Fachbereich 4

Design und Implementierung einer Anwendung zum Visualisieren von Relationen zwischen Lernobjekten

Studienarbeit

Vorgelegt von:

Adriane Niepel
Matrikelnummer: 202121016
Studiengang: Computervisualistik

Betreuer:
Dipl.-Inf., Dipl.-Ing.(FH) Marc Santos,
Institut für Wissensmedien

Koblenz, im Januar 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Inhaltsverzeichnis

1 Einleitung in das Thema

Im Bereich des E-Learning wird das „filesharing“ zunehmend wichtiger. Alle Lernobjekte die einmal erstellt wurden, sollen in einer Datenbank gesammelt werden und für jeden autorisierten Benutzer zu jederzeit abrufbar sein.
Wichtig ist die Möglichkeit auf dieser Datenbank eine Suchanfrage zu stellen. Über eine einfache Suche nach bereits existierenden Lernobjekten können Arbeitsschritte erleichtert und verringert werden.

Die Wichtigkeit, Relationen zwischen Lernobjekten visuell verständlich darzustellen, ist auch ein Bereich der immer wichtiger zu werden scheint. Dies bedeutet, dass nach einer Suche auf einer Datenbank und deren Ergebnisrückgabe, nicht nur die Ergebnismenge ausgegeben, sondern vor allem auch die Relationen zwischen ihnen dargestellt werden sollen.
Diese Sortierung der Ergebnismenge bringt Verständniserleichterung für die Zusammenhänge einzelner Objekte. Dies bedeutet für den Benutzer eine effektivere Arbeit und weniger Aufwand beim Erstellen neuer Lernobjekte.

Bisher existieren zum Thema „Visualisierung von Relationen“ nicht viele Arbeiten. Aus diesem Grund sollte eine Anwendung entwickelt werden, mit welcher Suchanfragen an eine Datenbank gestellt werden können und die Ergebnismenge mit den entsprechenden Relationen visualisiert ausgegeben wird.

Um dies zu realisieren ist es notwendig die Datenbank (oder auch das Repository) näher zu betrachten und zu wissen, wie die Inhalte, in diesem Fall Lernobjekte, und deren Inhalte wiederum gespeichert werden.
Ist die Kommunikation mit dem Client und der Datenbank klar, so werden noch grundlegende Visualisierungstechniken benötigt, um die Ergebnisse hinterher graphisch darzustellen.

Da es bisher, wie schon erwähnt, kaum Visualisierungen von Relationen gibt, werden verschiedene Visualisierungsformen untersucht, um eine sinnvolle Art und Weise zu finden die Ergebnisse darzustellen.
Des Weiteren werden Prototypen evaluiert, die die erläuterten Visualisierungsansätze verwenden. Somit kann die Entwicklung auf deren Erfahrungen aufbauen.

Die Entwicklung der Anwendung „LOSVis“ (LearningObjectsSearchVisualization) wird von Vorüberlegungen über die in Kapitel 2 erläuterten Grundlagen geprägt.

Nach Designentscheidungen und Strukturüberlegungen konnte die Implementierung und Entwicklung mit Hilfe verschiedener Techniken stattfinden. Für das bessere Verständnis befinden sich Beispiele und Code, also alles bezüglich der Realisierung, in Kapitel 3.
Zusätzlich zeigt der Abschnitt „Ausblick“ viele Ideen, die während der Entwicklung entstanden sind, jedoch nicht mehr umgesetzt werden konnten. Diese könnten eventuell bei einer Weiterentwicklung hilfreich sein und Grundsteine legen, die nicht verloren gehen sollten.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen betrachtet werden, die für eine Entwicklung einer Software zu eben dem eingeführten Thema wichtig sind. Hier sollen ausschließlich theoretische Vorüberlegungen deutlich gemacht werden.

Zu betrachten ist auf jeden Fall die Datenbank Fedora und ihre Möglichkeiten, wie sie Lernobjekte, deren Eigenschaften und Relationen abspeichert und wie man diese ansprechen kann. Ansonsten kann keine sinnvolle Suche nach Daten stattfinden.

Des Weiteren gibt es eine Vielzahl von Möglichkeiten das Ergebnis hinterher visuell darzustellen. Da die Visualisierung nicht wahllos sondern bedacht gewählt sein soll, sollen hier einige Formen betrachtet werden, die für die Umsetzung der Visualisierung von Nutzen sein könnten.

Hinzu kommt noch die Betrachtung von Prototypen, die mit den verschiedenen Visualisierungstechniken arbeiten.
Durch Evaluierung einiger Teile dieser Prototypen, soll mehr Erkenntnis über die Vor- und Nachteile der verschiedenen Formen entstehen.

Die verschiedenen technischen Aspekte und Anforderungen an die Software folgen im anschließenden Kapitel und schlagen so die Brücke zu der Implementierung.

2.1 Fedora

Was ist Fedora überhaupt und was für einen Nutzen hat es für das Projekt?

Diese Frage stellt sich zunächst, wenn die Aufgabenstellung lautet mit diesem System ein Projekt zu realisieren. In diesem Abschnitt soll genau auf diese Fragestellung eingegangen werden. Es soll jedoch nur eine kurze Einführung in Fedora geben, um für das Verständnis notwendige Grundlagen zu bilden.
All die Erläuterungen, die nicht zu der Entwicklung dieses Projekts erforderlich sind, werden an dieser Stelle nur kurz umrissen und können bei Bedarf in entsprechenden Quellen genauer nachgelesen werden.

Fedora steht für „Flexible Extensible Digital Object Repository Architecture" *[s. 1].*

Fedora wurde entwickelt um ein Datenbanken-System zu schaffen, das speziell mit vielen verschiedenen Datenformen möglichst flexibel umgehen kann. Da heute nicht nur einfach strukturierte Daten, wie Schriftdokumente existieren, sondern auch komplexe Bild- und Audiodaten gespeichert und verarbeitet werden müssen, wurde dieses Repository-System entwickelt. Für sehr komplexe Objekte wurde es dann zunehmend wichtiger auch Relationen abspeichern zu können.
Zusätzlich ist Fedora ein Open Source Projekt, und darf darum auch überall ohne Probleme eingesetzt werden.
[vgl. 2]

Wie sehen jetzt also Objekte aus, die in Fedora gespeichert werden?

Der Inhalt von Objekten besteht nicht nur aus dem eigentlichen Inhalt, wie der Benutzer ihn auf dem Bildschirm sieht und man ihn sich allgemein vorstellt. Es gehören noch andere Dinge dazu. Objekte können für Inhalte wie Bilder, Bücher, Texte, Lernobjekte, Datensätze und viele andere Dinge stehen.
Jeder so genannte „Datastream" steht hierbei für einen Teil des kompletten Objekts. Und jeder „Dissiminator" repräsentiert einen Service um verschiedene Sichten auf den Inhalt des Objektes zu zeigen.
[vgl. 3]

Was genau dies zu bedeuten hat wird durch die folgenden Abbildungen und Beispiele klar.

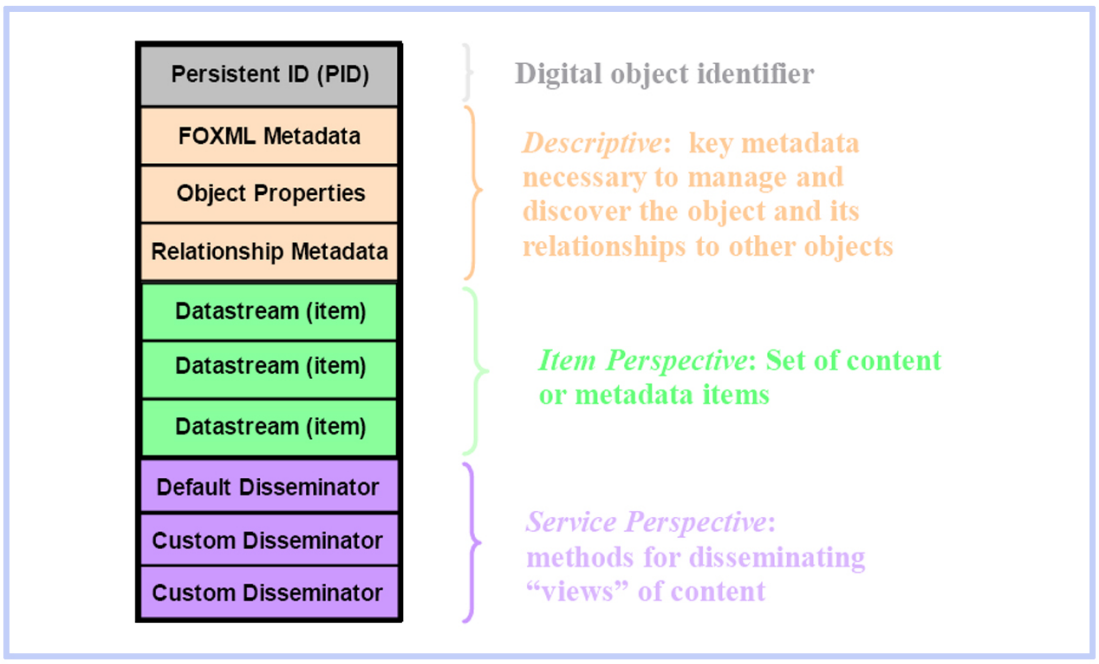

Abbildung 1: Fedora Objekt Modell

Die Abbildung zeigt einen Aufbau eines Fedora Objekts. Es wird durch 4 Teile repräsentiert.

Zunächst gibt es für jedes Objekt zwingenderweise einen Identifier. Er ist einmalig und kann das Objekt zu jeder Zeit genau identifizieren.

Der zweite Teil setzt sich aus Metadaten zusammen. Sie sind bedeutsam, um das Objekt zu verwalten. Er ist der beschreibende Teil des Objekts.
Hierzu gehören unter anderem Objekttyp, Datum der Erstellung und Modifizierung und natürlich ein Titel. Hier finden sich aber auch unter anderem die Relationen des Objekts zu anderen Objekten, die für eine Erweiterung von LOSVis dann wichtig werden.

Der dritte Teil beschreibt „Datastreams“. Wieviele ein Objekt davon besitzt ist nicht begrenzt. Die Datastreams beschreiben jeweils einen Teil des gesamten Inhalts des Objekts. Hier befindet sich ein Verweis auf den Ort des wirklichen Inhalts des Objekts, was zum Beispiel Bilder, Texte oder Audiodaten sein können. Alle Formen von Metadaten, außer System-Metadaten, werden als Inhalt behandelt und werden also auch als Datastreams repräsentiert. Die Inhalte, auf die in den Datastreams verwiesen wird, können im Repository liegen oder auf eine fremde Quelle verlinken.

Der letzte Teil eines Datenobjekts in Fedora sind die „Dissiminatoren“. Sie ermöglichen verschiedene Sichten auf den eigentlichen Inhalt, der sich in den Datastreams befindet.
Hier sind demnach die Funktionen enthalten, die auf den Inhalt angewandt werden können. So kann ein Dissiminator zum Beispiel zu einem Bild, welches sich in einem Datastream befindet, ein Thumbnail erzeugen. Dann befindet sich in dem Dissiminator zum Beispiel die Funktion „erzeugeThumbnail“. Von diesen Funktionen kann es zu einem Objekt beliebig viele geben.
[s. 4]

Als Beispiel hierzu lässt sich folgende Abbildung näher betrachten:

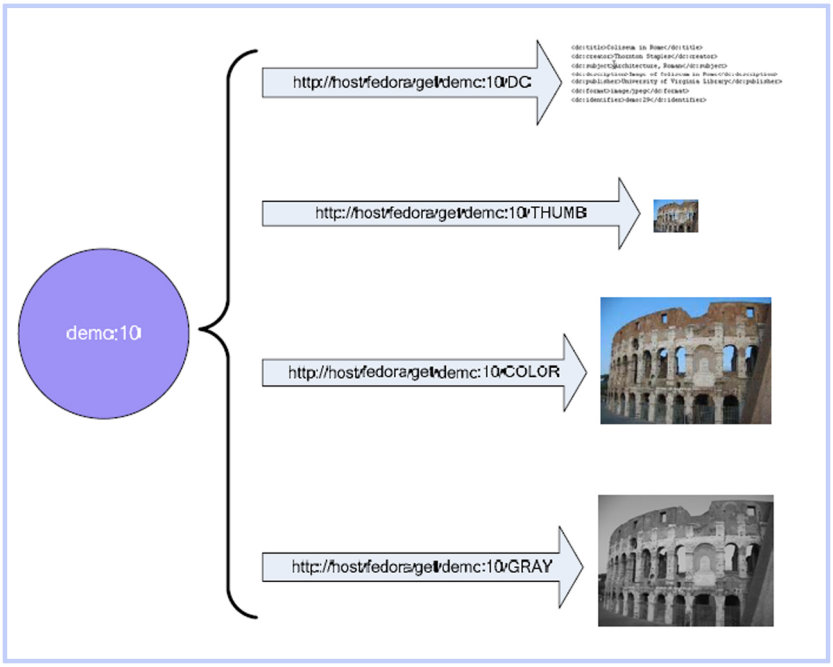

Abbildung 2: Fedora Datastreams Verweise

Das Objekt ist in diesem Fall „demo:10“. Es beinhaltet vier Datastreams. Der Erste ist ein Text, die anderen Drei sind Bilder. Eigentlich ist es jedoch nur ein einziges Bild. Die anderen beiden Bilder sind ausschließlich verschiedene Sichten auf dieses eine Bild. Die Ergebnisse der Sichten unterscheiden sich in Farbe und Größe. In der Datenbank abgespeichert ist hierzu nur ein einziger Inhalt.
Die Dissiminatoren werden auf das Bild angewandt und erzeugen aus einem Farbbild zum Beispiel ein Grauwertbild oder ein Vorschaubild. Dies lässt sich an der nachstehenden Grafik erkennen.

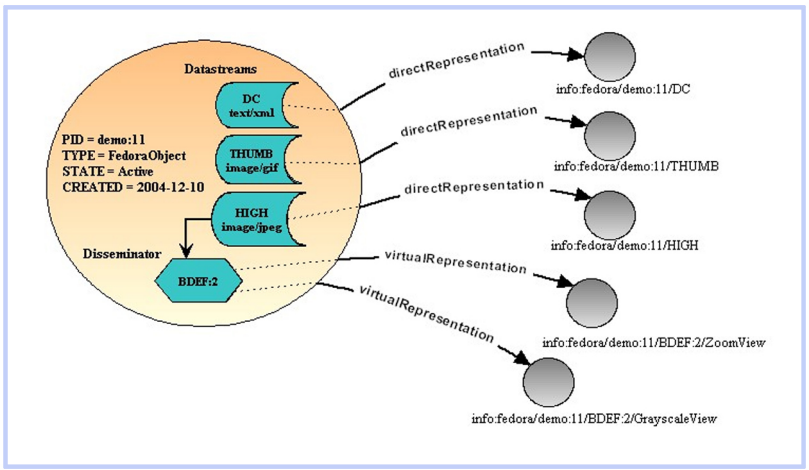

Abbildung 3: Fedora Objekt Model

Zu sehen ist zum Beispiel der Datastream „HIGH image/jpg“. Hier befindet sich der Verweis auf das Originalbild. Mit Hilfe des Dissiminators „BDEF:2“ werden virtuelle Repräsentationen erzeugt. Diese können dann zum Beispiel das Bild als Grauwertbild darstellen.
[vgl. 5]

Soviel soll an dieser Stelle zu dem grundlegenden Aufbau eines Fedora Objekts genügen. Der Zusatz der Relationen soll nur an dieser Stelle noch kurz angerissen werden. In der kommenden Abbildung sind Relationen zwischen verschiedenen Objekten sichtbar.

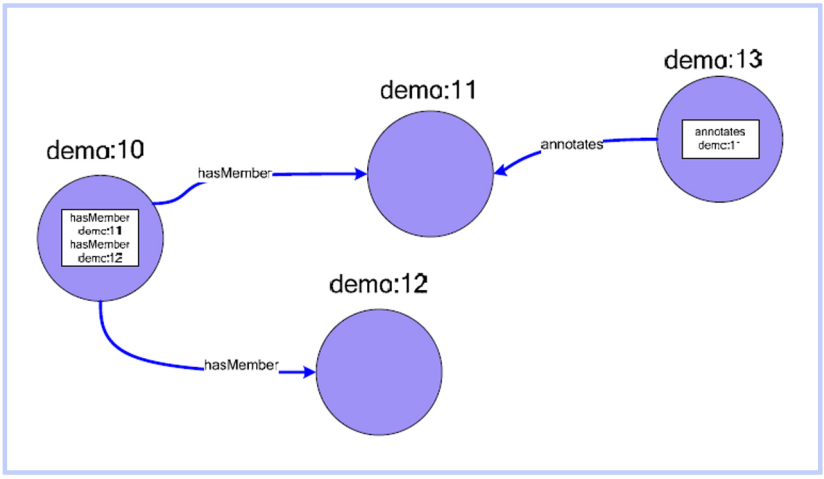

Abbildung 4: Fedora Relationen

Die dargestellten Pfeile stellen die Relationen zwischen den einzelnen Objekten dar. Sie werden wie in „demo:10“ und „demo:13“ zu erkennen ist, jeweils in dem Objekt selber abgespeichert. Diese Relationen befinden sich in den Metadaten des Objekts, also wie bereits erwähnt in dem Beschreibungsteil des Objekts.

