen, wodurch die Datenbestände innerhalb der MMO-Datenbank deutlich erweitert werden könnten. Außerdem garantiert diese Schnittstellenoffenheit die Möglichkeit zur Entwicklung alternativer

Benutzeroberflächen. Denkbar wäre beispielsweise eine webbasierte Oberfläche mit einem im Vergleich zum MMO-Client beschränkten Funktionsumfang, über die auch Benutzer alternativer Plattformen (z.B. Apple, Linux, ...) in den Inhalten des MMO-Systems navigieren können.

5.3. Verwaltung multimedialer Inhalte in MOBs

Innerhalb des in dieser Arbeit erörterten Systems konnten komfortable Mechanismen für die übersichtliche Verwaltung multimedialer Objekte entwickelt werden. Dabei werden die zu verwaltenden Multimediadateien in Form von sog. Medical Objects (MOBs) erfaßt und verwaltet. In einem MOB können auch mehrere Multimediadateien zu einer logischen Einheit zusammengefaßt und mit Zusatzinformationen (Attribute, Kommentare, ...) versehen werden. Die Abgrenzung dieser logischen Einheiten zueinander wird im Programm dadurch erreicht, daß jedes MOB in einem eigenen Fenster verwaltet wird, wodurch eine zusammenhängende Darstellung zusammengehöriger Inhalte ermöglicht wird. Die Übersichtlichkeit wird dabei durch die Unterteilung jedes dieser Fenster in verschiedene inhaltliche Bereiche in Form von virtuellen Karteireitern gewährleistet.
Der Ansatz, eine Folge mehrerer Multimediadateien als eigenständiges Objekt zu interpretieren, wurde in keiner der betrachteten Literaturquellen verfolgt, obwohl er den Vorteil bietet, Objekte, die nur im unmittelbaren Zusammenhang zueinander sinnvoll sind (z.B. Präsentationsfolien oder CT-Serien), strukturiert und zusammenhängend erfassen zu können.

Da Bilddaten in der Medizin besonders weit verbreitet sind, wurde das Programm um Funktionen zur Hervorhebung elliptischer und rechteckiger Bildbereiche („Regions of Interest“, ROI), die mit Annotationstexten versehen werden können, erweitert. Derartige Funktionen findet man auch in klinischen Bilddatenbank-Systemen (z.B. in [18]).

5.4. Steuerung des Datenzugriffs

Die differenzierte Steuerung des Zugriffs auf die gespeicherten MOBs ist eine wesentliche Anforderung an das System. Vergleicht man die Zugriffssteuerung des vorliegenden Systems mit denen der Dateisysteme von Unix-Systemen (z.B. Linux, Solaris, ...), so werden einige Parallelen ersichtlich: Auch in Unix-Systemen können verschiedenen Personenkreisen Schreib- und Leserechte eingeräumt werden [2]. Dabei werden neben dem Eigentümer einer Datei, dessen Rechte üblicherweise höchstens aus Gründen der Systemsicherheit beschnitten werden, die Personenkreise „group“ und „other people“ unterschieden, die mit den Benutzergruppen „Institusangehörige“ und „alle Anderen“ des MMO-Systems vergleichbar sind. Darüber hinaus kann für jedes MOB eine eigene „Autorenliste“ angelegt werden, für die die Zugriffsrechte separat verwaltet werden können. Folglich stellt der MMO-Client dem Anwender noch differenziertere Zugriffssteuerungen zur Verfügung, als die Dateisysteme Unix-basierter Multiuser-Systeme.

5.5. MOB-Suche anhand UMLS-basierter Attribute

Die Möglichkeit, relevante MOBs in der Datenbank zu finden, stellt ein weiteres, wichtiges Ziel dar. Hierfür wurde eine Funktion in das System integriert, über die die MOBs anhand ihrer Attribute gesucht werden können. Da bezüglich multimedialer Inhalte „kein einheitliches Dateiformat festlegbar“ [22] ist, können sowohl die Attributsnamen als auch die zugehörigen Ausprägungen beliebig vom Anwender definiert werden, wodurch ihm ein Höchstmaß an Flexibilität zur Verfügung steht. Dabei ist es möglich, die jeweils angegebenen Begriffe aus dem UMLS Meta-Thesaurus auszuwählen und ihre semantische Bedeutung anhand eines „Concept Unique Identifiers“ (CUI; eindeutiger Bezeichner für eine Gruppe von synonymen Begriffen) festzulegen. Die Suchanfragen können in analoger Form konstruiert werden, so daß die gewünschten Suchbegriffe nicht mehr von der Sprache oder der exakten Schreibweise abhängen. Vielmehr kann nach der tatsächlichen Bedeutung des Suchwortes, die durch den CUI identifiziert wird, gesucht werden [10]. Zusätzlich können die

