

Bericht zum Master-Projektpraktikum im Rahmen des Forschungsprojekts INVISIP



Thomas Memmel (B.Sc)

Lehrstuhl für Mensch-Computer Interaktion WS 2002/2003, Februar 2003. Betreuer: Frank Müller

Universität Konstanz, Informatik & Informationswissenschaft

Zusammenfassung

Der Praktikumsbericht beschreibt die konzeptuellen Überlegungen und Programmierarbeiten an einer plattformunabhängigen Applikation zur Visualisierung von geo-räumlichen Metadaten. Diese Anwendung entstand an der Universität Konstanz im Rahmen eines Forschungsauftrags der Arbeitsgruppe für Informationssysteme innerhalb des europäischen Projektes INVISIP¹ (Information Visualization For Site Planning). Die seitens des Projektteams Konstanz entwickelte Softwareapplikation verfügte in ihrer ersten Version über zwei Arten von Visualisierungen, einen Scatterplot (Gundelsweiler 2002) und eine tabellenbasierte SuperTable² Lösung (Mommel 2002). Neue Visualisierungen wie ein 3D-Scatterplot, eine weitere Variante der und ein so genannter Circle Segment View, die in der nächsten Version integriert werden sollten, stellten neue Anforderungen an das zugrunde liegende Datenmodell. Zusätzlich sollten mehrere Datenbanken angeschlossen werden. Der vorliegende Praktikumsbericht beschreibt die Anforderungen, die an die nächste Generation der Softwareanwendung gestellt wurden, die als Projektbezeichnung den dem Forschungsprojekt gleich lautenden Namen „INVISIP“ erhielt und später mit dem Namen „VisMeB“ (Visueller Metadatenbrowser) getauft wurde.

1 Einleitung

Im Jahr 1998 wurde das EU-Forschungsprojekt INSYDER ins Leben gerufen. Das System sollte das Document Retrieval im Internet erleichtern. Der Mehrwert des Systems gegenüber konventionellen Suchmaschinen sollte darin bestehen, neben einer Volltextanzeige verschiedene Visualisierungen zur Verfügung zu stellen, mit denen sich relevante Dokumente schneller identifizieren und in Relation zueinander betrachten ließen. Eine so genannte „ResultTable“ ermöglichte eine gewohnt tabellarische Darstellung der Metadaten, wobei ein geteiltes Programmfenster dazu verwendet werden konnte im unteren Bereich des Bildschirms den Volltext eines Dokumentes anzuzeigen (Abbildung 1). Der INSDYER Scatterplot stellte alle Dokumente zweidimensional in einem Koordinatensystem dar. Diese Darstellungsform eignete sich besonders, um Zusammenhänge zwischen Dokumenten erkennen zu können. Mit Hilfe von Barcharts und einer so genannten „SegmentView“ wurden Relevanzwerte von Dokumenten und Textabschnitten visualisiert (Abbildung 1).

¹ Finanziert von der Europäischen Kommission, IST Programm, Projekt Nummer IST-2000- 29640.

² Begriff „SuperTable“ aus Müller et al. 2002.

Das INSYDER System sollte Benutzer nicht mit vielen und unübersichtlichen Ergebnislisten kognitiv überlasten. Suchanfragen sollten vom Benutzer leicht eingegeben und verwaltet werden können. Die Anzeige der Ergebnisse sowie deren Ranking sollten möglichst einfach und verständlich sein. Die Visualisierung der Suchergebnisse sollte dem Benutzer bei der Extraktion der für ihn relevantesten Daten eine optimale Hilfe sein, indem nützliche Dokumente vom Suchenden schnell identifiziert werden können.

Bei der Entwicklung des Systems waren die so genannten „5 T“ Kriterien die Anleitung zur Gestaltung der grafischen Benutzeroberfläche. Diese fünf „T“ stehen für eine Konzentration auf die Zielgruppe der Anwendung (Target Group), die typischen Aufgaben der Benutzer (Typical Tasks), die technische Umgebung (Typical Environment), die Beschaffenheit und Art der Daten, die visualisiert werden sollen (Type Of Data) und die Benutzbarkeit der Anwendung mit minimalem Trainingsaufwand (Training) (Mann & Reiterer 1999)