2.2 Visualisierungstechniken

Um Suchergebnisse zu visualisieren, gibt es verschiedene Formen. Um vor allem die Relationen zwischen den Ergebnissen deutlich zu machen, eignen sich manche Techniken natürlich besser als andere.

Eine Möglichkeit besteht zum Beispiel daraus, die Ergebnisse der Suche in einer Liste darzustellen (vergleiche „Google“ und ähnliche Suchmaschinen). Die Relationen könnten in diesem Fall farblich gekennzeichnet werden.

Sinnvoller erscheint aber die Möglichkeit durch Netzstrukturen den Benutzer besser und intuitiver ansprechen zu können. Denn die Darstellung der Relationen ist vor allem dafür da, den Benutzer in seiner Arbeit zu unterstützen. Hierfür soll die Darstellung nicht verwirren, sondern möglichst einfach zu verstehen sein.

Zunächst sollen Vater-Tochter-Beziehungen visualisiert werden, hierzu eignet sich eine hierarchische Baumstruktur. Diese ist in dem Prototyp LOSVis realisiert worden.

Die hier folgenden weiteren Überlegungen zu anderen, besseren Visualisierungsmöglichkeiten, wurden jedoch im Vorfeld angestellt und die Ergebnisse sollen trotz fehlender Umsetzung an dieser Stelle festgehalten werden.
Wenn ein Algorithmus entwickelt ist, der die Baumstruktur ablösen könnte, so ist diese recht einfach einzupflegen und der Layout Algorithmus für die Baumstruktur auszutauschen.

In diesem Kapitel sollen nun verschiedene Visualisierungsmöglichkeiten näher betrachtet werden. Alle Techniken stellen hierarchische Strukturen oder Netzstrukturen dar.
Es werden unter anderem farbliche Kennzeichnungen und variable Größen der Knoten verwendet. Durch diese Darstellung können die Knoten der verschiedenen Hierarchie-Tiefen gleichzeitig betrachtet und im Zusammenhang verglichen werden.

Zu den im Folgenden betrachteten Techniken gehören:

„Tree Map“ (2.2.1),

„Hyperbolic Tree“ und „Degree-of-Interest Tree“ (2.2.2),

„Venn-diagrams“ (2.2.3) und

„Space Tree“ (2.2.4).

2.2.1 Tree Map

Folgende Grafiken sollen erst einmal einen Eindruck verschaffen, wie Visualisierungen mit Hilfe der „Tree Map“-Technik aussehen können.

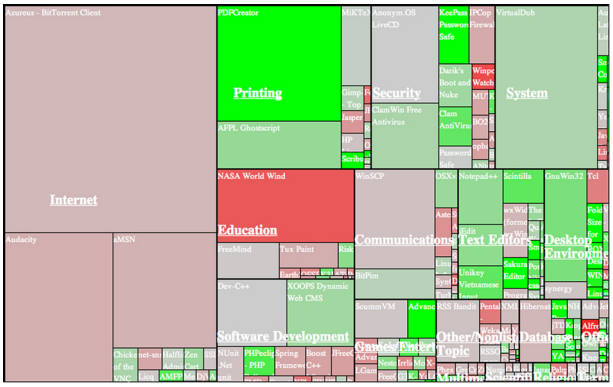

Abbildung 5: Treemap Variante 1

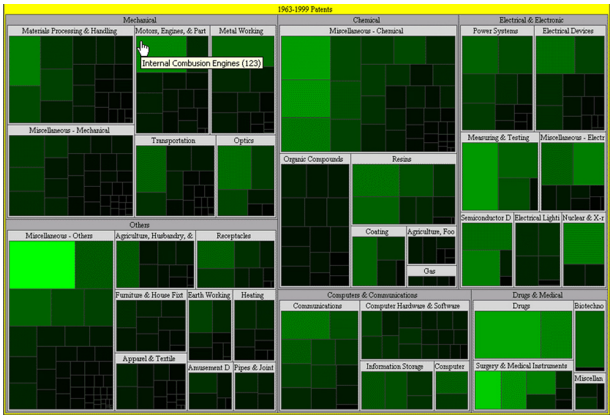

Abbildung 6: Treemap Variante 2

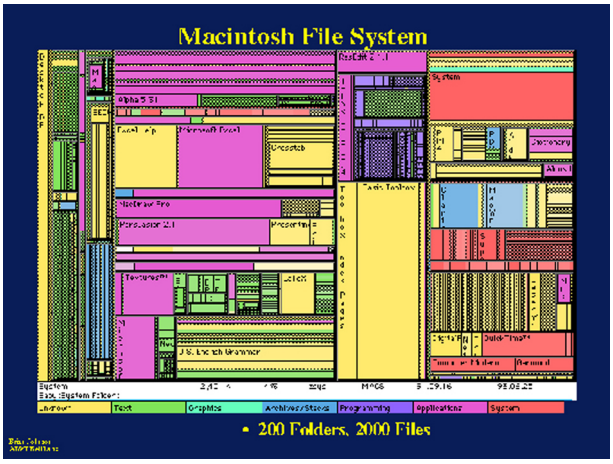

Abbildung 7: Treemap Variante 3

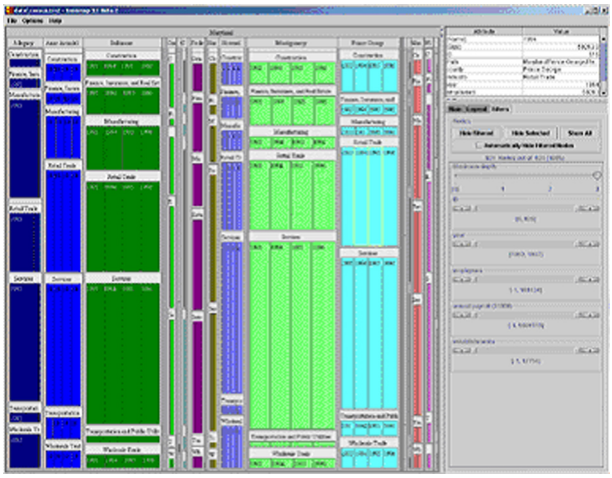

Abbildung 8: Treemap Variante 4

Alle Darstellungen sind so genannte „Tree maps“. Tree maps teilen den Raum in kleinere Räume um Wichtigkeit und Zusammengehörigkeit darzustellen. Dazu werden zusätzlich Farben verwendet, um einen schnelleren Überblick zu gewährleisten.
In den ersten beiden Abbildungen wird eine horizontale Anordnung gewählt, in der dritten Abbildung gibt es eine Mischform von horizontaler und vertikaler Anordnung, was ein wenig chaotisch wirken mag.
In der letzten Abbildung ist eine vertikale Anordnung zu erkennen.

Tree maps haben den Vorteil, dass mehrere Ebenen gleichzeitig sichtbar sind und wenn das Prinzip einmal verstanden, die Übersicht ziemlich leicht zu behalten ist.
Durch die farbliche Unterteilung kann zusätzlich gute Hilfestellung für das Verständnis geleistet werden.

Der Nachteil an dieser Technik ist ganz klar, dass es keine Relationen unter den verschiedenen Hierarchiestufen geben kann, denn eine Querverknüpfung zu einem anderen Objekt ist nicht darstellbar.
[vgl. 9]

2.2.2 Hyperbolic Tree und Degree-of-Interest Tree

Der „Hyperbolic Tree“ bezeichnet, wie in Abbildung 9 zu sehen, eine Baumstruktur. Hier sind alle Verknüpfungen zu dem ausgewählten Knoten, der standardmäßig in der Mitte angeordnet wird, zu sehen.
Diese Darstellungsform erlaubt die Visualisierung sehr vieler Informationen. Nur die Wichtigsten sind in dem Augenblick sichtbar, aber alle anderen sind vorhanden und können ohne großen Aufwand sichtbar gemacht werden. Der Zusammenhang von großen Datenmengen kann so einfacher und verständlicher dargestellt werden. Der Benutzer kann intuitiv die Äste verschieben und seinen Point-of-Interest auswählen und diesen jederzeit ändern.
[vgl. 7]

Bei großen Datenmengen kann es jedoch zunächst zu einer Informationsüberflutung führen. Bis der Benutzer die Zusammenhänge kennt und sich alle Äste angeschaut hat, dauert es unter Umständen einige Zeit.
Die Schrift wird unter Umständen zu klein, so dass diese kaum lesbar ist. Einige Äste werden, wie in der Abbildung zu sehen ist, verkleinert dargestellt und „auslaufen“ gelassen, was soviel bedeuten soll, dass Verknüpfungen außerhalb des Graphen weitergehen, jedoch an dieser Stelle abgebrochen werden.

Je nach Zusammenhang im Kontext und Wichtigkeit der einzelnen Äste, entsteht eine Struktur, wie sie in Abbildung 10 weiter unten zu sehen ist. Mit dieser Darstellungsform können nur hierarchische Strukturen dargestellt werden.

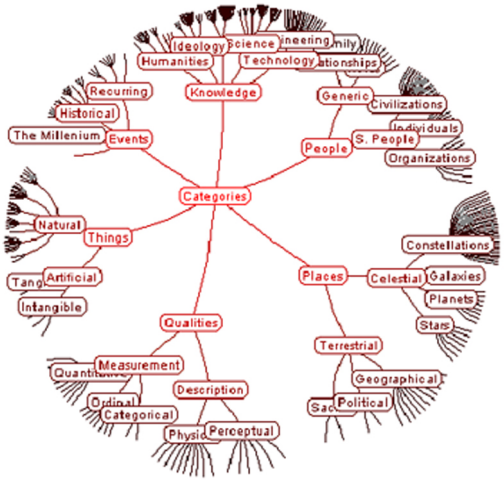

Abbildung 9: Hyperbolic Tree 1

In der Abbildung sieht man ein Beispiel zu dem Hauptbegriff „Categories“ einer Datenbank.
Vorteil an dieser Darstellung ist, dass die Hierarchisierung auch durch die farbliche Kennzeichnung schneller zu erkennen ist und der Blick direkt auf das Wesentliche und den Hauptbegriff gelenkt wird.

Nachteilig ist das Verschwinden von Informationen. Der Nutzer ahnt eventuell nichts von den Informationen, die sich hinter den Ästen verbergen, die für ihn jedoch eventuell auch relevant sein könnten.

Eine bessere Lösung an dieser Stelle ist durch die Weiterentwicklung zu einem „Degree-of-Interest Tree“ zu erreichen.

In diesen Bäumen ist die Anordnung der einzelnen Objekte interaktiv. Das bedeutet, dass sich das Objekt des größten Interesses in der Mitte (beziehungsweise immer im Blickpunkt) befinden soll.
Das heißt, dass nach jeder neuen Auswahl des interessantesten Knotens ein neuer Aufbau stattfindet. Der Baum sieht somit nicht immer gleich aus, im Gegensatz zum Aufbau bei „Tree Maps“(Kapitel 2.2.1).

Der Benutzer muss sich dafür jedoch genau mit den Ästen beschäftigen, denn in einem Moment sind die Äste auf der linken Seite, nach einer neuen Prioritätssetzung und der neuen Anordnung befinden sich Diese dann eventuell rechts. Dem Benutzer muss also leicht verständlich angezeigt werden wie sich das Diagramm verändert hat.
Auch hier werden hierarchische Strukturen dargestellt, wobei Querverknüpfungen nicht vorgesehen sind.

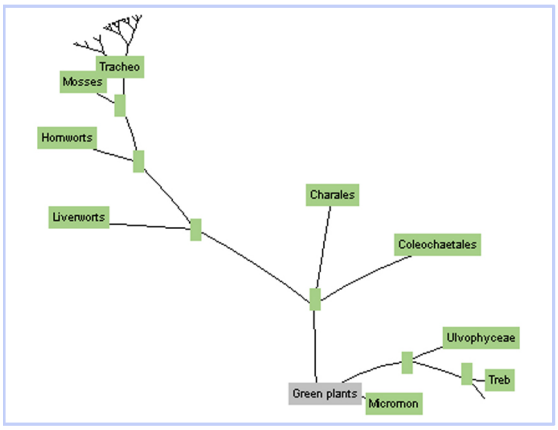

Abbildung 10: Hyperbolic Tree 2

In der Abbildung kann man einen Hyperbolic Tree sehen, der die Funktion des „Degree-of-Interest Tree“ beinhaltet. Durch Klicken auf einen Begriff kann dieser verschoben werden. Standardmäßig wird er in die Mitte des Bildschirmes gerückt, jedoch kann er durch das Gedrückt-halten der Maus, wie oben geschehen, an den unteren Rand oder eine andere Stelle verschoben werden. Die bisher zu sehenden Äste ordnen sich neu um den gewählten Begriff an. Eventuell verschwinden hierbei Äste und andere bisher nicht zu sehenden Äste und deren Begriffe erweitern den Baum. Sie reagieren also auf die Begrenzung der Visualisierungsfläche und verbiegen sich oder klappen sich um, je nach Platzmöglichkeit.

Auch in dem folgenden Beispiel sieht man, dass alle Knoten klein dargestellt werden, außer der „Node of interest“ und seine Relationen, also seine Väter. Ein Skalierungsalgorithmus wertet die Wichtigkeit aus und legt somit die Knotengröße fest. Bei den kleineren Knoten wird gar kein Text dargestellt. Welches Objekt sich genau hinter diesem Knoten verbirgt, ist jedoch so nicht einsehbar.

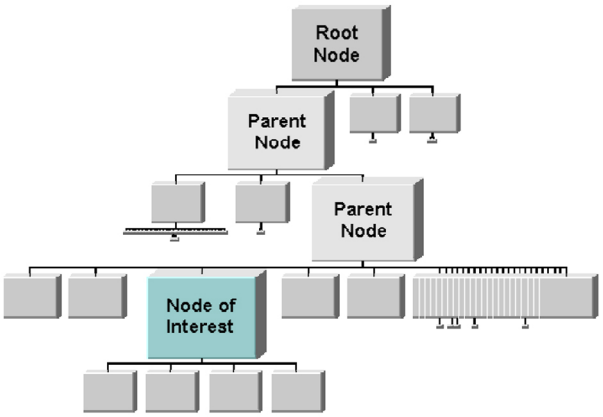

Abbildung 11: Degree-of-Interest-Tree

2.2.3 Venn-diagrams

„Venn diagrams“ werden speziell zur Visualisierung von Relationen verwendet, wenn eine Objektgruppe Gemeinsamkeiten aufweist.