semantischen Netze des UMLS dazu verwendet werden, die Suche mit einer optionalen Unschärfe zu versehen, um auch verwandte Begriffe automatisch in die Suche mit einzubeziehen. Dabei wird auch die semantische Nachbarschaft eines Suchbegriffs in der Suche berücksichtigt, wodurch die Anzahl der Suchergebnisse gesteigert werden kann. In diesem Zusammenhang kann der Anwender den Grad der zu verwendenden Unschärfe angeben, der einen Wert von eins (= exakte Suche) bis vier (= extrem unscharfe Suche) annehmen kann. Für jedes gefundene MOB wird dann ein Relevanz-Score errechnet, anhand dessen die Suchergebnisse in absteigender Reihenfolge sortiert werden. Auf diese Weise wird sichergestellt, daß dem Anwender stets die relevantesten Suchergebnisse als erstes präsentiert werden.
Die Vorteile einer Suche auf Basis des Metathesaurus und des semantischen Netzes des UMLS wurden beispielsweise auch im Rahmen des Projekts ARIANE [35,36] genutzt. ARIANE ist ein „Vermittlungs-framework“, mit dem verschiedene medizinische „heterogene Informationsquellen auf homogene Weise“ [35] abgesucht werden können. Dabei werden die verschiedenen Dokumente nach bestimmten UMLS-Begriffen, die vorgegebenen semantischen Strukturen entsprechen, abgesucht und indiziert. Die Suchanfrage eines Anwenders wird dann in einen UMLS-Begriff umgewandet und mit den oben genannten Indizierungsbegriffen verglichen. Dabei wird die Relevanz der Suchergebnisse anhand einer „semantic distance“ errechnet, dessen Verwendung vergleichbar ist mit einer unscharfen Suche im MMO-System.

5.6. Verknüpfungen von MOBs als weitere Suchalternative

Neben der Attributssuche gibt es auch die Möglichkeit, relevante MOBs anhand ihrer verwandtschaftlichen Beziehungen zueinander zu finden. Hierfür wurde das System um eine Funktion erweitert, die es dem Anwender erlaubt, Verweise auf verwandte Objekte in die eigenen MOBs zu integrieren. Diese Verknüpfungen können von anderen Benutzern verwendet werden, um sich auf der Suche nach bestimmten Objekten an der subjektiven Einschätzung des MOB-Autors zu orientieren, indem Sie, ausgehend von einem MOB, auf die

vom Autor als verwandt eingestuften Objekte zugreifen. Auf diese Weise könnten die Suchergebnisse der oben angesprochenen semantischen Attributssuche um subjektive, handverlesene Ergebnisse erweitert werden.

Im Rahmen der Verknüpfung von MOBs können die verwandtschaftlichen Verhältnisse sehr genau umschrieben werden. Zum einen ist es bei Bilddateien möglich, durch die Verwendung einer „Annotation-zu-MOB“-Verknüpfung anstelle einer „MOB-zu-MOB“-Verknüpfung einen genaueren Bezug herzustellen. Zum anderen wird die Art der verwandtschaftlichen Beziehung durch die Auswahl einer oder mehrerer vorgegebener Relationsbezeichnungen (z.B. „ist Vergrößerung von“, „wird textuell beschrieben durch“ etc.) genau definiert. Zusätzlich ist es möglich, die gewählte Verknüpfung mit einem optionalen, frei definierbaren Kontext zu versehen, um den Geltungsbereich der verwandtschaftlichen Beziehungen genauer zu spezifizieren.

Offensichtlich bieten diese Verknüpfungsmechanismen deutlich flexiblere Möglichkeiten, als beispielsweise die im Web üblichen Hyperlinks zur Verknüpfung von Dokumenten. Durch die Möglichkeit zur differenzierten Beschreibung der Beziehungen wurde ein semantisches Netzwerk über die in der MMO-Datenbank gespeicherten MOBs geschaffen, wie es sonst üblicherweise in medizinischen Wörterbüchern verwendet wird [7,33,49,52,53,58,59].

Der Ansatz, eine verwandtschaftliche Beziehung durch eine Relationsbezeichnung und einem Kontext zu beschreiben, wurde auch im Rahmen anderer wissenschaftlicher Projekte verfolgt. Als Beispiel wäre das System MDD-GIPHARM [7] zu nennen. Dieses Data Dictionary wurde mit einem semantischen Netzwerk versehen, dessen Verknüpfungen die Struktur *(obj-1, rel, obj-2, context)* haben, wobei *obj-1* und *obj-2* die zu verknüpfenden Einträge aus dem Data Dictionary sind. Die Einträge für *rel* und *context* geben die Bezeichnung der Relation und des Kontextes an. Im Gegensatz zu den Verknüpfungen im MDD-GIPHARM, deren Kontexte durch Einträge aus dem Data Dictionary repräsentiert werden, werden die Kontexte im MMO-System als Freitext angegeben. Dies ist notwendig, da die Kontexte im MMO-System auch

als Titel für die virtuellen Touren (vgl. Abschnitt 5.8) verwendet werden und somit nicht an einen festen Begriffsbestand gebunden werden können.