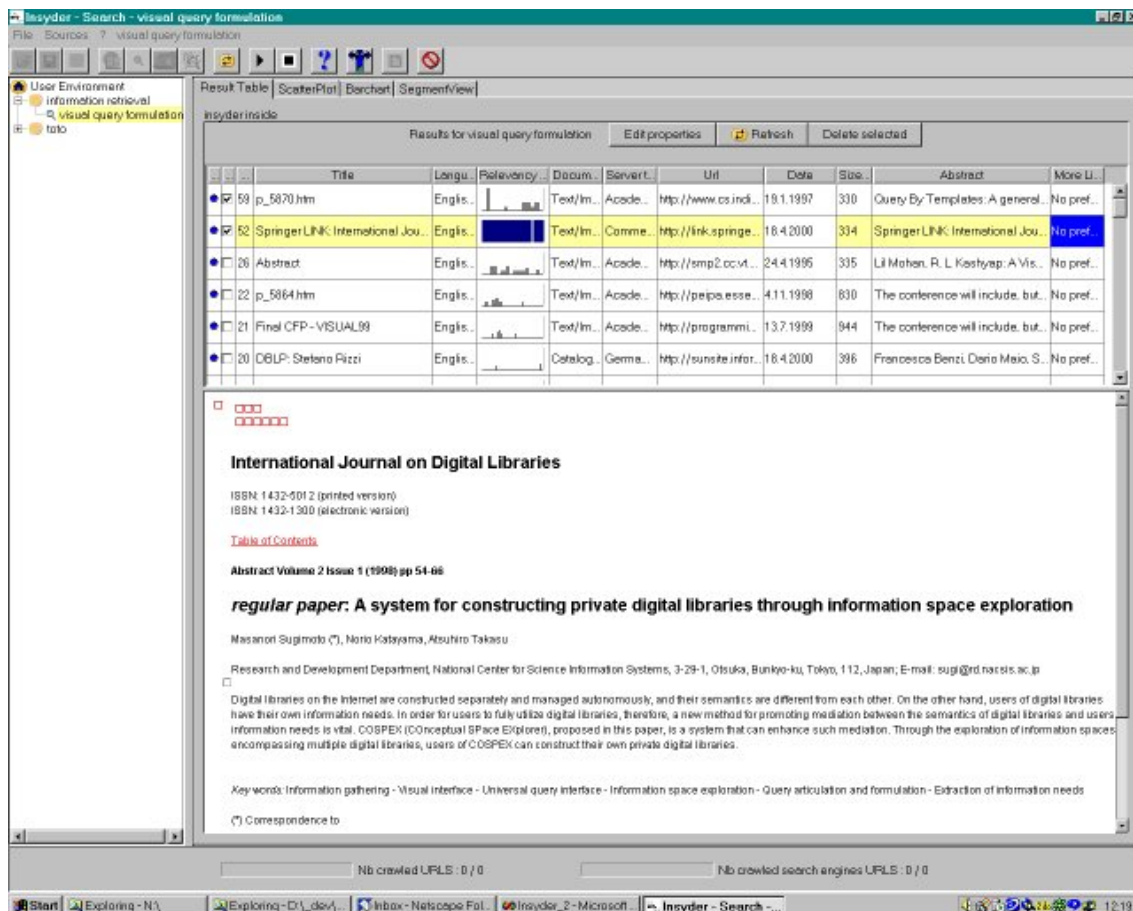


Abbildung 1: INSYDER mit ResultTable und Volltextansicht

Eine Evaluation von INSYDER mit 40 Benutzern hinsichtlich der „5 T“ Faktoren und der verwendeten Visualisierungen ergab, dass 50% der Benutzer die ResultTable als visuelle Unterstützung bei der Verwaltung der Suchergebnisse favorisierten und die tabellenbasierte Darstellung demnach auch am meisten und am längsten benutzt wurde. Gleichzeitig zeigte sich, dass die Mehrzahl der Nutzer allgemeine Schwierigkeiten bei der Verwendung der Visualisierungen hatte. Die Visualisierungen, wie Scatterplot oder BarCharts, wurden hauptsächlich nur im Zusammenhang mit der ResultTable verwendet und kaum als alternative Visualisierungen erkannt. Das Umschalten zwischen verschiedenen Visualisierungsformen verwirrte den Benutzer und war somit ein Faktor aufgrund dessen die ResultTable von allen Visualisierungen am meisten Zuspruch fand (Mann 2001).

Die Ergebnisse der Evaluation des INSYDER Systems führten zu der Überlegung, eine tabellenbasierte Visualisierung als Basis des Systems zu verwenden und die ResultTable zusammen mit anderen Visualisierungen zu einer SuperTable zu kombinieren und zu integrieren (Abbildung 2).