In der folgenden Abbildung sieht man drei Mengen von Objekten und deren Relation zueinander.
X, Y, Z bezeichnen drei Mengen, die verschiedene Eigenschaften besitzen. Diese Mengen besitzen Objekte, die auch Eigenschaften von anderen Mengen haben können.
Dies kann vorkommen (siehe Bereiche rot, blau, grün), muss aber nicht (siehe Bereiche gelb, cyan, magenta.). Alle Elemente in dem mittleren Teil, dem schwarzen Rechteck (X n Y n Z), sind Elemente die alle vorkommenden Eigenschaften vereinen. Dargestellte Überschneidungen lassen sich intuitiv interpretieren.
[vgl. 10]

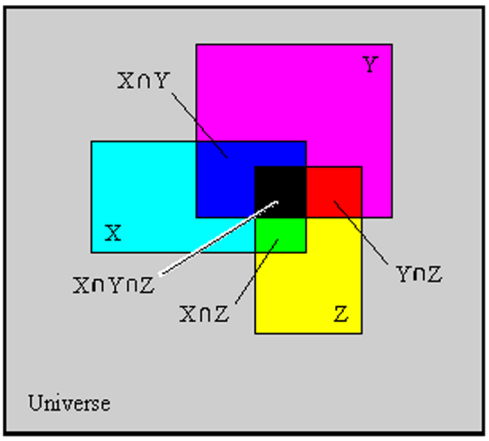

Abbildung 12: Venn Diagramm Variante 1

In dieser Darstellungsart lassen sich AND-Verknüpfungen gut darstellen, jedoch wird dies sicher unübersichtlich, wenn zu viele Objekte miteinander verknüpft werden. Sollen zu viele Überschneidungen mit verschiedenen Objekten erfolgen, ist dies unter Umständen graphisch einfach nicht mehr umsetzbar.
Ein konkreteres Beispiel hierzu ist das Folgende:

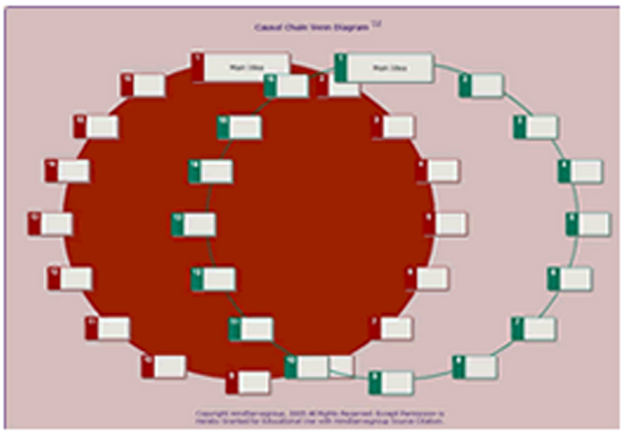

Abbildung 13: Venn Diagramm Variante 2

Recht deutlich zu erkennen ist, dass hier einzelne Elemente dargestellt werden, die sich auf dem Venn Diagramm anordnen. Sie sind eingeteilt in die roten und grünen Bereiche, aber es gibt auch Überschneidungen. Hier wurde wie oben die typische Kreisform genutzt.
Mehrdimensionalität kann jedoch nicht wirklich dargestellt werden.
Sollte dies jedoch nötig sein, gibt es Möglichkeiten Venn Diagramme auf eine veränderte Art und Weise darzustellen. Die folgenden Abbildungen zeigen verschiedene Darstellungsweisen.

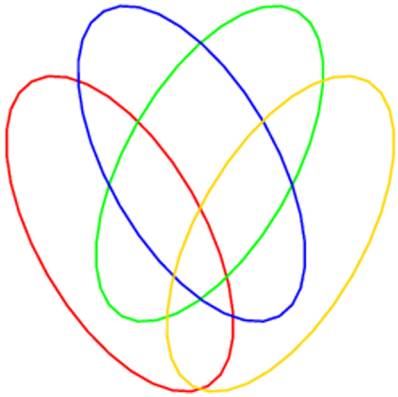

Abbildung 14: Venn Diagramm Variante 3

In diesem Fall ist auch der Kreis als Grundform gewählt, jedoch wurden Ellipsen geformt. Diese haben schon mehr Möglichkeiten Überschneidungen darzustellen und können Mehrdimensionalität besser ausdrücken und mehr verschiedene Gruppierungen darstellen.
Ist dies jedoch auch noch nicht ausreichend, so kann dies auch noch 3-dimensional dargestellt werden. Auch eine solche Abbildung kam durch Recherche heraus:

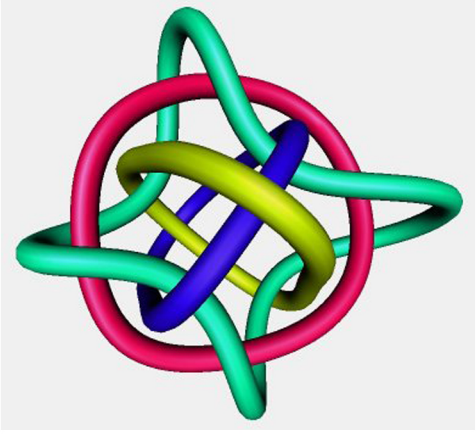

Abbildung 15: Venn Diagramm Variante 4

Sich vorzustellen wie die einzelnen Elemente dargestellt werden sollen und der Benutzer noch den Überblick bewahren kann, ist hierbei jedoch wirklich schwer.

2.2.4 Space Tree

Ein „Space tree“ bezeichnet ein Diagramm, in dem durch die Anordnung eine bestmögliche Platzeinteilung erreicht werden soll.
Diese Baumstruktur bezeichnet die gesamte räumliche Aufteilung. Sie passt sich den Platzgegebenheiten an und ändert die dargestellte Form des Diagramms dynamisch je nach Interaktion des Benutzers.
[vgl. 11]
Als Beispiel hierzu dient der Prototyp in Kapitel 2.3.2.

Vorteil an dieser Stelle ist natürlich, dass der Platz optimal ausgenutzt wird.
Der Nachteil, dass sich die Form der Struktur ständig ändert und zu Verwirrung führen kann, muss je nach Einsatzmöglichkeit beurteilt werden.
Weitere Erläuterungen und Beispiele zu „Space Trees“ finden sich im folgenden Kapitel zu entsprechendem Prototyp.

2.3 Welche Erkenntnisse (oder vorhandene Programme) gibt es auf dem Markt?

Für die Visualisierung von Relationen von Objekten (in diesem Fall Lernobjekte) gibt es keine allgemein bekannten, funktionsfähigen Programme.
Da aber einige Prototypen existieren, die sich mit ähnlichen Themen beschäftigen, werden zwei Dieser im Folgenden genauer betrachtet.
Sie verwenden die im Kapitel 2.2 vorgestellten Techniken.
Die Prototypen sind jeweils auf ein Thema spezialisiert und noch nicht für andere Bereiche getestet.
Eine Aussage, ob die Ideen, die umgesetzt wurden vielleicht auch für LOSVis brauchbar sind, sollte nach der Evaluierung Dieser besser zu beurteilen sein.

2.3.1 Prototyp Treemap, Hyperbolic Tree

Dieser Prototyp ist von Klerkx, Meire, Ternier, Verbert, Duval.
Alle Informationen entstammen einem Paper. Leider war es nicht möglich das Programm zu testen.
Doch einige interessante Aspekte konnten auch in dem hierzu existierenden Paper gefunden werden.
Es werden drei verschiedene Visualisierungsformen zur verschiedenen Ansicht eingebaut.
Sowohl die Technik des „Tree Map“ als auch des „Hyperbolic Tree“ und „Venn Diagramme“ sollen beim Einsatz der Software möglich sein.
Eine einfache Suchanfrage kann gestartet werden und die Ergebnisrückgabe geschieht sowohl in einer Liste, als auch in einem Diagramm, das die einzelnen Objekte zusätzlich ordnet. Der Vorteil an einer Tree Map wird darin gesehen, den Überblick über alle Ergebnisse zu behalten und eine hierarchische Klassifizierung zu erhalten.
Das System sollte in verschiedenen Umgebungen zu verschiedenen Zwecken einsetzbar sein, darum wurden verschiedene Visualisierungstechniken eingebaut, die nach Bedarf umgestellt werden können.
Interessant an der graphischen Oberfläche ist auf jeden Fall die Möglichkeit einen Regler zu benutzen, der die Anzahl der Suchergebnisse verringert oder erhöhen kann.
[vgl. 6]

2.3.2 Prototyp Space Tree, Degree-Of-Interest Tree

Der Prototyp „Eco lens“ sieht folgendermaßen aus:

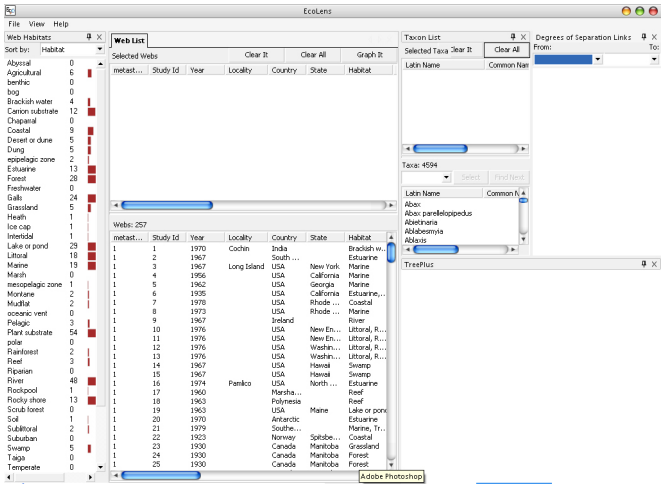

Abbildung 16: Prototyp Eco Lens

Dies ist die Umgebung, in dem eine Suche stattfinden kann. Durch auswählen der einzelnen Elemente verändern sich jeweils die anderen Listen und passen sich an.
Ist ein Element ausgewählt, so wird rechts unten ein Graph zu dieser Anfrage erstellt. Dieser ist eine Erweiterung zu den geführten Listen und stellt die Zusammenhänge zusätzlich graphisch dar. Dies ist der so genannte „TreePlus“.
Er benutzt das Prinzip des „Degree-of-Interest Tree“ und sieht wie folgt aus:

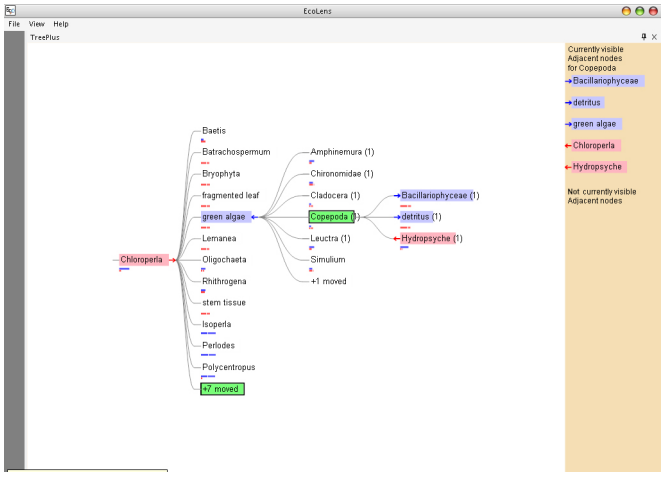

Abbildung 17: Tree Plus

Die rosa unterlegten Objekte stellen die ausgewählten Objekte dar. Nach dem Auswählen öffnet sich ein neuer Ast. Der Baum bezeichnet eine hierarchische Struktur und wächst immer nur nach rechts. Dies bedeutet, dass das Thema nach rechts immer spezialisierter wird. Der Baum kann in diesem Fall auch Wege rückwärts anzeigen. Dies kann dem kleinen roten Pfeil entnommen werden. Der unterlegte Balken rechts zeigt die nächsten Objekte an, die angehangen werden, wenn das grün markierte Objekt ausgewählt wird.

Noch ein anderes Beispiel für einen Prototyp eines Degree-of-Interest Tree ist der „TaxonTree“. Genau wie auch schon in dem Beispiel oben, wird das Objekt, welches den Benutzer interessiert angeklickt und es öffnen sich weitere Informationen bzw. Relationen hierzu weiter nach rechts im Baum.
[vgl. 12]
In der folgenden Abbildung kann man ihn im oberen Teil sehen.

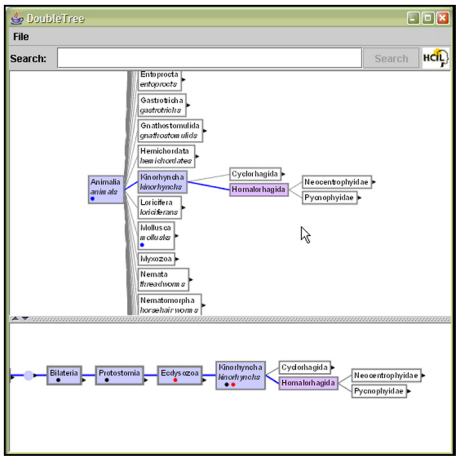

Abbildung 18: Double Tree

Eine Möglichkeit dies noch zu erweitern ist der „Double tree“.
Betrachtet man nur den oberen Teil sieht man einen Degree-of-Interest Tree, den „Taxon Tree“. Beides zusammen ergibt dann die Möglichkeit zwei Taxon Trees miteinander zu vergleichen, was dann den „Double Tree“ ausmacht. Durch das Klicken auf die Knoten öffnen sich die nächsten Verzweigungen. Über die Suchfunktion wird in beiden Bäumen derselbe Knoten gefunden und hervorgehoben.
[vgl. 12]

2.4 Fazit

Für eine Applikation wie LOSVis wäre eine Mischung aus den vorgestellten Netzstrukturen sinnvoll.
Um nur alleine ein Objekt und deren Inhalte darzustellen ist sicherlich eine Baumstruktur am intuitivsten. Diese müsste jedoch nicht den klassischen Aufbau haben, sondern könnte wie zum Beispiel der „Tree Plus“ aussehen. Damit ist gemeint, dass er nicht von oben nach unten, sondern auch von links nach rechts aufgebaut werden kann, trotz allem aber einen Baum, und somit eine hierarchische Struktur, repräsentiert.
Der Vorteil an einer vertikalen Struktur ist, dass der Benutzer, falls viele Einträge vorhanden sind, nach unten scrollen muss, und nicht zur Seite.
Wenn natürlich mehrere Ebenen existieren, dann muss auch hier nach rechts gescrollt werden, doch für die ersten Ebenen ist dies nicht notwendig.

Mit dem Hintergedanken an die Softwareergonomie ist das Scrollen nach unten einfacher und angenehmer, und sollte daher auch hier bevorzugt werden.
An dieser Stelle sei zu erwähnen, dass leider nicht genug Zeit blieb dies umzustellen, da der verwendetet Baumstruktur-Layout-Algorithmus von Netron den klassischen Aufbau hat und der Benutzer nun erst einmal gezwungen wird nach rechts zu scrollen.

Natürlich ist eine gute Aufteilung des kompletten Raumes wichtig.
Entsprechende Anordnung der Menüs, sowie der Suchfunktion und der Ergebnisliste wurden umgesetzt.

Bei dem Graph wäre es natürlich ein guter Zusatz, wenn sich der Baum dynamisch anpassen würde, falls der Benutzer einen Knoten näher betrachten will. Dies ist im Moment noch bedingt umgesetzt. Durch das Ein- und Auszoomen sollte der Benutzer diese Möglichkeit haben, jedoch, wenn Relationen zwischen den einzelnen Objekten dargestellt werden und mehrere Lernobjekte in dem Graphen auftauchen, dann sollte die Möglichkeit gegeben sein ein Hauptobjekt, oder auch „Point-of-interest“ zu wählen. Dafür eignet sich zweifellos das Prinzip des Degree-of-Interest Trees.

Vergleiche zweier Bäume werden bei LOSVis nicht benötigt. Dieser Teil wird darum nicht näher betrachtet. Hierzu würden auch Venn Diagramme einen Nutzen bringen. Diese werden vor allem zum Vergleichen und zum Finden von Gemeinsamkeiten zwischen Objekten verwendet.

Um die Eigenschaften und die Inhalte der einzelnen Objekte einem Objekt zuzuordnen wird eine hierarchische Struktur benötigt.
Für eine Erweiterung des Systems und einer Darstellung der Relationen zwischen allen Lernobjekten würde dies nicht mehr reichen und eine Netzstruktur müsste dargestellt werden. Diese wäre neu zu entwickeln und müsste wohl einen Kompromiss wie oben beschrieben darstellen.

3 Dokumentation und Implementierung

Im folgenden Kapitel wird ein Überblick über die Entwicklung der Implementierung der Software LOSVis (Learning Object Visualization) bis zu dem Stand des ersten Prototyps gegeben.

Vorüberlegungen über vorhandenen Techniken, so wie das Wissen über deren Möglichkeiten sind nötig, um Designentscheidungen zu Treffen und die Strukturierung des Programms vorzunehmen. Hierzu sind die vorangegangenen Kapitel entstanden.
Die Implementierung der Software LOSVis orientiert sich an diesen Überlegungen.

Die Funktionsweise wird anhand von Beispielen verdeutlicht und im Nachfolgenden der wichtigste Code zu den einzelnen Funktionalitäten Schritt für Schritt erklärt.

Die Implementierung von LOSVis lässt sicherlich noch Wünsche offen, denn es entstand erst wie erwähnt ein Prototyp.
Einige dieser nicht umgesetzten Ideen können jedoch als Grundlage für eine Weiterentwicklung dienen, darum wurden auch diese Ideen in einem extra Kapitel festgehalten.

3.1 Vorüberlegungen

Um eine Anwendung zu entwickeln, mit Hilfe derer Suchanfragen nach Relationen von Lernobjekten gestellt werden können und die Ergebnismenge visualisiert ausgibt, werden verschiedenste Techniken benötigt.
Betrachtet man die Menge der unterschiedlichen Techniken, werden Komplexität und Umfang der Anwendungsentwicklung deutlich.