5.7. MOBs in Galerien speichern

Um auf die gefundenen MOBs auch zu späteren Zeitpunkten zurückgreifen zu können, ist es möglich, diese in sog. Galerien zu verwalten. Galerien sind Objekt-Kollektionen, die vom Anwender in beliebiger Anzahl angelegt und im lokalen Dateisystem gespeichert werden können. Per Drag-and-Drop können sowohl ausgewählte Suchergebnisse, als auch MOBs aus anderen Galerien auf komfortable Weise hinzugefügt werden. Dabei werden nicht die MOBs selbst gespeichert, sondern Referenzen auf die entsprechenden Datenbankeinträge, wodurch etwaigen Änderungen an den gespeicherten MOBs automatisch in den Galerien berücksichtigt werden. Des weiteren ist es auch möglich, eine Galerie in der Datenbank zu speichern, indem diese, ähnlich wie eine Multimediadatei, in einem MOB erfaßt und gespeichert wird. Diese Vorgehensweise hat zwei entscheidende Vorteile. Zum einen konnte der Entwicklungsaufwand reduziert werden, da keine separaten Verwaltungsfunktionen für Galerien entwickelt werden mußten. Zum anderen können sämtliche Funktionen, die für die Verwaltung von multimedialen Objekten entwickelt wurden (Suche und Verschlagwortung über UMLS, Verknüpfungen zu anderen MOBs, Zugriffsrechte, ...), unmittelbar auch für die Galerien verwendet werden.

Derartig komfortable Speicherungsmechanismen sind eher eine Seltenheit. In der „Health Education Assets Library“ (HEAL) [16,30] gibt es beispielsweise keine Möglichkeit, Verweise auf gefundene Objekte langfristig zu speichern. Alternativ steht dem Anwender ein „Download Folder“ zur Verfügung, in dem ausgewählte Suchergebnisse bis zum Logout gespeichert werden können. Der Inhalt dieses Ordners kann als ZIP-Archiv heruntergeladen werden, in dem sowohl die Multimediadateien als auch eine Meta-Beschreibung in Form einer IMS-XML-Datei [34] enthalten ist.

5.8. Interaktive Lehreinheiten in Form von virtuellen Touren

Auch die Möglichkeit zur Erzeugung interaktiver Lehreinheiten stellt ein wichtiges Ziel dieser Arbeit dar. Hierfür wurde eine Exportfunktion entwickelt, die es ermöglicht, verlinkte HTML-Seiten aus den Datenbeständen zu erzeugen. Die Strukturen und Inhalte dieser HTML-Dokumente ergeben sich aus einer geschickten Interpretation der Funktionen zur Verknüpfung von MOBs. In diesem Zusammenhang spielt die Metapher des „virtuellen Rundgangs“ eine wichtige Rolle. Ein virtueller Rundgang ist ein theoretisches Konstrukt aus verknüpften Positionsknoten, die jeweils aus einer Abbildung, einem beschreibenden Text und Navigationselementen, über die benachbarte Knoten erreicht werden können, bestehen. Diese Konstrukte eignen sich besonders gut zur Darstellung von Touren durch Gebäude, aber auch medizinische Lehrinhalte können unmittelbar nach diesem Konzept strukturiert werden, wie in [55] bestätigt wird. Um eine größtmögliche Flexibilität zu gewährleisten, wurden drei verschiedene Rundgangstypen entwickelt: Ungeführte-, freie geführte- und streng geführte Touren. Hierdurch kann die Komplexität der resultierenden Navigationsstrukturen flexibel angepaßt werden. Die Möglichkeiten reichen dabei von Navigationsstrukturen, in denen der Benutzer sich frei in jede Richtung bewegen kann, bis hin zu fest vorgegebenen Stationslisten, die nur der Reihe nach aufgerufen werden können.

Durch eine geschickte Verknüpfung verschiedener MOBs können derartige Rundgänge auf einfache Weise mit dem MMO-Client konstruiert werden. Auf Knopfdruck können diese Konstruktionen nach HTML exportiert werden. Dabei entspricht jede Position im Rundgang, bestehend aus einem normalen Bild, einem Panoramabild oder einer Flash-Animation sowie einem beschreibenden Text, genau einer HTML-Seite. Die Navigationsbereiche dieser Seiten ergeben sich aus den Verknüpfungen der beteiligten MOBs und aus dem gewählten Rundgangstyp. Insbesondere kann die Navigation durch die Integration von Lageplänen unterstützt werden. Die Gestaltung des Seitenlayouts kann mit Hilfe von zentralen Templatedateien ohne großen Aufwand individuell angepaßt werden. Diese Templates bestehen im wesentlichen aus HTML-Code, der mit Platzhaltern und kleinen XML-Segmenten versehen wird. Hierdurch ist es

insbesondere möglich, das Layout bestehender Rundgänge durch die Anpassung dieser zentralen Dateien für alle Positionsknoten zu verändern, wodurch die Gestaltung des Exports vereinfacht wird.

Die hier beschriebenen Exportfunktionen werden bereits erfolgreich im Projekt LOTSE [42,43] zur Erzeugung virtueller Bibliotheksrundgänge verwendet. Die erzielten Resultate sind mit der in [23] vorgestellten virtuellen Tour durch die „Hardin Library for the Health Sciences“ vergleichbar. Dies zeigt, daß sich die mit dem MMO-Client erzeugten virtuellen Touren durchaus mit von Hand erstellten, individuell zugeschnittenen „next-generation ... virtual tour(s)“ [23] messen können.