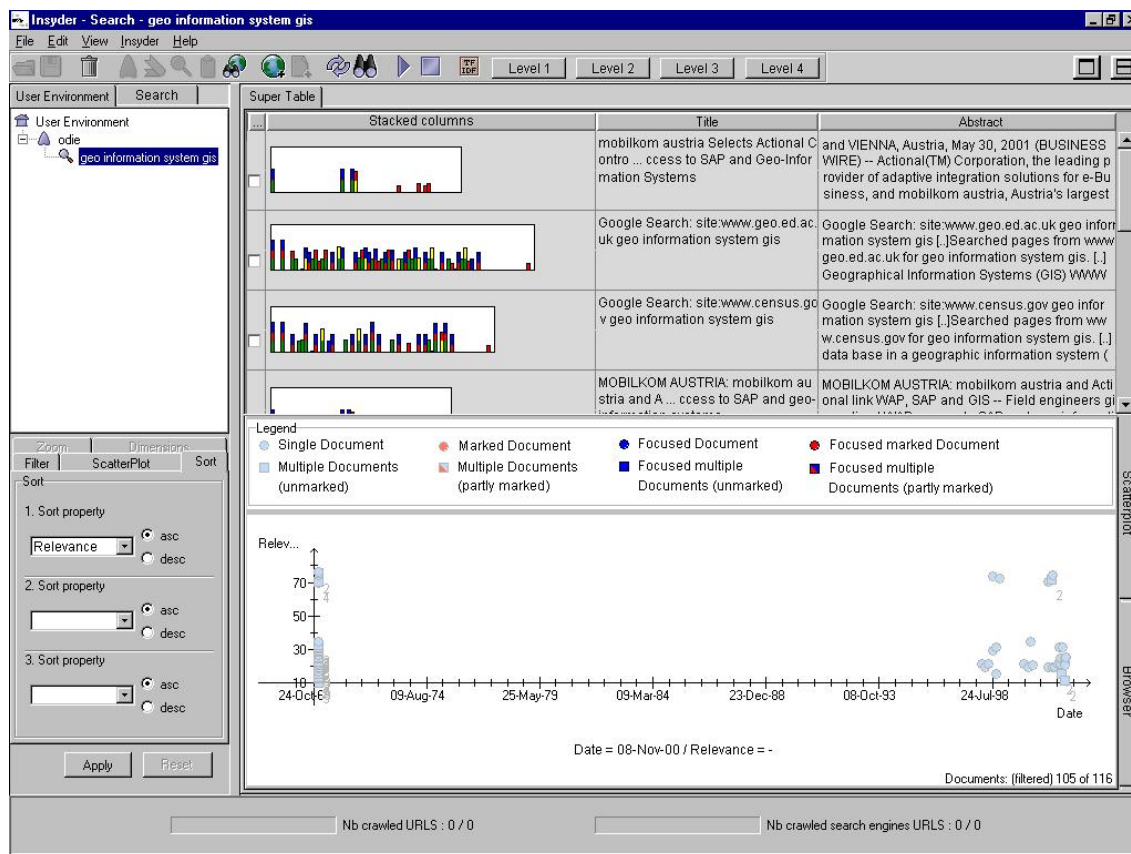


Abbildung 2: INSYDER Redesign HTML Prototyp

Die Verbesserungen und neuen Komponenten des INSYDER Redesign sind gleichzeitig Bestandteil des INVISIP Projektes. Ergebnis von INVISIP soll wie bei INSYDER ein System sein, das die Suche nach relevanter Information, sowie deren Auswertung und Verwaltung unterstützen soll. Jedoch ist die Anwendungsdomäne von INVISIP die geo-räumliche Ortsplanung, unter anderem also die Suche nach Bauplätzen oder Erschließungsgebieten. Das System soll eine Hilfestellung aller betroffenen Benutzergruppen, von der öffentlichen Einrichtung bis hin zum privaten Nutzer, sein. Dabei sollen Funktionen sowohl für den Privatbenutzer, als auch für Experten, wie Techniker oder Planungsbüros, zugeschnitten sein. Im Folgenden wird die von der Universität Konstanz entwickelte Anwendung ebenfalls als „INVISIP“ bezeichnet, wobei es sich dabei um einen internen Projektnamen handelte.

Das INVISIP Datenmodell konnte auf den von INSYDER generierten XML Dokumenten aufbauen. Sie waren das Resultat eines INSYDER Datenbankeexport im Anschluss an eine Suchanfrage mit der alten Anwendung. Im Herbst 2002 konnte die erste INVISIP Version als Nachfolger von INSYDER (Reiterer et. al 2000) als Release freigegeben und den europäischen Projektpartnern vorgestellt werden. Die Entstehung dieser Version wurde durch zwei Bachelor Abschlussarbeiten dokumentiert (Gundelsweiler 2002, Memmel 2002).

Bereits während der Fertigstellung der ersten INVISIP Version wurden zahlreiche Verbesserungsmöglichkeiten festgestellt. Insbesondere als problematisch wurde dabei die Geschwindigkeit der neuen tabel-

lenbasierten Visualisierung mit dem Namen „GranularityTable³⁴“ dokumentiert. Dabei handelt sich um eine Tabelle, in denen Visualisierungen und Interaktionselemente integriert sind. Die Tabelle erhielt ihren Namen vom zugrunde liegenden Granularitätskonzept. Das Konzept der Granularität sieht einen globalen, sowie für jedes Dokument einen einzelnen Regler vor, der genauso wie die verschiedenen Visualisierungen in die tabellarische Anordnung eingebettet wurde. Die GranularityTable umfasst vier Spalten, je eine für Selektion, Visualisierung, Text und den Regler. Mit dem Granularitätsregler kann der Benutzer die verschiedenen Detailstufen ansteuern, wobei mit zunehmender Granularitätsstufe mehr Detailinformation zum Dokument angezeigt wird. Abbildung 3 zeigt die GranularityTable in der ersten Stufe, wobei eine einzelne Spalte in die sechste und höchste Stufe geschaltet ist, in der der Volltext des Dokumentes angezeigt wird und die Textspalte mit der Visualisierungsspalte zusammenfällt.