Die Funktionsweise soll wie folgt aussehen:

Es gibt zwei Vorgänge, die nacheinander ablaufen. Zunächst wird der Benutzer einen Suchbegriff eingeben. Dieser wird im LOSVis Client zu einer Anfrage verarbeitet und an den Fedora Server geschickt. Hier wird die Datenbank nach dem Suchbegriff durchforstet. Anschließend wird das Ergebnis in XML-Format an den Client zurückgegeben. Dieser verarbeitet das Ergebnis und gibt es als Liste in der graphischen Oberfläche aus, die der Benutzer nun sehen kann. Vereinfacht kann man dies bildlich so darstellen:

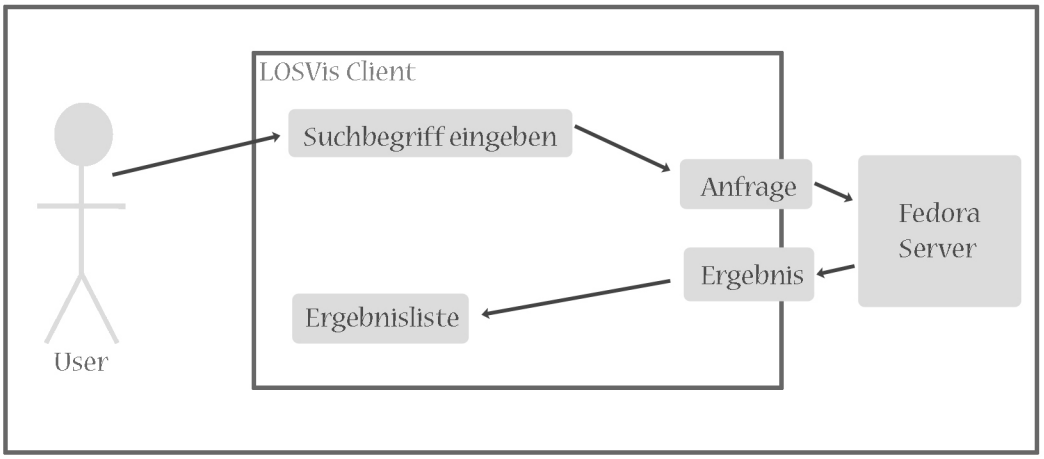

Abbildung 19: Use Case Diagramm 1: Anfrage mit Suchbegriff

Nachdem dies erfolgt ist gibt es für den Benutzer die Möglichkeit ein Ergebnis der Liste auszuwählen, um die Datastreams von diesem Objekt als Graph anzeigen zu lassen. Dies sieht als Use case wie folgt aus:

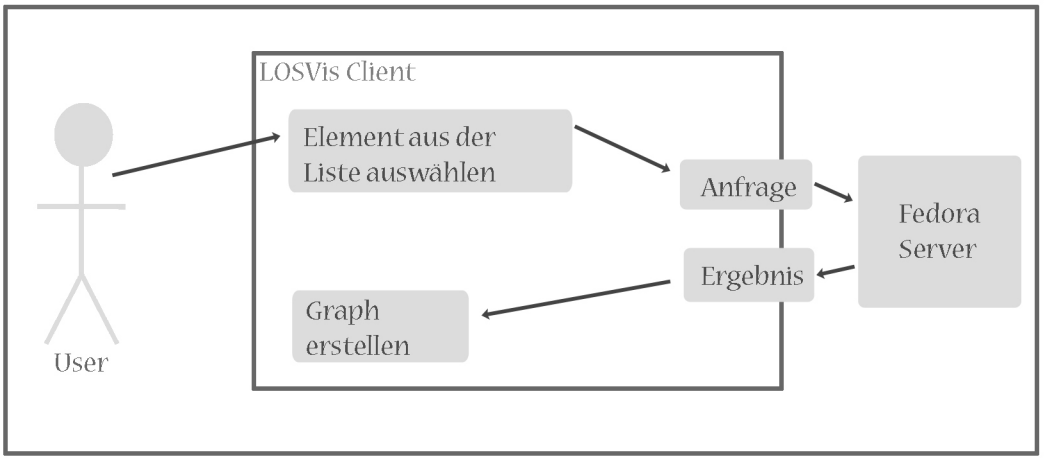

Abbildung 20: Use Case Diagramm 2: Anfrage nach Datastreams

Es fällt auf, dass beide Vorgänge Gemeinsamkeiten aufweisen. In beiden Fällen muss eine Verbindung zu der Datenbank aufgebaut werden. Es werden lediglich unterschiedliche Daten abgefragt. Diese werden auch unterschiedlich ausgewertet. Hierzu verwendet LOSVis verschiedene Funktionen, die im Laufe dieses Kapitels näher erklärt werden.

Mit welchen Hilfsmitteln sollte diese Funktionsweise also umgesetzt werden?

Die Anwendung wurde unter Microsoft Visual .NET 2005 in der Programmiersprache C# entwickelt. Sie besteht aus einer Client-Server Architektur, da die Datenbank (Fedora) von überall, über eine graphische Benutzeroberfläche (Abk. GUI, engl. Graphical User Interface), ansprechbar sein sollte.

Die Kommunikation zu dem Repository geschieht mittels Web Services. Die Anfragesprache ist hierbei fedoraspezifisch.

Die Ergebnisse werden im XML-Format an den Client zurückgeliefert. Dieser muss darum XML verarbeiten können.

Nach Auswahl eines einzelnen Elements wird ein Graph erstellt und mit all seinen Datastreams in einer Baumstruktur angezeigt. Für diese Visualisierung kommt das Open-Source Graphenframework Netron (derzeit Version 2.2.2036) zum Einsatz.
Zusätzlich erscheint in der Graphendarstellung auch die URL (engl. Uniform Resource Locator), mit der das Lernobjekt verlinkt ist und der Benutzer letztendlich an das Lernobjekt herankommt.

3.2 Anforderungsanalyse und Designentscheidung

„Eine Anforderung (Requirement) an ein System, ist die Beschreibung der vom System bereitzustellenden Dienste und Restriktionen (Einschränkungen), die bei der Erstellung des Systems berücksichtigt werden müssen."
[s. 8]

Anforderungen werden unterschieden in „funktionale Anforderungen" und die „nicht-funktionale Anforderungen".

Funktionale Anforderungen:

„Funktionale Anforderungen beschreiben, *was* in einem Softwaresystem gemacht werden soll."
[s. 8]
Funktionale Anforderungen beschreiben die genauen Funktionen, die das System haben muss um nach Fertigstellung abgenommen zu werden, und solche, die als „nice-to-have" eingestuft werden. Diese müssen unbedingt unterschieden werden, um eine Abnahme zu ermöglichen. Im Folgenden werden die Zusätze (nice-to-have) mit NTH abgekürzt.
Die funktionalen Anforderungen in LOSVis sind die Folgenden:

1. Kommunikation des Benutzers mit dem System:

- Der Benutzer kann den Client über eine GUI bedienen.
- Hier kann er eine Anfrage starten. Anfragen bestehen entweder aus der Texteingabe eines Suchbegriffs in die Textbox oder als NTH auch eine Formatsuche über die Eingabe der Dateiendung, um nach einem speziellen Datentyp (zum Beispiel einem Bild) zu suchen.
- Der Benutzer muss ein Element aus der Ergebnisliste auswählen können, um mit diesem speziellen Objekt weiterarbeiten zu können.
- Der Benutzer darf das Ergebnis seines Graphen speichern, drucken, ein- und auszoomen.
- Mit Klicken auf die URL des Datastreams in der entsprechenden Shape muss er zu dem Objekt weitergeleitet werden.

2. Funktionalitäten intern im Client:

- Die Anfragen müssen von dem Client in eine Anfrage für den Server umformuliert werden.
- Der Client verarbeitet die vom Server in XML zurückgegebenen Daten und stellt sie in einer Liste dem Benutzer zur Verfügung.
- Nach Auswahl eines speziellen Ergebnisses aus der Liste muss der Client eine Anfrage nach dessen Datastreams formulieren.
- Die zurückgegebenen Datastreams müssen in Shapes dargestellt werden.

3. Kommunikation des Clients mit dem Server:

- Der Client greift über Webservices auf die Datenbank zu.
- Die Suche nach Lernobjekten und Datastreams erfolgt über die fedoraspezifische Anfrage.
- Über Webservices werden Ergebnisdaten an den Client zurückgeliefert. Die zurückgegebenen Daten sind im XML-Format.

Nichtfunktionale Anforderungen:

„NFRs in Softwaresystemen zeigen, *wie* etwas gemacht wird. Es wird auch häufig von „quality requirements"gesprochen. Nicht funktionale Anforderungen sind die globalen Zwänge in der Softwareerstellung, wie etwa Entwicklungskosten, Betriebskosten, Geschwindigkeit oder auch Verlässlichkeit." *[s. 8]*

Portabilität: Das System muss auf lauffähig sein.

Stabilität: Durch eine weitestgehend geführte Eingabe seitens des Anwenders, werden Fehler bereits im Vorfeld abgefangen. Dadurch sollte das System stabil arbeiten können. Ansonsten sollen aussagekräftige Fehlermeldungen an den Benutzer geschickt werden.

Benutzerfreundlichkeit: Das System wird über eine graphische Benutzeroberfläche zu bedienen sein, die möglichst einfach und intuitiv erscheint.

Wartbarkeit: Das System ist ausführlich dokumentiert und besitzt eine übersichtliche Programmstruktur, die die Wartung erleichtert.

Sicherheit: Die Suchanfrage kann ohne spezielle Nutzerauthentifizierung durchgeführt werden. Der Nutzer sieht dann das Daten für die gestellte Suchanfrage vorhanden sind. Die eigentlichen Daten können jedoch nur mit Zugangsdaten von der Datenbank abgerufen werden.

Skalierbarkeit: Die Funktionsfähigkeit des Systems soll auch bei vielen Daten gewährleistet sein, in diesem Fall wird die Ergebnisliste länger, was für das System kein Problem darstellen sollte.

Benutzerschnittstellen und Fehlerverhalten: Die Bedienung des Systems wird dem Benutzer anhand eines Graphical User Interfaces bereitgestellt. Einstellungen von Daten in die Datenbank werden nicht mit Hilfe dieses Clients erfolgen, lediglich das abrufen der Daten. Das System gibt bei nicht korrekt gestellter Such-Anfrage eine Fehlermeldung an den Client aus.

User Interface Design:

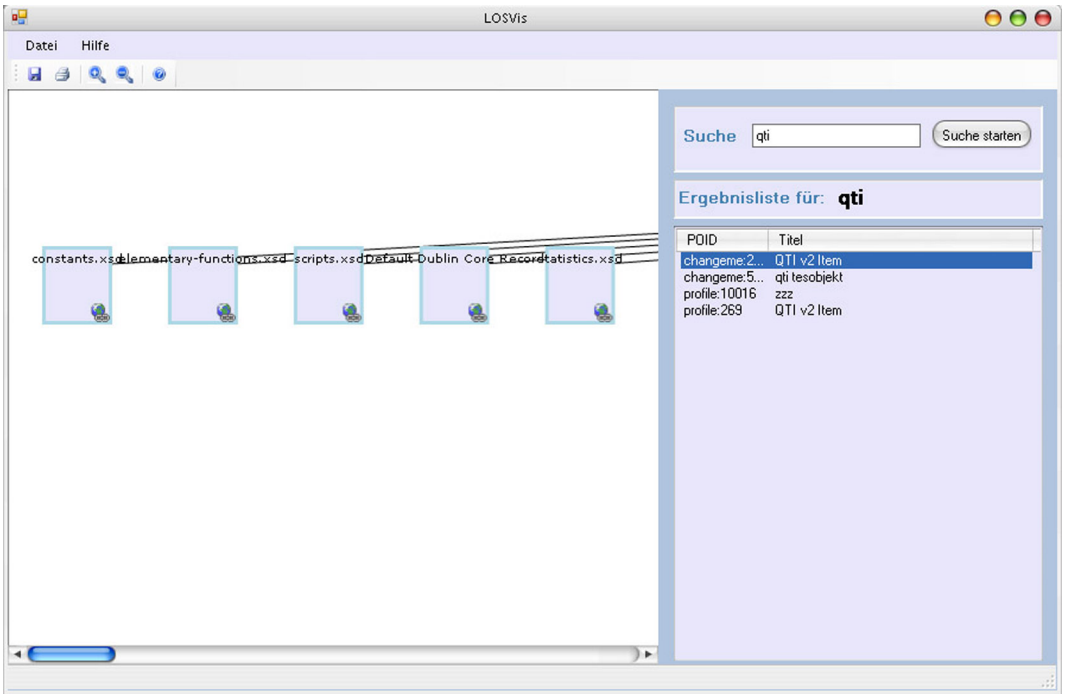

Abbildung 21: LOSVis: Graphische Oberfläche

Die GUI sieht nach einer vollständigen Suche beispielsweise aus wie der vorangegangenen Abbildung.
Die Funktionalitäten sind wie vorher beschrieben eingebaut.

Zu finden sind 2 Menüs, die die programmüblichen Einträge enthalten. Diese können statt durch die Menüs auch mit Hilfe der Buttons oder Shortkeys bedient werden.

Rechts ist sowohl das Eingabefeld für die Stichwortsuche zu finden, als auch die Ergebnisliste darunter, die alle Einträge enthält die entsprechend in der Datenbank gefunden werden konnten.

Auf der linken Seite befindet sich der Graph, der nach dem Auswählen eines Suchergebnisses erzeugt wird.

Sobald die Verbindung mit dem Server aufgenommen wird, erscheint ein Ladebalken in der unten angebrachten Statusleiste, der dem Benutzer eine Rückmeldung über den aktuellen Prozessstatus liefert.

3.3 Anwendungsbeispiele

An einem Beispiel soll die Funktionsweise von LOSVis deutlich gemacht werden. Der Beispielbegriff für die Erläuterung sei hier „qti“.
Zunächst sieht der Startbildschirm wie folgt aus:

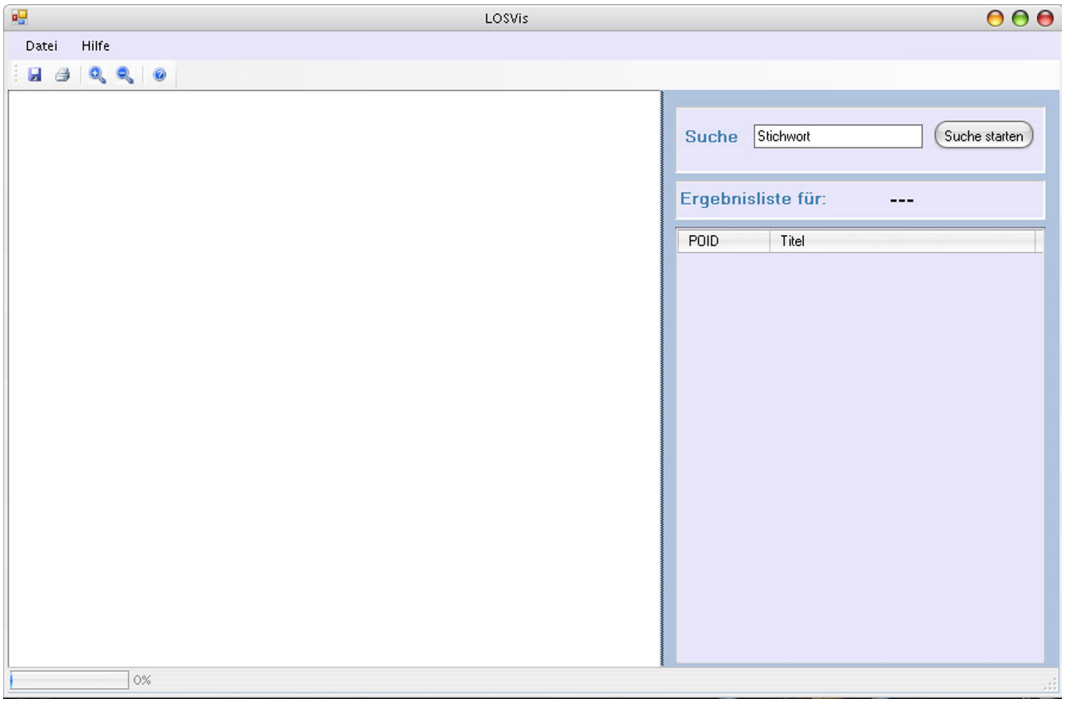

Abbildung 22: LOSVis: Startbildschirm

Hier kann der Benutzer oben rechts einen Suchbegriff eingeben. Tut er dies, in diesem Beispiel also „qti“, beginnt der Suchvorgang, oder es ergibt sich folgendes Bild:

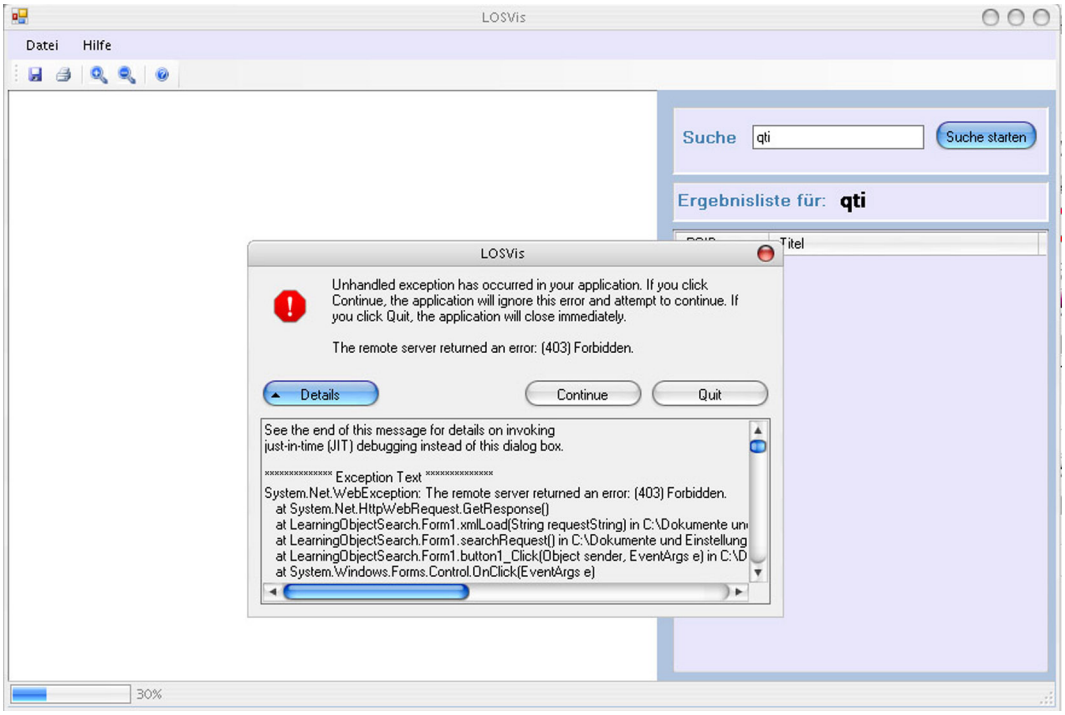

Abbildung 23: LOSVis: Fehlermeldung

Wenn diese Fehlermeldung ausgegeben wird, dann besteht keine Verbindung zum Server. Im Moment ist es notwendig ein VPN (Virtual Private Network) zur Universität Koblenz-Landau Campus Koblenz herzustellen, um mit dem Server Verbindung aufzunehmen.