5.9. Ausblick

Die Erfahrungen, die das LOTSE-Team während der Erstellung der Bibliotheksrundgänge sammeln konnte, wurden im Entwicklungsprozeß des MMO-Clients berücksichtigt, wodurch die Anwenderfreundlichkeit des Systems kontinuierlich verbessert werden konnte. Allerdings wäre es wünschenswert, die Benutzerfreundlichkeit des Systems zusätzlich mit Hilfe etablierter Verfahren aus dem Bereich der „Human-Computer-Interaction“ sowie der „Cognitive Science“ [11,39,45,46] verifizieren zu können. In diesem Zusammenhang sind Methoden wie „usability engineering“ (z.B. Lautes Denken und Viedoanalysen des Anwenderverhaltens, angewendet z.B. in [14]), „rapid prototyping“ und „iterative development“ zu nennen, mit deren Hilfe repräsentative Benutzer stärker in die Entwicklung und Bewertung grafischer Benutzerschnittstellen einbezogen werden können. Da die Anzahl der tatsächlichen MMO-Anwender momentan allerdings noch sehr gering ist, ist eine derartige Verifikation lediglich im Zusammenhang mit möglichen Weiterentwicklungen des MMO-Clients denkbar. Denn neben den vielen Funktionen, die bereits in die Software integriert wurden, gibt es natürlich auch Aspekte, deren Umsetzung über den Rahmen dieser Arbeit hinausgehen.

In diesem Zusammenhang wäre beispielsweise der Begriff „Contentbased retrieval“ zu nennen, mit dem das Auffinden von Bilddaten anhand visueller

Ähnlichkeit umschrieben wird [6,8]. Für die Integration einer derartigen Funktionalität, die eine interessante Bereicherung darstellen würde, bildet das MMO-System eine geeignete technische Basis, so daß ein entsprechender Ansatz in zukünftigen Weiterentwicklungen verfolgt werden könnte.

Auch die Darstellung der Ergebnisse einer Datenbankabfrage könnte noch erweitert werden. Wie in Abschnitt 5.5 erwähnt, kann eine Datenbanksuche anhand UMLS-basierter Attribute durchgeführt werden. Dabei ist es möglich, über die Option „unscharfe Suche“ auch verwandte Begriffe mit in die Suche einzubeziehen. Alle Attribute, die aufgrund der Unschärfe gefunden werden, sind über das semantische Netz des UMLS mit einem der Suchbegriffe verbunden.

In der hier vorgestellten Version des MMO-Clients erhält der Anwender als Ergebnis einer Suchanfrage zunächst eine Liste der gefundenen Attribute, die in Form einer zweistufigen Baumstruktur dargestellt wird (vgl. Abschnitt 3.2.6.2). Zwar orientiert sich die Reihenfolge der Listeneinträge an der Relevanz der mit diesen Attributen jeweils assoziierten gefundenen MOBs, aber die verwandtschaftlichen Zusammenhänge der jeweiligen Begriffe werden aus dieser Struktur nicht ersichtlich. Daher wäre es denkbar, anstelle der zweistufigen Baumstruktur eine mehrstufige Darstellung zu entwickeln, anhand der die hierarchischen Zusammenhänge der gefundenen Begriffe ersichtlich werden. Basis für einen solchen Ansatz könnte die in [26] vorgestellte Anwendung sein. Dieses Programm, das wie der MMO-Client mit Borland Delphi entwickelt wurde, ermöglicht es, die polyhierarchische Struktur des semantischen Netzwerks des UMLS in einer monohierarchischen Baumstruktur darzustellen.

Auch die stetige Weiterentwicklung des UMLS (z.B. [13]) sollte in zukünftige Versionen des System integriert werden. Zwar werden sich die meisten Veränderungen auf die im Backend verwalteten Datenbestände beziehen, aber falls es zu grundsätzlichen strukturellen Änderungen des UMLS kommt, so könnten hierdurch auch funktionale Anpassungen des MMO-Clients notwendig werden.

6. Danksagung

Abschließend möchte ich mich bei all denjenigen bedanken, die mir bei der Anfertigung der Dissertation hilfreich zur Seite gestanden haben.