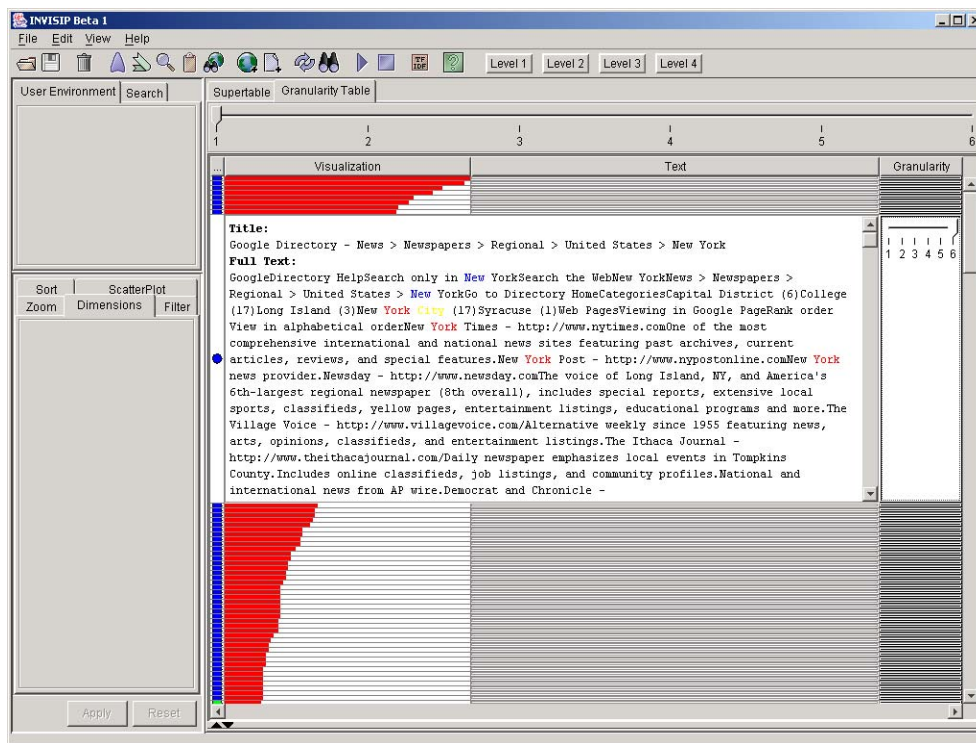


Abbildung 3: INVISIP - Version 1 GranularityTable

Pro Datensatz enthält diese Tabelle eine Zeile, so dass bei sehr vielen Daten ein großes und speicherintensives Java Objekt entsteht. Das zugrunde liegende, auf dem Model-View-Controller Pattern (Gamma et al. 1994) basierende Datenmodell, besaß an sehr vielen Stellen Funktionalitäten, die auf Grund von stetigen Veränderungen und Erweiterungen des Programmcodes bald von keiner Visualisierung („View“) des Systems mehr benötigt wurden. Andererseits fehlten dafür aber neue Dienste im Datenmodell, die von vielen Visualisierungen, vor allem von der GranularityTable benötigt wurden. Somit mussten viele Berechnungen zur Visualisierungszeit durchgeführt werden, die eigentlich bereits zu einem früheren Zeitpunkt vom Datenmodell durchgeführt werden müssten. Gerade die verschiedenen TableCellRenderer – dies sind die Systemkomponenten, die bei jedem Zeichnen der Tabelle aufgerufen werden, um den Zelleninhalt der GranularityTable zu zeichnen - sollten frei sein von jeglicher algorithmischer Berechnung. Mit dem alten Datenmodell wurden diese TableCellRenderer bei großen Datenmengen (> 200 Dokumenten) zum Flaschenhals der GranularityTable und damit auch der gesamten Applikation.

³ GranularityTable ist die Entwicklungsbezeichnung für eine SuperTable mit Granularitätskonzept nach Müller et al. 2002.

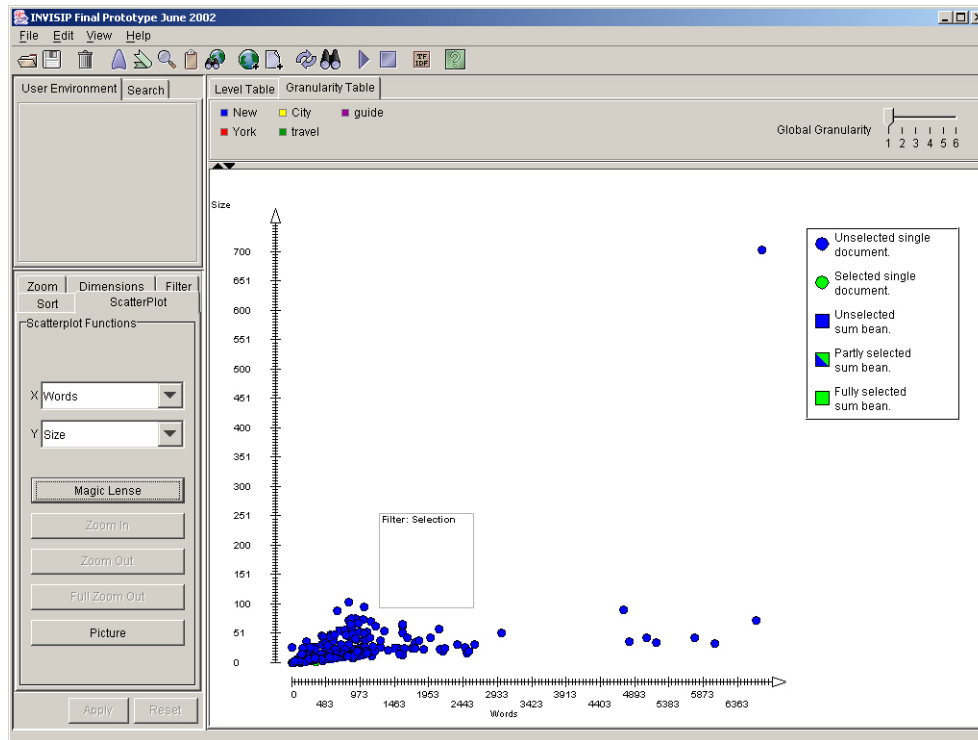


Abbildung 4: INVISIP - Version 1 Scatterplot

Die Klassen des Datenmodells führten sehr viele Referenzen auf andere teilnehmende Komponenten und die Verwaltung dieser Referenzen über die JAVA Referenztabelle kostete sehr viel Zeit, was mit dem Erfordernis eines schnellen Datenzugriffs konfliktierte.

Das Datenmodell sollte daher so umgebaut werden, dass es zum einen schlanker und leistungsfähiger sein könnte und zum anderen die Views stärker als bisher entlasten konnte.