Nach einer internen Umschreibung innerhalb des Clients von dem Suchbegriff in die fedoraspezifische Anfragesprache, wird mit Hilfe von Web Services nun diese Anfrage an Fedora versendet. Als Ergebnis wird, wieder über einen Web Service, ein XML-String zurückgegeben. Dieses beinhaltet POID und Titel der Objekte, die die Ergebnismenge der Anfrage darstellen.
Der Benutzer bekommt diesen XML-String nicht zu sehen, doch wenn man ihn sich ausgeben lässt, sieht dieser für das Beispiel „qti“ so aus:

```
<?xml version="1.0" encoding="UTF-8"?>
<result xmlns="http://www.fedora.info/definitions/1/0/types/">
<resultList>
  <objectFields>
      <pid>profile:269</pid>
      <title>QTI v2 Item</title>
  </objectFields>
  <objectFields>
      <pid>changeme:230</pid>
      <title>QTI v2 Item</title>
  </objectFields>
  <objectFields>
      <pid>changeme:574</pid>
      <title>qti tesobjekt</title>
  </objectFields>
  <objectFields>
      <pid>profile:10016</pid>
      <title>zzz</title>
  </objectFields>

</resultList>
</result>
```

Abbildung 24: LOSVis Konsolenausgabe: xml string

Durch das Parsen des XML-Strings innerhalb des Clients, wird Dieser ausgelesen und dem Benutzer in einer Liste innerhalb der GUI ausgegeben.

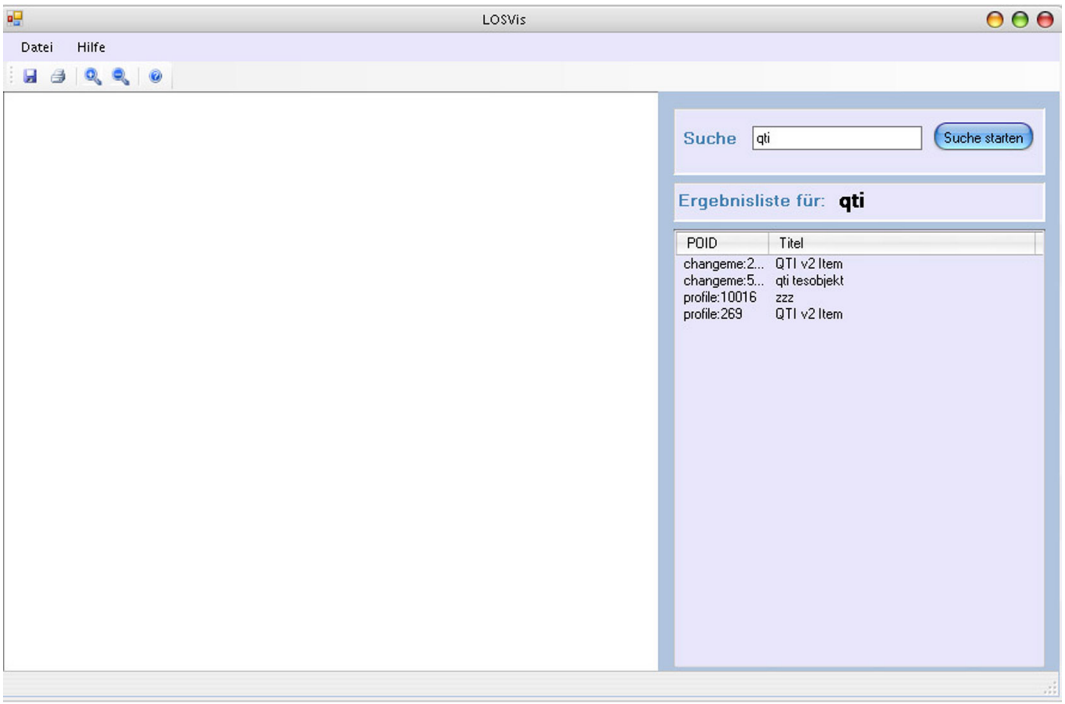

Abbildung 25: LOSVis: Ergebnisliste

Der Benutzer sieht nun alle Ergebnisse in einer Liste, die zu seiner Anfrage passen. Er hat nun die Möglichkeit ein spezielles Ergebnis auszuwählen. Durch Klicken auf eine Zeile der Liste wird dieses Objekt ausgewählt und in einer Shape auf der linken Seite des Programms angezeigt.
Zusätzlich werden zu dem ausgewählten Objekt all seine Datastreams abgerufen.
Datastreams gehören zu Fedora Objekten und beinhalten Texte, Grafiken etc, alles was ein Objekt ausmachen kann.

Ein Beispiel hierfür wäre eine Powerpoint Datei, die mehrere Folien enthält. Grafiken und Töne können wiederum Inhalt dieser Folien sein. All diese Inhalte werden in Datastreams referenziert.

Diese Datastreams bekommt man wieder über eine Anfrage, die der Client über einen Web Service an Fedora versendet und die POID des gewählten Objekts übergibt, um die entsprechenden Datastreams in einem XML-String zurückzubekommen.

Bei einer Ausgabe dieses Strings auf der Konsole sieht das dann wie folgt aus:

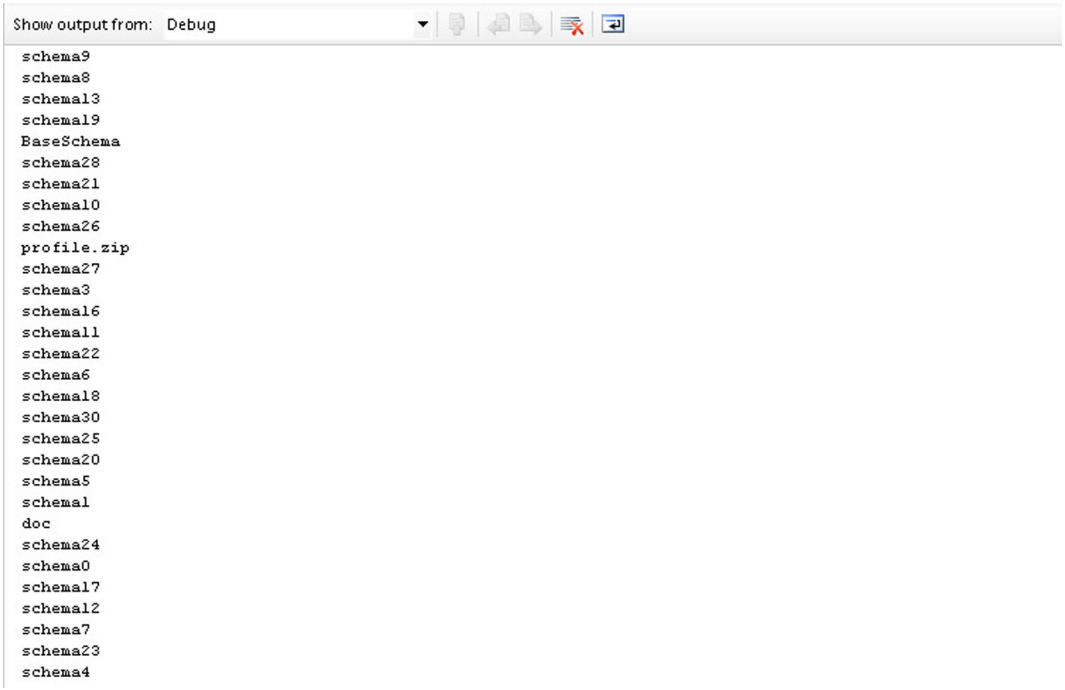

Abbildung 26: LOSVis Konsolenausgabe: Xml von Datastreams

Dies ist nur ein Auszug, welche Ids zum Beispiel bei einer Anfrage nach Datastreams zurückkommen können.

Es gibt keine Begrenzung der Anzahl der Datastreams. Daher kann man vorher nicht wissen, ob das Objekt einen oder 50 Datastreams beinhaltet und wie der Graph hinterher anzuordnen ist. Das heißt es müssen erst alle Datastreams ausgelesen werden und erst dann kann eine sinnvolle Anordnung erzeugt werden.

Die Labels der ausgelesenen Datastreams werden jeweils in eine extra Shape innerhalb des linken Bereichs von LOSVis geschrieben. Abbildung 27 zeigt den Auslesevorgang, bei dem noch keine Anordnung stattfindet, der Beispielanfrage.

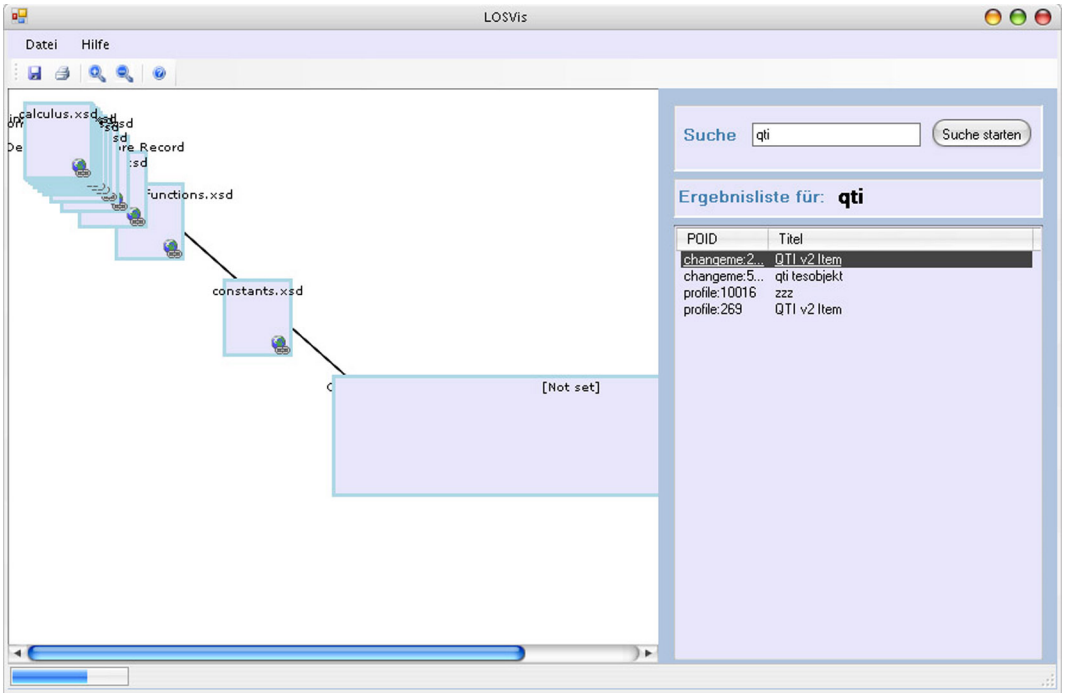

Abbildung 27: LOSVis: Graphenerstellung

Nachdem alle Datastreams ausgelesen wurden, werden die Shapes graphisch als Baumstruktur an das Objekt angehangen. Das kann unterschiedlich aussehen, je nachdem wie viele Datastreams an einem Objekt anhängen.

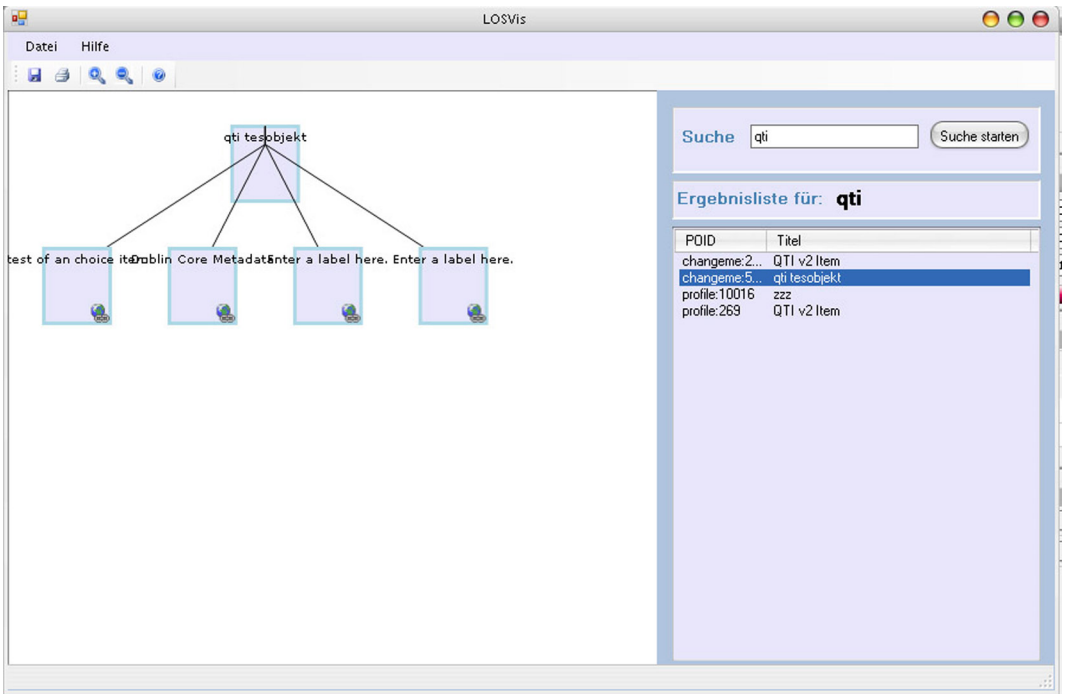

Abbildung 28: LOSVis: Graphenerstellung nach Layoutalgorithmus

In diesem Fall ist das Ganze recht übersichtlich, da es nur vier Datastreams sind, die angezeigt werden. Es passt alles in das Fenster und lesbar ist in der Schriftgröße auch.