Mein besonderer Dank gilt Herrn Prof. Dr. Hans-Ulrich Prokosch für die interessante Aufgabenstellung sowie für die Betreuung und Förderung der Arbeit. Durch seine kompetenten Ratschläge und seine Offenheit für neue Ideen wurde die Erstellung dieser Arbeit in die richtigen Bahnen gelenkt, ohne den notwendigen wissenschaftlichen Freiraum einzuschränken.
Des weitern bedanke ich mich bei Herrn Dr. Thomas Frankewitsch, der für die Entwicklung des Backend-Systems verantwortlich ist. Ohne die vielen konstruktiven Diskussionen, in denen wir die verschiedensten Ideen für die Funktionen des MMO-Systems „ausgeheckt“ haben, wäre der MMO-Client sicherlich nicht das geworden, was er heute ist. Darüber hinaus erwies sich Herr Frankewitsch als geduldiger, hilfreicher Korrekturleser und konnte mir viele wertvolle Tips für den Einstieg in die Programmierung mit Borland Delphi geben.
Beim LOTSE-Team der ULB Münster möchte ich mich recht herzlich für die Anregungen zur Verbesserung der Benutzerfreundlichkeit des MMO-Clients bedanken. Besonders die Gespräche mit Frau Angelika Kachel und Herrn Holger Przibytzin hatten positiven Einfluß auf die Entwicklung der Funktionen zum Erstellen und Exportieren virtueller Touren.

Um eine Doktorarbeit zu einem erfolgreichen Abschluß zu bringen, ist ein ausgeglichenes soziales Umfeld unabdingbar. In diesem Zusammenhang möchte ich meiner Familie, besonders meinen Eltern und Geschwistern dafür danken, daß sie mich stets bei meinem Promotionsvorhaben unterstützt haben.
Last but not least bedanke ich mich bei meiner Freundin Jutta Lücking, die stets Freud und Leid der letzten drei Jahre mit mir geteilt hat. Ohne ihre anspornenden, aufmunternden Worte wäre es undenkbar gewesen, die Dissertation in der vorliegenden Form fertigzustellen.

7. Abbildungsverzeichnis

8. Literaturverzeichnis

[1] Bellon E, Wauters J, Fernandez-Bayo J, Feron M, Verstreken K, Van Cleynenbreugel J, Van den BB, Desmaret M, Marchal G, Suetens P (1997) Using WWW and JAVA for image access and interactive viewing in an integrated PACS. Med Inform (Lond) 22:291-300.

[2] Bourne S.R. (1983) The UNIX System. Addison-Wesley Publishing Company, S 120-124

[3] Boyer C, Baujard V, Scherrer JR (2001) HONselect: multilingual assistant search engine operated by a concept- based interface system to decentralized heterogeneous sources. Medinfo 10:309-313.

[4] Brandt CA, Nadkarni P, Marenco L, Karras BT, Lu C, Schacter L, Fisk JM, Miller PL (2000) Reengineering a database for clinical trials management: lessons for system architects. Control Clin Trials 21:440-461.

[5] Brelstaff G, Moehrs S, Anedda P, Tuveri M, Zanetti G (2001) Internet patient records: new techniques. J Med Internet Res 3:E8.

[6] Bucci G, Cagnoni S, De Dominicis R (1996) Integrating content-based retrieval in a medical image reference database. Comput Med Imaging Graph 20:231-241.

[7] Bürkle T, Prokosch HU, Michel A, Dudeck J (1998) Data dictionaries at Giessen University Hospital: past--present--future. Proc AMIA Symp 875-879.

[8] Cai W, Feng DD, Fulton R (2000) Content-based retrieval of dynamic PET functional images. IEEE Trans Inf Technol Biomed 4:152-158.

[9] Campbell KE, Oliver DE, Shortliffe EH (1998) The Unified Medical Language System: toward a collaborative approach for solving terminologic problems. J Am Med Inform Assoc 5:12-16.

[10] Campbell KE, Oliver DE, Spackman KA, Shortliffe EH (1998) Representing thoughts, words, and things in the UMLS. J Am Med Inform Assoc 5:421-431.

[11] Carroll JM (1997) Human-Computer Interaction: Psychology as a Science of Design. Annu Rev Psycho 61-83.

[12] Chu LF, Chan BK (1998) Evolution of web site design: implications for medical education on the Internet. Comput Biol Med 28:459-472.

[13] Cimino JJ (1998) Auditing the Unified Medical Language System with semantic methods. J Am Med Inform Assoc 5:41-51.

[14] Cimino JJ, Patel VL, Kushniruk AW (2001) Studying the human-computer-terminology interface. J Am Med Inform Assoc 8:163-173.

[15] Ciuca I, Jitaru E, Alaicescu M, Moisil I (2000) A neural network ActiveX based integrated image processing environment. Stud Health Technol Inform 77:1210-1214.

[16] Dennis S, Uijtdehaage S, Candler C (2001) Introducing the Health Education Assets Library: a National Multimedia Repository. Multimedia in Health Sciences Education 97-101.

[17] Dersch, H., URL: http://www.fh-furtwangen.de/~dersch/ (Zuletzt besucht am 04.04.2002)

[18] Dionisio JD, Cardenas AF, Taira RK, Aberle DR, Chu WW, McNitt-Gray MF, Goldin J, Lufkin RB (1996) A unified timeline model and user interface for multimedia medical databases. Comput Med Imaging Graph 20:333-346.

[19] Dolin RH, Alschuler L, Behlen F, Biron PV, Boyer S, Essin D, Harding L, Lincoln T, Mattison JE, Rishel W, Sokolowski R, Spinosa J, Williams JP (1999) HL7 document patient record architecture: an XML document architecture based on a shared information model. Proc AMIA Symp 52-56.