Zu den Verbesserungsmöglichkeiten addierten sich weitere neue Anforderungen an das INVISIP Datenmodell. So sollte von der bisherigen Einspeisung der Metadaten durch XML Dokumente umgestiegen werden auf Datenbankbindung. Diese Portierung erschien gerade deshalb notwendig, weil der INVISIP Projektpartner Fraunhofer für die Datenversorgung verantwortlich war und die Metadaten mit Hilfe einer Oracle Datenbank zur Verfügung stellen wollte. Zum Testen der Anwendung und bis zur Fertigstellung der Datenbank Architektur seitens Fraunhofer sollte es jedoch möglich sein, auch weitere Datenbanken anbinden zu können. Dabei sollte es sich ausdrücklich nicht um Datenbanken gleichen Typs handeln und auch die Inhalte sollten variieren. Die INVISIP Software sollte dadurch zu einem hochgradig dynamischen System ausgebaut werden, welches einfach an unterschiedlichste Datenquellen im Intra- oder Internet angeschlossen werden kann.

Durch die Anbindung verschiedener Datenquellen, die mit unterschiedlichsten (Meta-)Daten gefüllt waren, konnte nicht mehr davon ausgegangen werden, dass die Visualisierungen des INVISIP Systems stets eine feste Anzahl von Daten anzeigen würden. Vielmehr würden sich die Visualisierungen an die Datentypen anpassen und damit ihr Aussehen verändern müssen.

Insgesamt ergab sich ein umfangreiches Aufgabenpaket für die Entwicklung des nächsten INVISIP Release. Während sich zahlreiche Programmierer aus dem INVISIP Team um die Entwicklung weiterer Visualisierungen kümmerten, wurde die Überarbeitung des Systemkerns wie in Tabelle 1 gelistet zwischen den Programmierern aufgeteilt, die die erste INVISIP Version gemeinsam erstellt hatten und damit den besten Einblick in die Systemarchitektur hatten. Meine Aufgabe war die Portierung des Datenmodells von XML auf eine dynamische, datenbankgestützte Lösung.

Aufgabenbeschreibung	Programmierer Mommel, Thomas	Programmierer Gundelsweiler, Fredrik
Portierung von XML auf Datenbankanbindung	•	
Variable Anbindung an verschiedene Datenbanken	•	
Verbesserung der Geschwindigkeit		•
Erleichterung des Zugriffs auf Daten		•
Vereinfachung des bestehenden Datenmodells, Neukonzeption	•	•

Tabelle 1: Aufteilung der Arbeiten für Release 2 des Datenmodells

2 Das neue datenbankgestützte INVISIP Datenmodell

Bei den Vorüberlegungen zur Gestaltung des neuen Datenmodells stellte sich die zentrale Frage, wie der Zugriff auf die verschiedenen Datenbanken dynamisch gestaltet werden sollte. Dabei musste insbesondere das bisher verwendete „WWWDocument“ in den Mittelpunkt der Überlegungen gerückt werden. Das WWWDocument repräsentierte genau einen Metadatensatz und ist der eigentliche Träger von Information im INVISIP Datenmodell. Es erhielt seinen Namen aufgrund des bisherigen Inhaltes, welcher sowohl im INSYDER Projekt, wie auch in der ersten Version von INVISIP, aus Relevanzwerten und Metadaten über Dokumente aus dem World Wide Web bestand. Für jedes eingelesene XML Dokument existierte im Datenmodell genau eine Instanz der Klasse WWWDocument und dadurch genau eine systeminterne Repräsentation.

Das WWWDocument konnte bisher eine relativ statische Form haben, da die Datenfelder der entsprechenden Java Klasse letztlich Abbildungen der XML Tags der eingelesenen Dokumente waren. Durch die Anforderung einer Anbindung an variable Datenquellen konnte diese Projektion nicht mehr erfolgen. Auf einer abstrakten Ebene musste das WWWDocument daher offen sein für eine dynamische Anzahl von Werten, sowie für unterschiedlichste Datentypen.

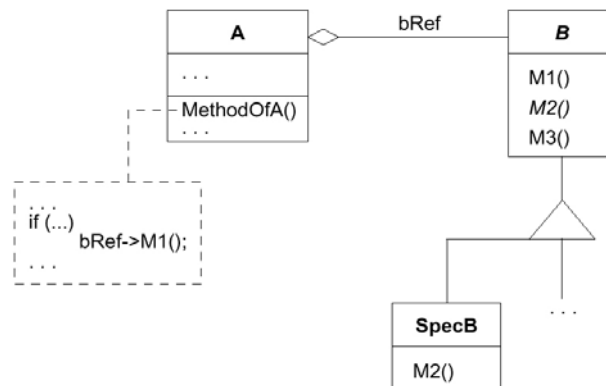


Abbildung 5: Abstrakte Kopplung

Somit wurden verschiedene Typen von WWWDocuments eingeführt, die alle von einer abstrakten Klasse AbstractMetaData (Quellcode 1) abgeleitet sind. Die implizite Aufspaltung des WWWDocuments in mehrere Bestandteile erfolgte dabei nach dem Prinzip der abstrakten Kopplung (Abbildung 5), welches bereits bei der ersten Version von INVISIP an vielen Stellen Anwendung finden konnte.

Auf Ebene der Datentypen musste ein Abgleich mit den in Datenbanken verwendeten Feldern getroffen werden. Dies entsprach einer Art Mapping zwischen den uneinheitlichen Datentypen verschiedener Datenbanksysteme (Vgl. MySQL, Oracle, MS SQL, etc.) und den existierenden Datentypen von Java.