Es kann allerdings passieren, dass es sehr viele Datastreams als Ergebnis gibt, dann erhält man folgendes Bild:

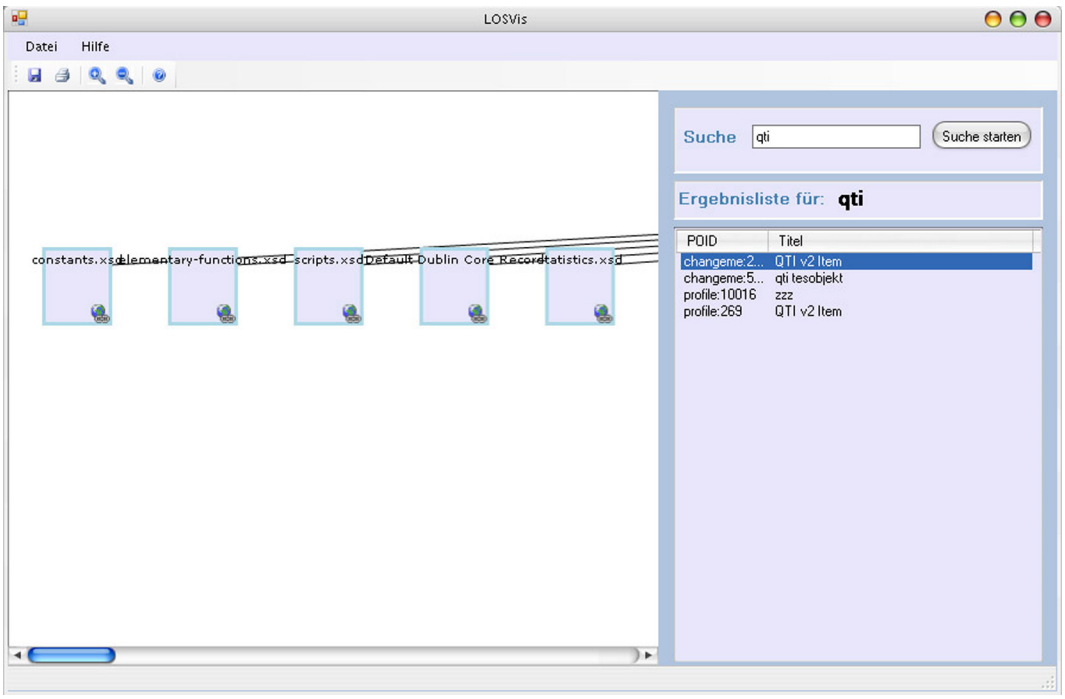

Abbildung 29: LOSVis: Graphenerstellung nach Layoutalgorithmus

Der Benutzer sieht also nur einen Teil des Ganzen, und die untere Scrollbar erscheint.
Da der Nutzer so nicht den Überblick hat, kann er zum Beispiel heraus- und hereinzoomen, indem er die Buttons in der Leiste oberhalb des Graphen nutzt oder über die beiden Menüs auf die Funktionen zugreift.

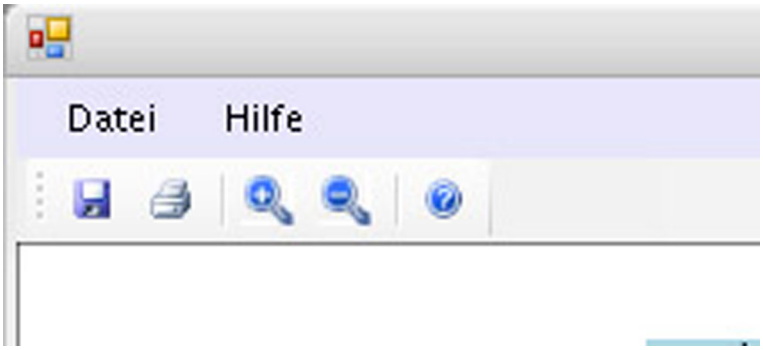

Abbildung 30: LOSVis: Menüleisten

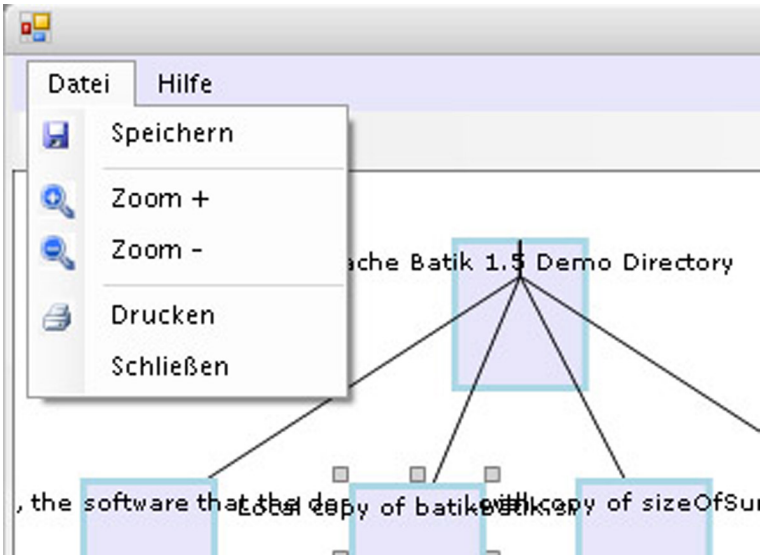

Abbildung 31: LOSVis: Dateimenü

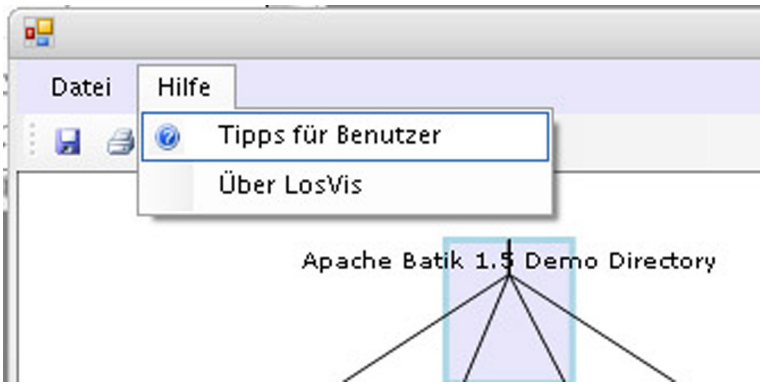

Abbildung 32: LOSVis: Hilfemenü

In der Leiste oder im Menü „Datei“ kann der Benutzer die Ansicht ein- oder auszoomen, speichern oder drucken.
Das Zoomen ermöglicht es dem Benutzer einen Überblick zu bekommen (siehe Abbildung 33).

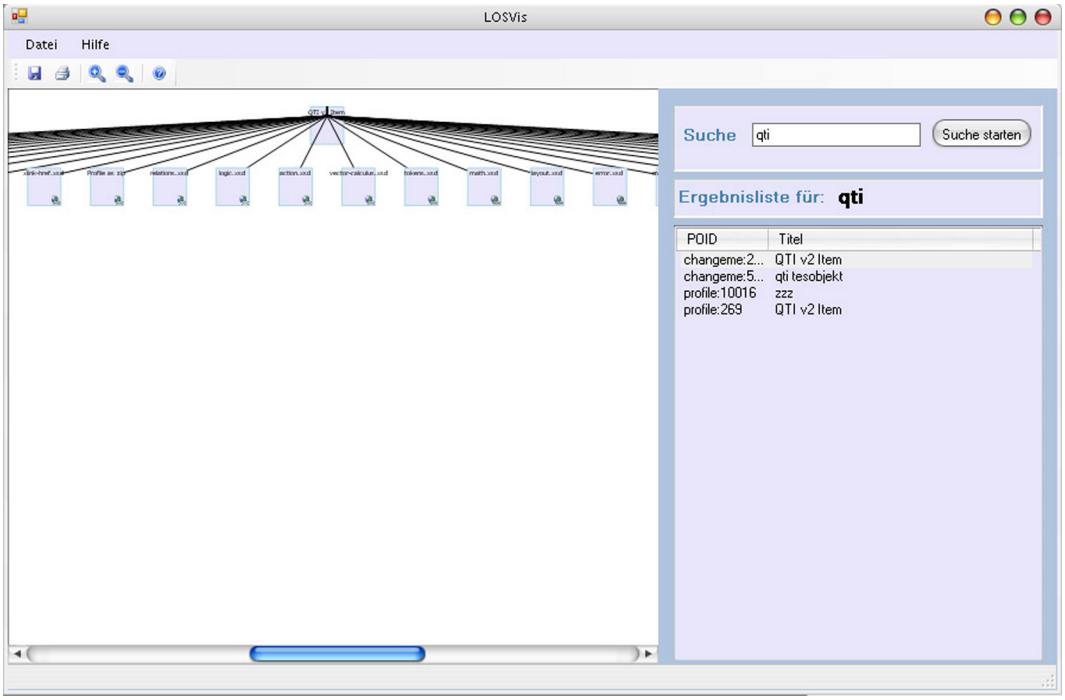

Abbildung 33: LOSVis: Auszoomen

Das Einzoomen hingegen macht einzelne Teile besser lesbar. (siehe nachfolgende Abbildung).

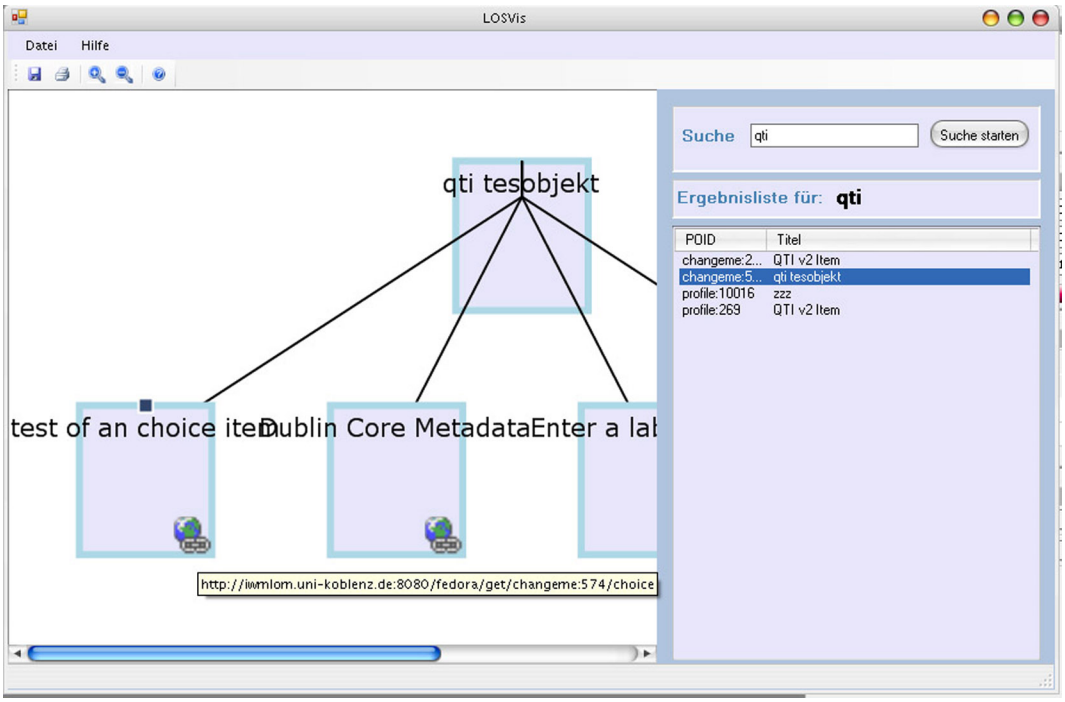

Abbildung 34: LOSVis: Einzoomen

Zusätzlich kann der Anwender innerhalb eines Datastreams auf das Icon klicken, das über die URL direkt zu dem Datastream leitet. Der Browser wird geöffnet und eine Authentifizierung verlangt.

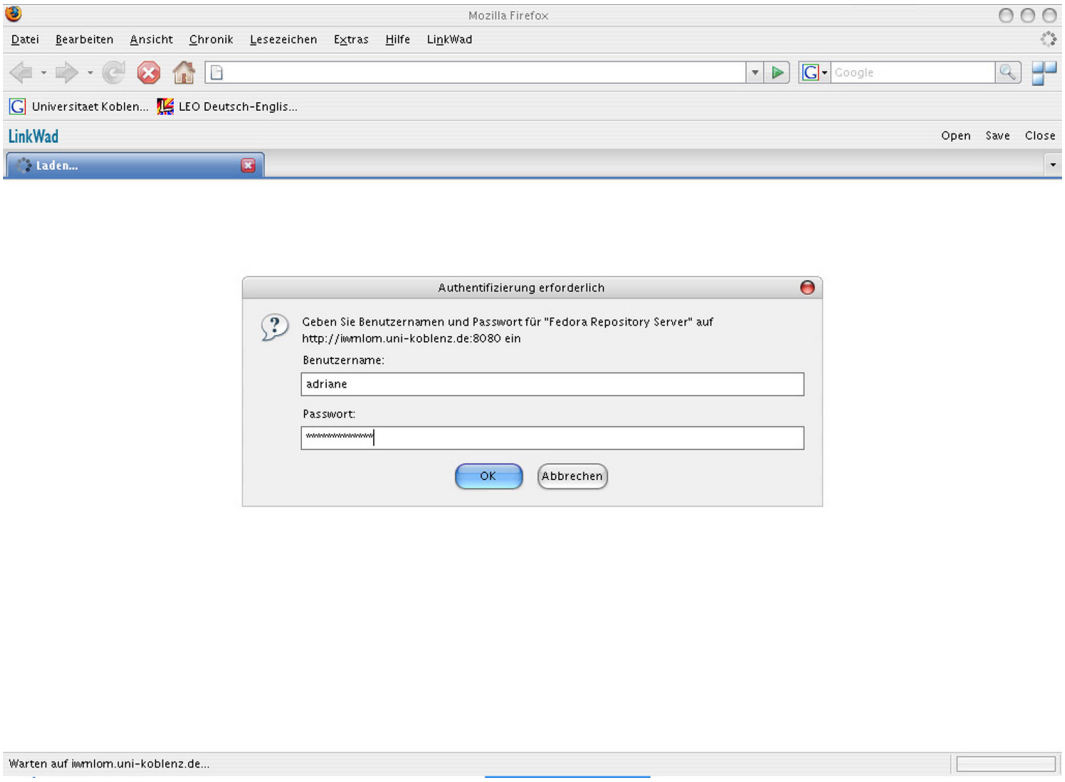

Abbildung 35: LOSVis: Autorisierungsanfrage

Über diese Weiterleitung mit Hilfe des WebBrowsers kommt jeder berechtigte User an den Inhalt der Datastreams. Nicht-autorisierte Benutzer können nur die Suchfunktion nutzen und sehen, dass Ergebnisse vorliegen, dürfen diese jedoch nicht herunterladen.

Die Datastreams können zum Beispiel ein Link zu einer Website sein, ein Bild, XML-files, gezippte Dateien und alle möglichen anderen Inhalte, die in dem Repository liegen oder verlinkt sind. Nach der Suche mit dem Suchbegriff „demo“ kann man folgende Situation betrachten. Zunächst wird eine Liste von zwei Ergebnissen zurückgegeben. Wählt man das zweite Element der Liste, erhält man den folgenden Graphen:

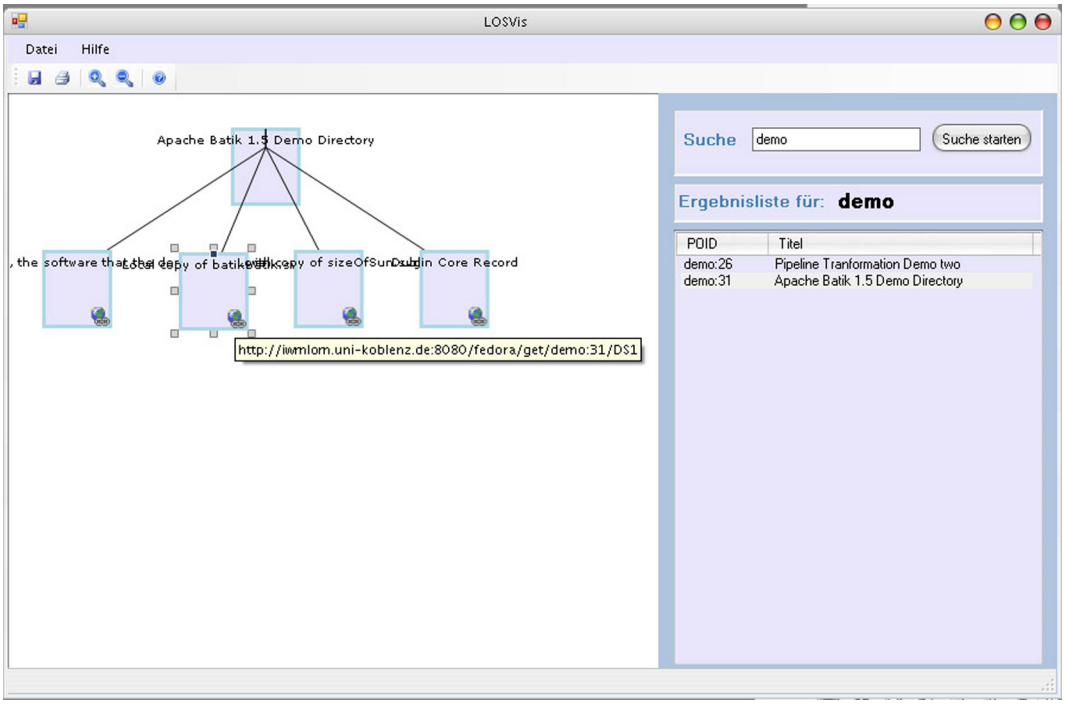

Abbildung 36: LOSVis: URL von Datastreams

Hier sieht man, wie die URL aufgebaut ist, und dass sie beim Mouse-over angezeigt wird. Wird nun das URL-Symbol in der ersten Shape angeklickt kann man im Browser folgendes sehen:

Abbildung 37: Referenz auf Homepage

Der Datastream beinhaltete also einen Link zu dieser Webseite, auf diese der Benutzer geleitet wird.
Die URL der zweiten Shape leitet auf ein Bild. Auch dieses öffnet sich nach der erfolgreichen Authentifizierung im Browser wie folgt:

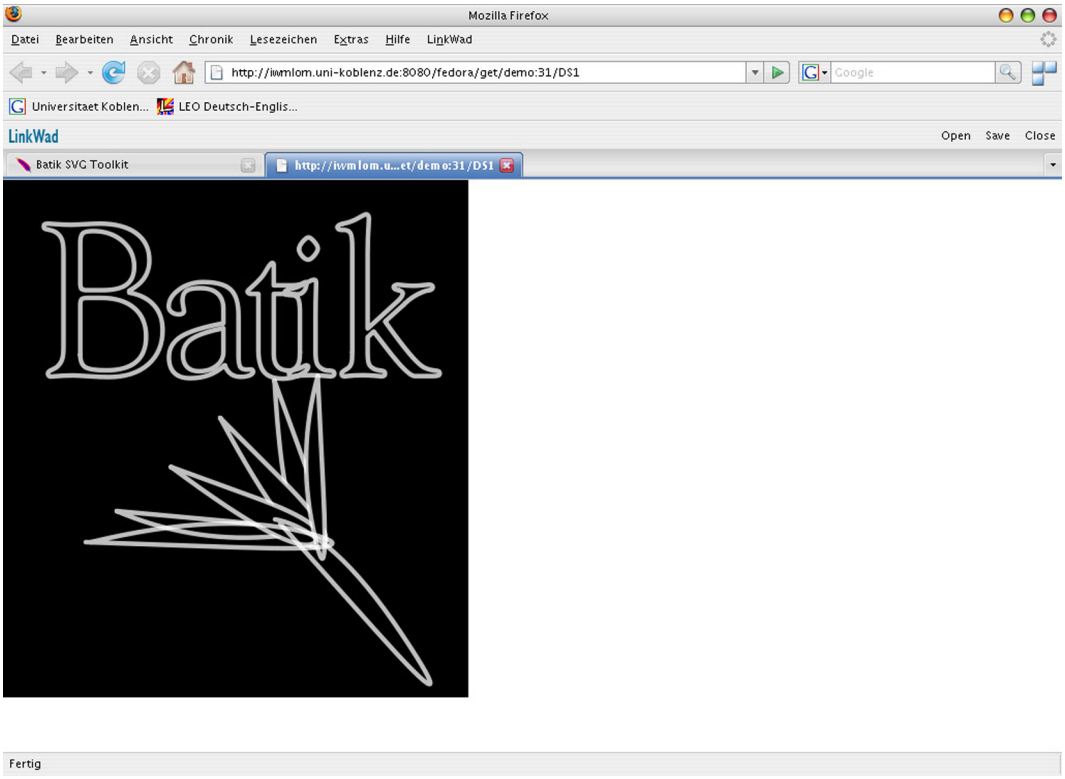

Abbildung 38: Referenz auf ein Bild

Nachdem der Benutzer zu dem eigentlichen Inhalt gelangt ist, kann er diesen speichern (bei einem Bild zum Beispiel) oder anschauen (zum Beispiel bei einer Webseite).

Will der Benutzer nach einem anderen Suchbegriff suchen, muss er diesen nur wieder in das Suchfeld eingeben. Die Liste und der Graph des vorherigen Ergebnisses werden gelöscht und entsprechend die neuen Werte ausgegeben.

3.4 Struktur und Implementierung von LOSVis

Damit die Software LOSVis wie oben beschrieben funktioniert bedarf es einzelner Projekte, die zusammenspielen müssen.
Dazu gehören die verschiedenen Techniken, die im Einführungskapitel erwähnt wurden. Das ganze System besteht aus den folgenden Elementen:

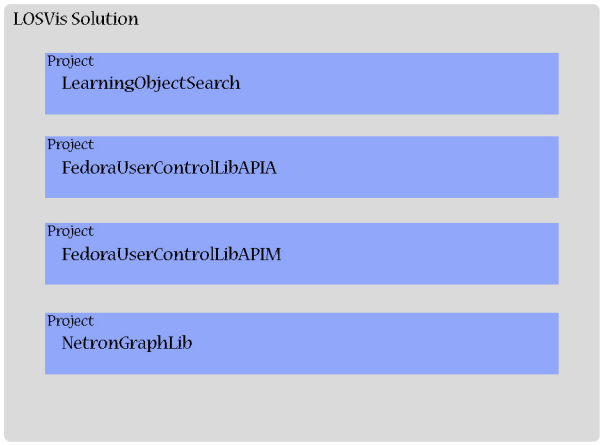

Abbildung 39: Inhalte LOSVis Solution

„LearningObjectSearch“ ist das Projekt, das im Laufe dieser Arbeit entstanden ist. Die Anderen wurden lediglich an verschiedenen Stellen angepasst, so dass sie an der entsprechenden Stelle angemessener ihren Zweck erfüllen.

LearningObjectSearch sollte unterteilt werden in eine Suchklasse, eine XML-Parsen-Klasse, eine GUI-Klasse, eine Shapes-Anlegen-Klasse, eine WebService-Klasse, die die Datastreams ausliest und eine Helper-Klasse für den WebService.
Im Laufe der Arbeit stellte sich jedoch heraus, dass zuviele Werte an jede Klasse übergeben werden mussten und es für den Zweck zu kompliziert wurde. Darum beschränkte sich das Ganze auf die Klassen, die in der nachfolgenden Grafik zu sehen sind.

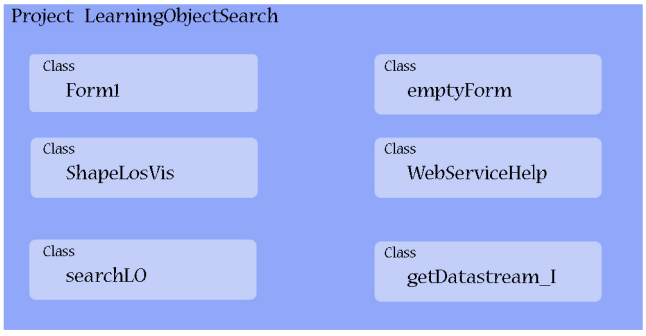

Abbildung 40: Inhalte Projekt "LearningObjectSearch"

Um erst einmal einen groben Überblick zu bekommen, wieso diese Klassen existieren und welchen Zweck sie erfüllen, hier erst mal eine grobe Darstellung des Systems.
In „Form1“ befinden sich alle GUI-Elemente und alle wichtigen Funktionen, die Hilfsfunktionen und Klassen hieraus aufrufen.

„ShapeLOSVis“ ist eine solche Hilfsklasse. Sie beschreibt die Shapes, wie sie nachher im Graphen dargestellt werden.

„searchLO“ sollte ursprünglich alle Suchfunktionen beinhalten, wurde aber dann nur noch zu Testzwecken eingesetzt und im Status, wie der Prototyp zum Abschluss dieser Arbeit läuft, nicht verwendet.

„emptyForm“ beschreibt eine Helferklasse, die lediglich eine leere Windows-Form beinhaltet. Wird der Aufruf eines neuen Fensters benötigt, so erzeugt man ein Objekt dieser Klasse und kann die Daten in dieses neue Fenster schreiben.

Die Klassen „WebServiceHelp“ und „getDatastream_I“ sind aus dem Projekt FedoraUserControlLibAPIM kopiert und angepasst worden.
„WebServiceHelp“ wird benötigt um eine Verbindung mit dem Repository aufzunehmen und die Authentifizierung vorzunhemen.
„getDatastream_I“ ist auch eine Testklasse, um auszuprobieren wie Datastreams aus dem Repository abgerufen werden können.

LOSVis funktioniert durch das Zusammenspiel der einzelnen Klassen. Wichtige Teile der Klasseninhalte werden im Nachfolgenden im Zusammenhang genauer erläutert.

Vorweg noch ein Zusatz zu der Klasse „searchLO“.
Die Suchfunktionen, die letztendlich im Einsatz sind, finden sich in der Klasse „Form1“.
Dennoch sind die 3 Funktionen dieser Klasse nutzbar, dienten aber letztendlich nur zu Testzwecken. Sie sollen kurz erläutert werden, da sie zu weiteren Testzwecken gebraucht werden können oder bei einer Weiterentwicklung Anwendung finden könnten.

public void searchForLob(): die Möglichkeit nach Datastreams, mit Hilfe von ControlHosts, die bereits in der FedoraUserControlLibAPIA vorhanden waren, zu suchen.

public void searchForLobByAPIA(string searchString): die Möglichkeit über die FedoraUserControlLibAPIA mit Hilfe eines Suchbegriffes direkt nach POID und ähnlichem zu suchen.

public void searchForLobByRESTProtocol(): die Möglichkeit nach Relationen zwischen Lernobjekten zu suchen (Diese Funktion kam nie zum Einsatz, sie müsste noch umgeschrieben und angepasst werden. Sie greift im Moment auf die ControlHosts der FedoraUserControlLibAPIA zu und kann so noch nicht in LOSVis eingebaut werden, ist aber für eine Weiterentwicklung als Testfunktion dienlich.)

Die wichtigsten Inhalte von LOSVis befinden sich in der Klasse „Form1“. Von Dieser ausgehend werden auch an entsprechender Stelle die anderen Klassen erläutert.

Was beinhaltet „Form1“ also nun wirklich?

Ruft der Anwender das Programm auf, so erscheint die GUI, die sich in dieser Klasse befindet. Hier findet sich also die komplette GUI-Programmierung. Das Auslagern einzelner Elemente wurde komplizierter als zunächst angenommen und somit wurden auch die Suchfunktion, das XML-Parsen und das Anlegen der Shapes wieder mit in die Klasse „Form1“ integriert.
Hier ein Überblick über diese Klasse:

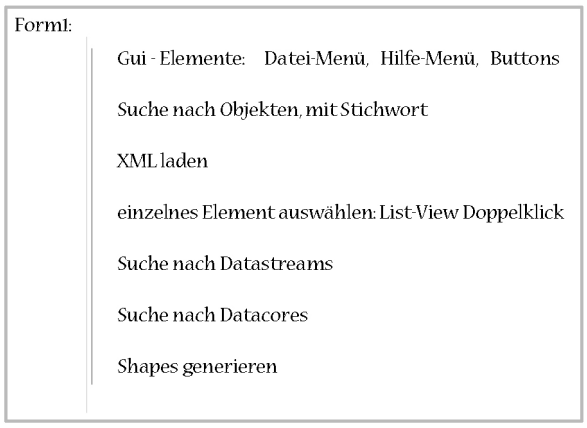

Abbildung 41: Inhalt Klasse „Form1“

Diese Funktionalitäten der Abbildung 41 sind im Laufe der Arbeit entstanden. Für die Visualisierung, also um die Shapes am Schluss zu generieren, wird die bereits vorgegebene „Netron Graphen Bibliothek“ angesprochen. Erst einmal ein paar erklärende Worte zu den einzelnen Funktionen, wie sie oben dargestellt sind.

Die Menüs und die Buttons enthalten die programmüblichen Funktionen. Dazu gehören die Möglichkeiten, einen Graphen nach der Erstellung ein- oder auszuzoomen, damit der Benutzer alles von der Größe her erkennen und lesen kann, den Graphen abzuspeichern (als .jpg) oder auszudrucken.
Alle Menüs sind ohne Schwierigkeiten um weitere Funktionalitäten erweiterbar.

Die Funktion die den eingegebenen Suchbegriff verarbeitet nennt sich „searchRequest()“.
Am Anfang ist es notwendig, dass bei jeder neuen Suche sowohl die Liste als auch der Graph gelöscht werden, falls die Suche nicht die erste ist, sondern in der Ergebnisliste und im Graphen schon Einträge vorhanden sind. Dies geschieht mit folgenden Befehlen:

```
listView1.Items.Clear();
this.graphControl1.NewDiagram(true);
```

Ist das Programm nun auf jeden Fall wieder auf den Ausgangswert zurückgesetzt, kann die eigentliche Suche beginnen.

```
if (textBox1.Text == "" | textBox1.Text == "Stichwort")
{      MessageBox.Show("bitte Suchbegriff eingeben");
}
else
{  searchterm.Text = textBox1.Text;
   searchString = fedoraServerUrl;
   searchString = searchString + "terms=" + textBox1.Text + "&xml=true" + "&pid=true"
        + "&title=true";
   xmlLoad(searchString);
}
```

Dazu wird die Texteingabe abgefragt. Wenn die Textbox leer oder immer noch mit dem Startwert „Stichwort"gefüllt ist, bekommt der Benutzer mitgeteilt, dass er einen Suchbegriff eingeben soll.
Hat er einen Begriff eingegeben wird der searchString, also der String für die Suchanfrage, an die Datenbank zusammengesetzt. Dieser enthält die URL des Fedora Servers, den eingegebenen Suchterm und eventuell weitere Optionen.
Die hier angegebenen Optionen sind noch nicht alle die möglich sind. Diese werden später in dem Abschnitt für Erweiterungen noch genauer beschrieben.
In diesem Fall sagen die Angegebenen aus, dass die Ergebnisrückgabe von Fedora in XML Format angegeben wird, und dass die PersonalID (pid) und der Titel zurückgegeben werden.

Wenn dieser SearchString zusammengesetzt ist, wird dieser der Funktion xmlLoad übergeben, die daraufhin aufgerufen wird.

In dieser Funktion wird zunächst die Authentifizierung veranlasst, in dem ein Objekt der Klasse „Authentification“ der FedoraUserControlLibAPIA erzeugt wird.

```
Authentication auth = new Authentication();
auth.Authenticate();
```

Der Zugang wird hier noch über den Benutzer „adriane“ und das Password „studienarbeit“ geregelt, was in Zukunft noch geändert werden muss.

Als nächstes wird in der xmlLoad die Verbindung geprüft. Kann diese nicht hergestellt werden, so soll eine Fehlermeldung erscheinen.

```
HttpWebRequest myRequest = (HttpWebRequest)HttpWebRequest.Create(searchString);
 myRequest.Method = "GET";
 myRequest.Credentials = auth.myCredentials.GetCredential(
                          new Uri("http://iwmlom.uni-koblenz.de:8080/fedora"),"Basic");
HttpWebResponse myResponse = (HttpWebResponse)myRequest.GetResponse();
```

Mit Hilfe der Klasse HttpWebRequest wird ein Objekt erzeugt, das die Anfrage an Fedora durchführt. Die Authentifizierung wird über das vorher angelegte Objekt „auth“ geregelt.
In der Variable „myResponse“ befindet sich dann das Ergebnis der Anfrage.
Damit das Ganze nun lesbar wird, encodiert der StreamReader den zurückgegebenen Stream. Dieser wird in einem XML Dokument abgespeichert, wie im folgenden Code zu sehen ist.

```
StreamReader reader =
        new StreamReader(myResponse.GetResponseStream(), Encoding.UTF8);

        resultXMLString = reader.ReadToEnd();
        XmlDocument xmlResponse = new XmlDocument();
        xmlResponse.LoadXml(resultXMLString);
        XmlElement root = xmlResponse.DocumentElement;
```

Nun existiert ein XML Tree statt einem String. Dieser wird nun ausgelesen, um in die Ergebnisliste eingetragen zu werden.

```
for (int itemcount = 0; itemcount < objCount; itemcount++)
{
  ListViewItem itemPOID = new ListViewItem(resultList.ChildNodes.Item(
                                                  itemcount).ChildNodes.Item(0).InnerText, 0);
  string itemTitle = resultList.ChildNodes.Item(itemcount).ChildNodes.Item(1).InnerText;
  itemPOID.SubItems.Add(itemTitle);
  this.listView1.Items.Add(itemPOID);
}
```

Die Ergbeniselemente wurden vorher gezählt, so dass diese jetzt durchgegangen und in die Liste mit POID (Personal Object Identifier) und Titel eingetragen werden können.
Falls die Ergebnisliste allerdings leer war, so bekommt der Benutzer eine Pop-up-Box angezeigt:

```
 else
 {  MessageBox.Show("Leider wurde kein Ergebnis für Ihre Anfrage gefunden");
 }
```

An dieser Stelle muss der Benutzer entweder eine neue Anfrage stellen oder die Ergebnisliste ist gefüllt und er kann sich nun sein Ergebnis heraussuchen, welches er näher betrachten möchte.