[20] Dolin RH, Alschuler L, Boyer S, Beebe C (2000) An update on HL7's XML-based document representation standards. Proc AMIA Symp 190-194.

[21] Dolin RH, Rishel W, Biron PV, Spinosa J, Mattison JE (1998) SGML and XML as interchange formats for HL7 messages. Proc AMIA Symp 720-724.

[22] Drescher D (1999) ImageCollector - eine Multimedia-Datenbank. Multimedia in der Medizin Proceedings des Aachener Workshops am 27./28.10.1999 51-60.

[23] Duncan JM, Roth LK (2001) Production of the next-generation library virtual tour. Bull Med Libr Assoc 89:331-338.

[24] Extensible Markup Language (XML) Homepage des W3C, URL: http://www.w3.org/XML/ (Zuletzt besucht am 07.02.2003)

[25] Frankewitsch T, Filler TJ, Prokosch HU (1999) Entwicklung und Etablierung einer dezentralen multimedialen Objekt-Datenbank nach Maßgabe verteilter Nutzbarkeit sowie Sharing von Fachwissen und Ressourcen. Multimedia in der Medizin Proceedings des Aachener Workshops am 27./28.10.1999 39-49.

[26] Frankewitsch T, Prokosch HU (2000) Graphical tool for navigation within the semantic network of the UMLS metathesaurus on a locally installed database. Stud Health Technol Inform 77:847-851.

[27] Frankewitsch T, Prokosch U (1999) Multimedia explorer: image database, image proxy-server and search- engine. Proc AMIA Symp 765-769.

[28] Frankewitsch T, Prokosch U (2001) Navigation in medical Internet image databases. Med Inform Internet Med 26:1-15.

[29] Health Education Assets Library - Partnering with HEAL, URL: http://www.healcentral.org/joinheal.htm (Zuletzt besucht am 07.02.2003)

[30] Health Education Assets Library, URL: http://www.healcentral.org (Zuletzt besucht am 07.02.2003)

[31] Heitmann KU, Dudeck J (2002) Important Step to Fill the Gap? - The German SCIPHOX Project. XML Europe Conference Proceedings .

[32] Heitmann KU, Schweiger R, Dudeck J (2002) Discharge and referral data exchange using global standards - The SCIPHOX project in Germany. Stud Health Technol Inform 78:679-684.

[33] Humphreys BL, Lindberg DA, Schoolman HM, Barnett GO (1998) The Unified Medical Language System: an informatics research collaboration. J Am Med Inform Assoc 5:1-11.

[34] IMS Global Learning Consortium, URL: http://www.imsproject.org (Zuletzt besucht am 07.02.2003)

[35] Joubert M, Aymard S, Fieschi D, Fieschi M (2001) ARIANE: a mediation framework with health information sources. Medinfo 10:343-347.

[36] Joubert M, Fieschi M, Robert JJ, Volot F, Fieschi D (1998) UMLS-based conceptual queries to biomedical information databases: an overview of the project ARIANE. Unified Medical Language System. J Am Med Inform Assoc 5:52-61.

[37] Katz AS, Tilkemeier PL (1997) Multimedia image display: a view to the future. Curr Opin Cardiol 12:566-570.

[38] Kindler H, Peter RU, Baranov AE, Fliedner TM, Densow D (1998) Providing dermatological photographs using the multimedia extension of the international computer database for radiation accident case histories. Int J Med Inf 51:39-50.

[39] Kushniruk AW, Patel VL (1998) Cognitive evaluation of decision making processes and assessment of information technology in medicine. Int J Med Inf 51:83-90.

[40] Lin CC, Duann JR, Liu CT, Chen HS, Su JL, Chen JH (1998) A unified multimedia database system to support telemedicine. IEEE Trans Inf Technol Biomed 2:183-192.

[41] Matthies HK, von Jan U, Porth AJ, Tatagiba M, Stan AC, Walter GF (2000) Multimedia-based courseware in the Virtual Learning Center at the Hannover Medical School. Stud Health Technol Inform 77:541-545.

[42] Obst O (2000) Library Online Tour & Self-paced Education - Projekt der Bibliothek vom BMBF bewilligt: Bald kommt der LOTSE an Bord. med information 4:6.

[43] Obst O, Scholle U (2001) Bald kommt der Lotse an Bord: Projekt der Universitäts- und Landesbibliothek Münster bei Global Info. ProLibris 6:12.

[44] Open XML, URL: http://www.philo.de/xml/ (Zuletzt besucht am 07.02.2003)

[45] Patel VL, Kaufman DR (1998) Medical informatics and the science of cognition. J Am Med Inform Assoc 5:493-502.

[46] Patel VL, Kushniruk AW (1998) Interface design for health care environments: the role of cognitive science. Proc AMIA Symp 29-37.

[47] Patterton HG, Graves S (2000) DNAssist, a C++ program for editing and analysis of nucleic acid and protein sequences on PC-compatible computers running Windows 95, 98, NT4.0 or 2000. Biotechniques 28:1192-1197.