Somit wurde eine Abstraktion auch auf Seite der Datenbankbindung eingeführt, da zu jeder Datenbank eine eigene Verbindung hergestellt werden muss (Datenbanktreiber, Login Daten, etc.). Um eine Art Datentyp Standard zur Verfügung zu stellen, der an alle Datenbank Verbindungsklassen vererbt werden konnte und der existierte, so lange die entsprechenden Unterklassen diese Angaben nicht überschrieben, wurde dem Interface Database (Quellcode 2) die abstrakte Klasse AbstractDatabase zur Seite gestellt.

```
public abstract class AbstractMetaData implements Comparable, Serializable
{
    ...
    /**
     * Gives a string representation of the data type.
     * @return the type of the data as string
     */
    public String getType() {
        return toString();
    }

    /**
     * @param o the comparable object
     * @return the higher / lower value 1 / -1 for quick sort
     */
    public abstract int compareTo(Object o);

    /**
     * @return the string representation of the data
     */
    public abstract String getString();
}
```

Quellcode 1: Klasse AbstractMetaData

```
public interface Database extends Serializable
{
    public void connect();
    public ResultSet execute(String query);
    public void close();
    public Vector getTableNames();
    public Vector getColumnNames(String tableName);
    public String getName();
    public void assignMinMaxDescData(Vector v, JProgressBar bar);
    public void buildWWWDocuments(Vector v, JProgressBar bar);
    public Vector askForOccurance(String colName, JProgressBar progress);
    public void addFullTextToNodeVector(Vector nodeVector);
    public AssignmentTreeNode getDefaultTree(AssignmentTreeNode root);
}
```

Quellcode 2: Interface Database

Durch Dynamic Loading können später zur Laufzeit der Applikation weitere Datenbankklassen hinzugefügt und geladen werden, die konform der Spezifikation das Interface Database implementieren.

Mit Hilfe der Methode getType() werden die Inhalte von angeschlossenen Datenbanktabellen auf die zugrunde liegenden Datentypen untersucht und damit wird bestimmt, welchem internen Datentyp des Datenmodells die Tabellenwerte zugeordnet werden (Quellcode 3). Zum Auslesen der Datentypen einer angebundenen Datenbank stellt Java bereits die entsprechenden Methoden zur Verfügung, die nur noch zur Anwendung gebracht werden müssen (Quellcode 4).

```

public AbstractMetaData getType(String type)
{
    if(type.equalsIgnoreCase("int4"))return new IntegerMetaData(0);
    if(type.equalsIgnoreCase("float4"))return new DoubleMetaData(0);
    if(type.equalsIgnoreCase("numeric"))return new IntegerMetaData(0);
    if(type.equalsIgnoreCase("number"))return new IntegerMetaData(0);
    if(type.equalsIgnoreCase("varchar"))return new StringMetaData("");
    if(type.equalsIgnoreCase("varchar2"))return new StringMetaData("");
    if(type.equalsIgnoreCase("text"))return new StringMetaData("");
    else return new IntegerMetaData(0);
}

```

Quellcode 3: Methode getType() der Klasse AbstractDatabase

```

public Vector getColumnNames(String tableName) {
    Vector names = new Vector();
    try {
        ResultSet res = dbConnection.getMetaData().getColumns(TABLE, null, tableName, "%");
        while (res.next()) {
            AbstractMetaData data = getType(res.getString(6));
            names.addElement(new Object[] {res.getString("COLUMN_NAME"), data});
        }
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    return names;
}

```

Quellcode 4: Methode getColumnNames() der Verbindungs-Klasse FraunhoferDatabase

Das dynamische Auslesen verschiedenster Datenquellen ist somit möglich. Mit Hilfe des Werkzeuges „Assignment“ (dt. Zuweisung) können nun flexible Zuordnungen von Daten aus den Tabellen der Datenbanken auf die Visualisierungen vorgenommen werden. Bis zur Fertigstellung der Komponente „Visual Assignment“ (Gundelsweiler & Memmel 2003) wird als Platzhalter eine Art Default Assignment in das Datenmodell Framework eingebaut. Die Art und Weise wie der Benutzer später die Visualisierungen an die Datenquelle anpassen kann, so dass die Möglichkeiten der Anwendung optimal genutzt werden können, wird somit vorerst simuliert. Die Idee des Assignment Konzeptes ist es, bestimmte Visualisierungen auch nur mit bestimmten Datentypen belegen zu können. Beispielsweise sollen BarChart Diagramme nur mit Datentypen belegt werden dürfen, die zum Typ Integer oder Double konvertiert werden können.

Durch die dynamische Zuordnung kristallisiert sich heraus, dass natürlich die gleichen Daten mehrmals innerhalb unterschiedlicher Visualisierungen verwendet werden können. Dies ist insbesondere in der GranularityTable der Fall, die in den tieferen Anzeigestufen durchaus nochmals dieselben Metadaten wie die darüber liegenden Granularitätsstufen enthalten kann.