Nach einem Doppelklick auf das ausgesuchte Ergebnis wird ein neuer Graph angelegt und das ausgesuchte Objekt als Wurzelshape in den Graphen geschrieben. Die Shape des Wurzelknoten ist ein Objekt der Klasse shapeLOSVis. In dieser ist festgelegt wie die Shapes aussehen sollen und welche Eigenschaften sie besitzen.

```
public shapeLosVis() :base()
    {  Rectangle = new RectangleF(0, 0, 400, 100);
       knotenpkt = new Connector(this, "Verbindungsknoten", true);
       knotenpkt.ConnectorLocation = ConnectorLocation.South;
       Connectors.Add(knotenpkt);
       IsResizable = true;
    }

public override void Paint(Graphics g)
    {  Pen.Color = Color.LightBlue;
       Pen.Width = 3.0f;
       base.Paint(g);
       g.FillRectangle(Brushes.Lavender, Rectangle.X, Rectangle.Y, Rectangle.Width,
                                                                   Rectangle.Height);
       g.DrawRectangle(Pen, Rectangle.X, Rectangle.Y, Rectangle.Width, Rectangle.Height);
       StringFormat sf = new StringFormat();
       sf.Alignment = StringAlignment.Center;
       g.DrawString(this.Text, Font, TextBrush, Rectangle.X + (Rectangle.Width / 2),
                                                                   Rectangle.Y + 3, sf);
    }
```

Festgelegt wird, dass die Shape ein Rechteck in einer definierbaren Größe sein soll. Außerdem wird entschieden wo der Verbindungspunkt oder eventuell mehrere Verbindungspunkte angebracht sind und ob das Rechteck, wenn es erzeugt wurde, durch den Benutzer größenveränderbar ist.
In der Paint-Funktion wird Farbe, Breite, Format und ähnliches festgelegt.

In die erzeugte Wurzelshape wird also der Titel des ausgewählten Objekts geschrieben. Nun sollen alle zu dem Objekt gehörigen Datastreams ausgelesen und an die Wurzel angehangen werden.

Um die Datastreams auszulesen wird zunächst über die WebServiceHelper Klasse die Verbindung hergestellt.
Danach wird über die Web Reference „info.fedora.www“ mit Hilfe der Funktion getDatastreams die Datastreams ausgelesen. Hierzu werden der Funktion drei Werte übergeben. Einmal die POID des selektierten Objekts aus der Ergebnisliste, eine „DateTime“, die auf „0“ gesetzt wird, so dass einfach alles angezeigt wird (egal welches Datum) und zuletzt die Art des Teils des Fedora Objekts, was zurückgegeben werden soll. In diesem Fall ist dieser Wert „A“. Damit erhält man komplett alle Datastreams.

Um zum Beispiel nur die Datacores auszulesen, kann an dieser Stelle „DC“ stehen. Wie das dann aussieht, ist in der Funktion „findDatacores“ zu finden. Diese wird zur Zeit nicht verwendet, ist aber nutzbar und kann bei Bedarf eingesetzt werden. Sie war letztendlich nur eine Vorstufe zur Suche nach den Datastreams.

Nachdem nun alle Datastreams ausgelesen sind und diese nun an das Wurzelobjekt angehangen werden sollen, müssen für jeden einzelnen Datastream Shapes angelegt und mit dem jeweiligen Titel versehen werden.
Dies geschieht in der Funktion „createNetronNodes“. Dieser werden die Datastreams, die POID und die Wurzelshape übergeben, so dass alle Datastreams später an diese Wurzelshape angehangen werden können.
Der Code dazu sieht folgendermaßen aus:

```
foreach (Datastream ds in dataStreams)
{  Datastream dsObj = helper.FedoraService.getDatastream(poid, ds.ID, "0");
   string URL = @"http://iwmlom.uni-koblenz.de:8080/fedora/get/" + poid + "/" + ds.ID;

   shapeLosVis testshape = new shapeLosVis();
   this.graphControl1.AddShape(testshape);
   this.graphControl1.AddConnection(wurzelshape.Connectors[0],
                                                        testshape.Connectors[0]);

   //ab hier der Algorithmus
   testshape.X = graphControl1.Width  / ((shapesCounter + 1) * 3);
   testshape.Y = graphControl1.Height  / ((shapesCounter + 1) * 3);
   //bis hier der Algorithmus

   testshape.FitSize(true);
   testshape.IsResizable = true;
   testshape.Text = dsObj.label;
   testshape.URL = URL;
}
```

Die Datastreams werden einzeln jeweils über die POID der Wurzelshape und die Datastream ID (ds.ID) abgerufen. Die URL, die hinterher in der Shape stehen soll und den Zugriff auf das Objekt für den Benutzer ermöglicht, wird in dem String „URL“ zusammengesetzt. Sie besteht aus der Serveradresse (http://iwmlom.uni-koblenz.de), dem Port (8080), der Datenbank (fedora), dem Aufrufer (get), der POID des Wurzelobjektes (poid) und der ID des Datastreams (ds.ID).

Als nächstes wird eine Shape angelegt und diese dem Graphen hinzugefügt. Zusätzlich bekommt er noch Konnektoren.
An der Stelle, wo „Ab hier der Algorithmus“ steht, wird im Moment jede neu generierte Shape ein Stück weiter nach links oben gesetzt, so dass sie nicht alle übereinander liegend erzeugt werden. Im nächsten Codeabschnitt kann man dann sehen, dass noch ein automatischer Anordnungsalgorithmus auf alle vorhandenen Shapes angewandt wird.

Statt des automatischen Algorithmuses sollte in Zukunft jedoch besser ein Eigener verwendet werden, der besser auf die Ergebnismenge eingehen kann.
Zuletzt werden den Eigenschaften des angelegten Shape-Objekts noch Werte zugewiesen. Dazu gehören zum Beispiel, dass der Text der Shape dem Datastream Label entsprechen soll und dass die URL der Shape dem vorher angelegten String „URL“ entspricht.

Wenn die Shapes komplett mit Inhalt gefüllt und erzeugt sind, wird nun der schon erwähnte automatische Layout Algorithmus von Netron aufgerufen.

```
this.graphControl1.SelectAll(true);
this.graphControl1.StartLayout();
this.graphControl1.Deselect();
this.graphControl1.Refresh();
```

Hierzu müssen zunächst alle Shapes selektiert werden. Dann wird der eingestellte Algorithmus auf diese angewandt. Danach werden sie deaktiviert, denn der Benutzer hat ja eigentlich vorher nichts ausgewählt und soll dadurch nicht irritiert werden, dass nun alle Shapes ausgewählt sind. Als letztes folgt das notwendige „refresh“ des Graphen, also die Aktualisierung der Darstellung, damit der Benutzer auch nun die eigentliche Anordnung zu sehen bekommt.

An dieser Stelle ist die Software einmal komplett durchgelaufen, der Benutzer kann eine neue Anfrage stellen und wieder von vorne beginnen. Alternativ klickt er auf eine URL in einer der Shapes, die als Ergebnis dargestellt wurden, um zu dem eigentlichen Objekt weitergeleitet zu werden.

Kleinigkeiten, die auf die Benutzerfreundlichkeit ausgerichtet sind, wurden zusätzlich entwickelt. Hierzu gehört der Ladebalken, der nur dann zu sehen ist, wenn ein Ladevorgang aktiv ist.
Dieser wird wie folgt programmiert:

```
this.toolStripProgressBar1.Value = 0;
this.prozentLabel2.Text = this.toolStripProgressBar1.Value + "%";
```

Mit diesem Code wird der Wert der ProgressBar, also dem Ladebalken, gesetzt. Außerdem werden dem Label, was den Wert in Prozent neben dem Ladebalken angibt, Werte hinzugefügt.

Zusätzlich gibt es die Möglichkeit die einzelnen Bereiche mittels Tab-Stop durchzuschalten und das Programm mit Shortcuts zu bedienen.
Die Icons wurden so ausgewählt, dass die Buttons intuitiv zu bedienen sind.

3.5 Ausblicke zur Weiterentwicklung

Wie schon in der Einleitung erwähnt gibt es noch einige Erweiterungen, die in dem Prototyp aufgrund von Zeitmangel nicht mehr eingebaut werden konnten. Dennoch möchte ich die Ideen an dieser Stelle festhalten um eventuell ein paar Grundsteine für eine Weiterentwicklung zu legen.

Im Moment ist es notwendig eine VPN-Verbindung zur Universität Koblenz-Landau Campus Koblenz herzustellen, damit LOSVis auf das Repository zugreifen und die Suche verwendet werden kann. Allerdings hat nicht jeder Anwender VPN auf seinem Computer eingerichtet. Je nach Nutzerumfeld besitzt nicht jeder eine Computerkennung der Universität Koblenz-Landau. Dies sollte sicherlich noch geändert werden.

So wie das eingegebene Stichwort bei der Suche im Augenblick in der Datenbank gesucht wird, und die POID und der Titel zurückgegeben werden, können jedoch auch noch andere Dinge wie „Autor“, „Beschreibung“, „Datum“ und ähnliches zurückgegeben werden. Hier ist eine Erweiterung ohne größere Schwierigkeiten machbar, indem der „searchstring“ um die Optionen erweitert wird. Die Liste müsste natürlich angepasst werden und mehr Spalten enthalten.

Bei der Ergebnisausgabe erhält der Benutzer die Datastreams in den Shapes dargestellt. Da die Methode zu der Suche nach Datacores bereits existiert, wäre es so möglich spezielle Eigenschaften auch noch zu den Datastreams darzustellen. Zum Beispiel ließe sich der MIME Typ herausfinden und könnte als Icon noch in der Shape erscheinen. Dann würde der Benutzer direkt auf einen Blick sehen, ob sich ein Bild, Text, Audio oder was auch immer hinter der URL verbirgt.

Statt nur nach den Datastreams der einzelnen Objekte zu suchen, besteht auch die Möglichkeit Relationen zwischen den einzelnen Objekten zu bilden. Als Beispiel bedeutet dies, dass alle Folien einer Powerpoint Präsentation in Relation zueinander stehen, denn sie gehören alle zu einer Präsentation. Diese Relationen zwischen abgelegten Objekten können abgerufen und dargestellt werden.
Die Funktion „searchForLobByRESTProtocol()“ kann zum testen verwendet werden und zeigt, wie der Abruf von Relationen nach Möglichkeit aussehen könnte.
Diese Funktion kam nie zum Einsatz, da sie noch umgeschrieben und angepasst werden müsste. Sie greift im Moment auf die ControlHosts mit Eingabefeldern der FedoraUserControlLibAPIA zu und kann so noch nicht in LOSVis eingebaut werden.

Außerdem wird für die Graphenerstellung in der Software LOSVis ein automatischer Layout Algorithmus von Netron verwendet. Dieser ist individuell nicht gut anpassbar und sollte ersetzt werden, so dass bei extrem vielen Datastreams eine Anordnung möglich ist, die nicht erwartet, dass der Benutzer ewig lang nach rechts scrollen muss.
Denkbar wäre es auch dem Benutzer die Möglichkeit zu geben, zwischen verschiedenen Darstellungsformen des Graphen zu wählen.

Für die Shapes fehlt noch ein Parsen der Strings, die als Texte in die Labels der Shapes übergeben werden. Wenn diese länger sind als die Shapes, muss eigentlich ein Zeilenumbruch erfolgen, was an dieser Stelle im Prototyp noch fehlt.

4 Fazit

Die vielen verschiedenen Techniken, die für die Entwicklung LOSVis zu lernen und umzusetzen waren, machte die Entwicklung nicht einfach und brauchte im Vorfeld bereits viel Zeit für die Einarbeitung. Das entstandene Ergebnis ist ein Prototyp. Viele Erweiterungen sind noch möglich und auch erwünscht.

Die aktuelle Funktionalität des Prototypen lässt sich wie folgt zusammenfassen:

LOSVis bietet eine graphische Benutzeroberfläche, die möglichst intuitiv gestaltet wurde. Menüs und Buttons sind übersichtlich angebracht. Platzmangel gab es an dieser Stelle noch keinen. Somit sind die Menüs problemlos um neue Funktionen erweiterbar.

Dem System wurden Kleinigkeiten hinzugefügt, um die ganze Software benutzerfreundlicher zu gestalten. Hierbei zu erwähnen ist der Ladebalken, die Möglichkeit mittels TAB die einzelnen Fenster durchzuschalten, nach der Stichworteingabe die Suche mit „Enter“ zu starten und die Befehle im Menü mit Shortcuts anzusteuern.

Die Funktionsweise des LOSVis-Clients besteht darin, durch eine Benutzereingabe einen Suchterm zu erhalten und auf Fedora danach zu suchen. Dabei werden der Titel und die Objekt-ID durchsucht. Das Ergebnis über alle zur Anfrage passenden Objekte aus der Datenbank erhält der Client im XML Format. Dieses XML kann er parsen und in einer Liste ausgeben. Wählt der Benutzer ein Element der Liste, startet der Client erneut eine Anfrage und erhält dieses Mal die Datastreams zu entsprechendem Objekt im XML Format gelistet. Dieses wird durch das Parsen wieder auseinander genommen und als Graph ausgeben. Die Shapes jedes einzelnen Datastreams enthalten eine URL die zu dem eigentlichen Inhalt des Datastreams führt.
Der Graph kann anschließend von dem Benutzer als JPEG gespeichert oder gedruckt werden. Zusätzlich besteht die Möglichkeit zu zoomen.

Das System funktioniert nun soweit wie ein Benutzer nach einem Objekt suchen und am Ende mit Hilfe der URL an den entsprechenden Inhalt des Objekts gelangen will.

Viele Dinge, die auch schon im Kapitel „Ausblick“ angeschnitten wurden sind als Erweiterungen denkbar und umsetzbar.

Die Erfahrung ein neues System aufzusetzen, in das viele verschiedene Techniken einfließen und vorhandene Frameworks zusätzlich genutzt werden müssen, war eine Herausforderung die mit dem Prototyp LOSVis endete. Und wenn LOSVis eingesetzt werden soll folgen noch viele Stunden weiterer Arbeit, die das System zu dem machen können, was es sein soll:

Eine Suchmaschine nach Relationen von Lernobjekten auf dem Repository-Management System Fedora.

5 Abbildungsverzeichnis

6 Quellenverzeichnis

[1]: **The Fedora Development Team**, Tutorial1: Introduction to Fedora, 2005, S.4

[2]: **The Fedora Development Team**, Tutorial1: Introduction to Fedora, 2005, S.5

[3]: **Fedora Projekt**, Overview: The Fedora Digital Object Model, Fedora Release 2.1.1, 2006, http://www.fedora.info/download/2.1.1/userdocs/digitalobjects/objectModel.html

[4]: **The Fedora Development Team**, Tutorial1: Introduction to Fedora, 2005, S.9

[5]: **Carl Lagoze, Sandy Payette, Edwin Shin, Chris Wilper**, Fedora:An Architecture for Complex Objects and their Relationships, Computing and Information Science, Cornell University, 2005, http://www.arxiv.org/ftp/cs/papers/0501/0501012.pdf

[6]: **Klerkx, Meire, Ternier, Verbert, Duval**, Information Visualisation: Towards an Extensible Framework for Accessing Learning Object Repositories, Dept. Computerwetenschappen, Katholieke Universiteit Leuven, 2004

[7]: **David Nation, Debbie Roberts**, Browse Hierarchical Data with the Degree of Interest Tree, http://www2.parc.com/istl/projects/uir/publications/items/UIR-2002-12-Nation-CHI2002-DOIDemo.pdf

[8]: **Nils Graue**, Requirements Engineering, Universität Duisburg-Essen, 2004, http://swe.uni-duisburg-essen.de/de/education/ws0405/RE/NilsGraue(2004)SemRE-NFRAusarbeitung.pdf

[9]:**HCIL**, Biodiversity Informatics Visualization, Stand 2006, http://www.cs.umd.edu/hcil/treemap/

[10]:**TechTarget**, Venn diagram, Stand 2006, http://whatis.techtarget.com/definition/0,,sid9_gci333063,00.html

[11]:**HCIL**, Biodiversity Informatics Visualization, Stand 2006, http://www.cs.umd.edu/hcil/spacetree/

[12]:**HCIL**, Biodiversity Informatics Visualization, Stand 2006, http://www.cs.umd.edu/hcil/biodiversity/#prototypes