[48] Patterton HG, Graves S (2000) DNAssist: the integrated editing and analysis of molecular biology sequences in windows. Bioinformatics 16:652-653.

[49] Prokosch HU, Bürkle T, Storch J, Strunz A, Müller M, Dudeck J, Dirks B, Keller F (1995) MDD-GIPHARM: Design and Realization of a Medical Data Dictionary for Decision Support Systems in Drug Therapy. Informatik, Biometrie und Epidemiologie in Medizin und Biologie 26:250-261.

[50] Rassinoux AM, Lovis C, Baud R, Geissbuhler A (2002) XML as Standard for Communicating in a Document-based Electronic Patient Record: a Three Years Experiment. Stud Health Technol Inform 78:673-678.

[51] rfc2045: Multipurpose Internet Mail Extensions (MIME) Part One, URL: http://www.faqs.org/rfcs/rfc2045.html (Zuletzt besucht am 07.02.2003)

[52] Ruan W, Bürkle T, Dudeck J (2000) A dictionary server for supplying context sensitive medical knowledge. Proc AMIA Symp 719-723.

[53] Ruan W, Bürkle T, Dudeck J (2000) An object-oriented design for automated navigation of semantic networks inside a medical data dictionary. Artif Intell Med 18:83-103.

[54] Schweiger R, Hölzer S, Heitmann KU, Dudeck J (2001) DTDs go XML schema--a tools perspective. Med Inform Internet Med 26:297-308.

[55] Temkin B, Acosta E, Hatfield P, Onal E, Tong A (2002) Web-based Three-dimensional Virtual Body Structures: W3D-VBS. J Am Med Inform Assoc 9:425-436.

[56] TGIFImage Download-Seite, URL: http://www.delphi32.com/vcl/673/ (Zuletzt besucht am 07.02.2003)

[57] Uijtdehaage SH, Dennis SE, Candler C (2001) A Web-based database for sharing educational multimedia within and among medical schools. Acad Med 76:543-544.

[58] UMLS Information, URL: http://umlsinfo.nlm.nih.gov/ (Zuletzt besucht am 07.02.2003)

[59] Unified Medical Language System (UMLS), URL: http://www.nlm.nih.gov/research/umls/ (Zuletzt besucht am 07.02.2003)

[60] W3C Document Object Model, URL: http://www.w3.org/DOM/ (Zuletzt besucht am 07.02.2003)

9. Anhang

9.1. Dokumententyp Definition (DTD) für MOBs

Es sind drei Fälle zu unterscheiden, in denen ein MOB in das XML-Format umgewandelt werden muß: Beim Speichern in der Datenbank, beim Herunterladen aus der Datenbank und beim Speichern im lokalen Dateisystem des Anwenders. Da der XML-Code in allen drei Fällen sehr ähnlich aussieht, wird im folgenden lediglich die DTD dargestellt, gemäß der die MOBs beim Speichern in der Datenbank modelliert werden.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

      Document Type Definition for the representation of medical
      multimedia objects (MOBs).

-->

<!ELEMENT medobject (editor, title, accesslist, authorlist?,
          attriblist, comment, sourcejoinedlist?, mediapanel+)>
<!ATTLIST medobject id CDATA #IMPLIED>
<!ATTLIST medobject creationdate CDATA #IMPLIED>

   <!ELEMENT editor (EMPTY)>
   <!ATTLIST editor login CDATA #REQUIRED>
   <!ATTLIST editor sessionkey CDATA #REQUIRED>

   <!ELEMENT title (#PCDATA)>

   <!ELEMENT accesslist (access+)>

      <!ELEMENT access (EMPTY)>
      <!ATTLIST access group (authors|institute|world) #REQUIRED>
      <!ATTLIST access value (none|read|write) #REQUIRED>

   <!ELEMENT authorlist (author*)>

      <!ELEMENT author (EMPTY)>
      <!ATTLIST author a_id CDATA #REQUIRED>
      <!ATTLIST author name CDATA #IMPLIED>
      <!ATTLIST author firstname CDATA #IMPLIED>
      <!ATTLIST author salutation CDATA #IMPLIED>
      <!ATTLIST author institution CDATA #IMPLIED>
      <!ATTLIST author adress CDATA #IMPLIED>
      <!ATTLIST author zip CDATA #IMPLIED>
      <!ATTLIST author town CDATA #IMPLIED>
      <!ATTLIST author country CDATA #IMPLIED>
      <!ATTLIST author email CDATA #IMPLIED>
```

```
    <!ATTLIST author fon CDATA #IMPLIED>
    <!ATTLIST author fax CDATA #IMPLIED>