Der grundlegende Unterschied zum vorherigen Datenmodell besteht darin, jeden Datenwert, den der Benutzer in den Visualisierungen von INVISIP anzeigen lassen will, nur genau einmal abzuspeichern. In der ersten INVISIP Version wurde eine Referenz auf ein WWWDocument an vielen Stellen an andere Komponenten weitergereicht oder dazu sogar eine Kopie des WWWDocuments erzeugt. Im neuen System werden durch die Vermeidung von Redundanz und übermäßigem Speicherbedarf bereits teilweise die Probleme mit der Performance der Applikation gelöst, die von (Gundelsweiler 2003) noch weiter erörtert und erodiert werden.

```

public void buildWWWDocuments(Vector nodeVector, JProgressBar progress) {
    ...
    try {
        //columnName = ((AssignmentTreeNode)nodeVector.elementAt(2)).getUserObject().toString();
        String numberquery = "SELECT Count(*) FROM " + TABLE;
        result = execute(numberquery);
        result.next();
        int numberOfDocs = result.getInt(1);
        progress.setValue(0);
        progress.setMaximum(numberOfDocs);
        progress.setString("Reading Documents From Database...");
        boolean fakedSelection = false;
        dataQuery = "SELECT";
        for (int i = 2; i < nodeVector.size() - 1; i++) {
            node = (AssignmentTreeNode) nodeVector.elementAt(i); //what's the column and table name?
            columnName = (String) node.getUserObject();
            dataQuery += " " + columnName + ",";
        }
        if (nodeVector.size() > 2) {
            node = (AssignmentTreeNode) nodeVector.elementAt(nodeVector.size() - 1);
            columnName = (String) node.getUserObject();
            dataQuery += " " + columnName;
            dataQuery += " FROM " + TABLE;
        }
        else dataQuery += " * FROM " + TABLE;

        result = execute(dataQuery);
        counter = 0;
        while (result.next()) {
            dataVector = new Vector();//0 and 1 is selection and ID --> so add both at beginning pos 0 and 1
            dataVector.addElement(new BooleanMetaData(fakedSelection)); //(fakedSelection),DOCUMENT_SELECTION);
            dataVector.addElement(new IntegerMetaData(counter)); //at the moment the faked id //result.getInt("oid"),DOCUMENT_ID);
            for (int i = 2; i < nodeVector.size(); i++) { //now add all other values
                node = (AssignmentTreeNode) nodeVector.elementAt(i);
                columnName = (String) node.getUserObject();
                if (node.getData().instanceof IntegerMetaData) metaData = new IntegerMetaData(result.getInt(columnName));
                else if (node.getData().instanceof StringMetaData) metaData = new StringMetaData(result.getString(columnName));
                ...
                progress.setString("Storing Min Data...");
                dataVector.addElement(metaData);
            }
            wwwDoc = new WWWDocument(dataVector);
            wwwDoc.setRow(dataManager.getArrayOfDocVector());
            dataManager.addWWWDocument(wwwDoc);
            progress.setString("Reading Dataset # " + counter);
            progress.setValue(counter++);
        }
        close();
    }
    ...
}

```

Quellcode 5: Methode buildWWWDocuments() in Klasse der FraunhoferDatabase

Der Zusammenbau eines WWWDocument funktioniert dabei wie im Folgenden erklärt. Hauptmethode hinsichtlich des Aufbaus der WWWDocuments und damit Teil des Kerns des Datenmodells ist die Methode buildWWWDocument(...), wie in Quellcode 5 abgebildet.

Ihr wird ein Vektor von Baumknoten vom speziell angepassten Typ `AssignmentTreeNode` übergeben, welcher sich von der Java Klasse `JTree` ableitet. Zum Testen des Software Frameworks werden die Baumknoten mit vordefinierten Zuordnungswerten besetzt. Zu einem späteren Zeitpunkt soll das visuelle Zuordnungstool (Visual Assignment) eben solche Baumstrukturen verwenden, um die Zuordnungen von Datentypen und Metadaten zu den unterschiedlichen Visualisierungen zu untergliedern und zu strukturieren (siehe Anhang, Abbildung 11). Der benutzerdefinierte Zuordnungsbaum wird in einem Zwischenschritt auf die Positionen eines Vektors übertragen, so dass die Methode keinen `JTree` bekommt, sondern eine zur Datenverwaltung besser geeignete Datenstruktur. Bei diesem Mapping werden zugleich mehrfach auf die Visualisierungen zugeordnete Metadaten als Duplikate erkannt und eliminiert (Quellcode 6).

```

/**
 * read all leaf nodes.
 * @param void
 */
private void readAllLeafNodes(AssignmentTreeNode node) {
    if(node.isLeaf()) {
        if(node.getUserObject().toString().startsWith("Free")||node.getUserObject().toString().startsWith("Default")) {
            //no item assigned --> set to -1 and do not add to vector
            setAssignmentNumberToNotAssigned(node.getID()); //set to -1
        }
        else { //item is assigned --> assignment number has to be set to correct vector position (later?), add to vector
            if(nodeVectorContains(node)) { //do not add if same was added before and set position to same position as added element
                //nodeHelpPos is saved in method nodeVectorContains() called in else if statement
                //System.out.println("CONTAINS NODE:"+node.getUserObject());
                setAssignmentNumberToAssigned(node.getID(), nodeHelpPos); //set to correct vector pos that already is in vector
                node.setID(nodeHelpPos); // set node ID too , evtl. not necessary because IDs are not used any more
            } else { //add new element
                nodeVector.addElement(node); // 0 and 1 are used by selection and document ID, this is because nodeVectorPos starts at 2
                setAssignmentNumberToAssigned(node.getID(), nodeVectorPos); //set to correct vector pos
                node.setID(nodeVectorPos); // set node ID too , evtl. not necessary because IDs are not used any more
                nodeVectorPos++;
                progress.setValue(counter++);
            }
        }
    }
    return;
}
else {
    for(int i = 0; i < node.getChildCount(); i++) {
        readAllLeafNodes((AssignmentTreeNode)node.getChildAt(i));
    }
}
}
}

```

Quellcode 6: Methode `readAllLeafNodes(...)`

Zweiter Übergabeparameter ist eine Referenz auf die in der Anwendung angezeigte Progress Bar. Die Übergabe dieses Parameters ermöglicht ein Update des Fortschrittsbalkens unmittelbar während des Zusammenbaus der WWWDocuments. Zunächst wird mit Hilfe einer SQL Abfrage an die Datenbank die Gesamtzahl der Metadatensätze in der Datenbank ermittelt (Quellcode 5, grün markiert). Damit kann die Progress Bar mit einem Maximalwert initialisiert werden, um den Fortschritt bei der Datenverarbeitung anzeigen zu können.