<!ELEMENT attriblist (attrib*)>

    <!ELEMENT attrib (name, value)>

       <!ELEMENT name (#PCDATA)>
       <!ATTLIST name cui CDATA #IMPLIED>

       <!ELEMENT value (#PCDATA)>
       <!ATTLIST value cui CDATA #IMPLIED>

<!ELEMENT comment (#PCDATA)>

<!ELEMENT sourcejoinedlist (rel*)>

    <!ELEMENT rel (mob+)>
    <!ATTLIST rel id CDATA #REQUIRED>

      <!ELEMENT mob (context?)>
      <!ATTLIST mob id CDATA #REQUIRED>

        <!ELEMENT context (EMPTY)>
        <!ATTLIST context name CDATA #REQUIRED>

<!ELEMENT mediapanel (caption?, shape*, binary)>
<!ATTLIST mediapanel mimetype CDATA #REQUIRED>
<!ATTLIST mediapanel clientfile CDATA #IMPLIED>

    <!ELEMENT capiton (#PCDATA)>

    <!ELEMENT shape (annotation?, rel*)>
    <!ATTLIST shape type (rectangle|ellipse) #REQUIRED>
    <!ATTLIST shape points CDATA #REQUIRED>

       <!ELEMENT annotation (#PCDATA)>

    <!ELEMENT binary (#PCDATA)>
```

9.2. Dokumententyp Definition (DTD) für Galerien

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

      Document Type Definition for the representation of galleries
      within the MMO-System.

-->

<!ELEMENT gallery (title, item*)>

   <!ELEMENT title (#PCDATA)>

   <!ELEMENT item (EMPTY)>
   <!ATTLIST item id CDATA #REQUIRED>
```

9.3. Dokumententyp Definitionen (DTDs) für Datenbankabfragen

9.3.1. DTD für die Suchanfrage

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

      Document Type Definition for performing database queries.
      Part 1: Search request

-->

<!ELEMENT searchrequest (fuzzy, attrib+)>
<!ATTLIST searchrequest login CDATA #IMPLIED>
<!ATTLIST searchrequest loginsession CDATA #IMPLIED>
<!ATTLIST searchrequest oldsearchsession CDATA #REQUIRED>

   <!ELEMENT fuzzy (EMPTY)>
   <!ATTLIST fuzzy level (1|2|3|4) #REQUIRED>

   <!ELEMENT attrib (name?, value?)>
   <!ATTLIST attrib searchoption (kann|muss) #REQUIRED>

      <!ELEMENT name (#PCDATA)>
      <!ATTLIST name cui CDATA #IMPLIED>

      <!ELEMENT value (#PCDATA)>
      <!ATTLIST value cui CDATA #IMPLIED>
```

9.3.2. DTD für die Übersicht der relevanten Attribute

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

      Document Type Definition for performing database queries.
      Part 2: Overview over relevant attributes

-->

<!ELEMENT searchresult (item*)>
<!ATTLIST searchresult searchsession CDATA #REQUIRED>

   <!ELEMENT item (item*)>
   <!ATTLIST item type (attribname|attribvalue) #REQUIRED>
   <!ATTLIST item caption CDATA #REQUIRED>
   <!ATTLIST item cui CDATA #IMPLIED>
```

9.3.3. DTD für die MOB-Liste

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--

      Document Type Definition for performing database queries.
      Part 3: List of MOBs belonging to a selected attribute

-->

<!ELEMENT imagelist (image+)>

   <!ELEMENT image (EMPTY)>
   <!ATTLIST image title CDATA #REQUIRED>
   <!ATTLIST image id CDATA #REQUIRED>
   <!ATTLIST image permission (r|w) #REQUIRED>
```

Lebenslauf

Markus Ruppel
Weseler Straße 3
48151 Münster
* 07.02.1973 in Münster
Familienstand: ledig, keine Kinder

Ausbildung / Grundwehrdienst

08/1979 bis 06/1983	Besuch der Annette-von-Dorste-Hülshoff-Grundschule in Münster.
08/1983 bis 06/1989	Besuch der Realschule Wolbeck in Münster.
08/1989 bis 06/1992	Besuch der gymnasialen Oberstufe des Gymnasiums Wolbeck in Münster. Abschluß: Abitur.
10/1992 bis 09/1993	Grundwehrdienst.
10/1993 bis 01/2000	Studium der Mathematik (Diplom) mit Nebenfach Betriebswirtschaftslehre an der WWU Münster. Abschlußnote: sehr gut.
Seit 04/2000	Promotionsvorhaben zum Dr. rer. medic. bei Prof. Prokosch am Institut für Medizinische Informatik und Biomathematik (WWU Münster).

Beruflicher Werdegang

04/1996 bis 10/1999	**Institut für Medizinische Informatik und Biomathematik (WWU Münster).** Anstellung als studentische Hilfskraft beim Projektteam „MedWeb“.
04/2000 bis 09/2002	**Binary Design GmbH (Münster).** Anstellung als Technical Consultant im Rahmen einer halben Stelle.
04/2000 bis 02/2003	**Institut für Medizinische Informatik und Biomathematik (WWU Münster).** Wissenschaftliche Tätigkeit im Zusammenhang mit der Erstellung eines Clients für eine am Institut erstellte Multimediadatenbank.
03/2001 bis 08/2001	**Universitäts- und Landesbibliothek Münster.** Anstellung im Rahmen einer halben Stelle als Softwareentwickler beim Projekt LOTSE.

Münster, 18.02.2003

Markus Ruppel