Da die Knoten den gleichen Namen tragen, wie die Spalten der Datenbanktabellen die sie repräsentieren, kann der um Duplikate reduzierte Vektor von Knoten (im Quellcode „`nodeVector`“ genannt) dazu verwendet werden, eine SQL Anfrage an die Datenbank zu stellen. An dieser Stelle findet somit das wichtige, dynamische Zusammensetzen einer Datenbankabfrage unter Berücksichtigung der Benutzerauswahl

statt. Dadurch wird erreicht, dass nicht mehr Datensätze von der Datenbank angefragt werden, als aufgrund der Zuordnung erforderlich (Quellcode 5, blau markiert).

Aufgrund der aus der Anfrage zurückgegebenen Datenbankergebnisse wird dann ein Vektor zusammengesetzt, der alle Unterklassen des Typs AbstractMetaData akzeptiert. Der Vektor hat seine volle Größe erreicht, wenn eine komplette Tabellenzeile aus dem Anfrageergebnis ausgelesen wurde. Dann wird ein neues WWWDocument erzeugt, welches mit Hilfe des Vektors initialisiert und so mit Daten gefüllt wird. Das fertige WWWDocument, das genau eine Tabellenzeile bzw. genau einen Metadatenatz repräsentiert, wird dem Datenmanager (Gundelsweiler 2003) zur Verwaltung übergeben

Durch eine Zuordnung des Benutzers erhalten verschiedene Visualisierungen alle Metadaten, die angezeigt werden sollen. Jede Visualisierung hat dabei eine oder mehrere Variablen in der Assignment Klasse. Diese stellen eine wichtige Komponente bei der Verwaltung von Metadaten in WWWDocuments und Visualisierungen dar. Den Visualisierungen muss nur noch mitgeteilt werden, an welcher Stelle dieses Datenvektors die Daten liegen, die der Visualisierung durch das Assignment zugeordnet worden sind.

```
private void setAssignmentNumberToAssigned(int nodeIntID, int vPos)
{
    if (nodeIntID == GRANULARITYTABLE_LEVEL1_1) GRANULARITYTABLE_LEVEL1_1 = vPos;
    else if (nodeIntID == GRANULARITYTABLE_LEVEL2_1) GRANULARITYTABLE_LEVEL2_1 = vPos;
    ...
    ...
}
```

Quellcode 7: Methode *setAssignmentNumberToAssigned ()* der Klasse *Assignment*

Für jede Visualisierung wird eine eindeutige ID in einer für alle Klassen zugreifbaren abstrakten Klasse geführt. Durch das Assignment wird dieser ID ein neuer Wert zugeordnet, nämlich die Position der anzuzeigenden Daten innerhalb des Datenvektors (Quellcode 7). Wird einer Visualisierung nichts zur Anzeige zugeordnet, wird der Wert der Visualisierung entsprechend mit „-1“ gekennzeichnet (Quellcode 8).

```
private void setAssignmentNumberToNotAssigned(int nodeIntID)
{
    if (nodeIntID == GRANULARITYTABLE_LEVEL1_1) GRANULARITYTABLE_LEVEL1_1 = -1;
    else if (nodeIntID == GRANULARITYTABLE_LEVEL2_1) GRANULARITYTABLE_LEVEL2_1 = -1;
    ...
    ...
}
```

Quellcode 8: Methode *setAssignmentNumberToNotAssigned ()* der Klasse *Assignment*

Greift die Visualisierung auf das Datenmodell zu, um Daten anzuzeigen, verwendet sie genau diese Variablen. Somit erhält die Visualisierung genau die Sorte von Daten, die aufgrund der Zuordnung für sie vorgesehen sind. Aufgrund der Notwendigkeit der Zuordnung kann ausgeschlossen werden, dass Visualisierungen falsche Daten bzw. Datentypen erhalten. Die Art der anzuzeigenden Daten soll in der Visualisierung keine Rolle spielen. Das heißt, es ist nicht wichtig, ob beispielsweise der Name eines Metadatum „rank“ oder „size“ ist. Wichtig ist vielmehr der Datentyp, der angezeigt werden soll, also beispielsweise Integer. Die Visualisierung weiß durch das neue Datenmodell zu jeder Zeit welche Datentypen durch die Variablen geliefert werden. Damit hat sie keinerlei Anpassungsprobleme.

Abbildung 6 zeigt eine einfache Darstellung der konzeptionellen Idee hinter dem neuen Datenmodell. Dabei wurden die Bereiche Assignment (Datenzuweisung) und DB-Connection (Datenbankanbindung) von mir umgesetzt. Während in der Klasse zur Datenbankverbindung die Metadaten eingelesen werden, dient der Teil Assignment dazu, die Metadaten zu strukturieren und mit Hilfe des Datenmanagers die WWWDocuments aufzubauen.

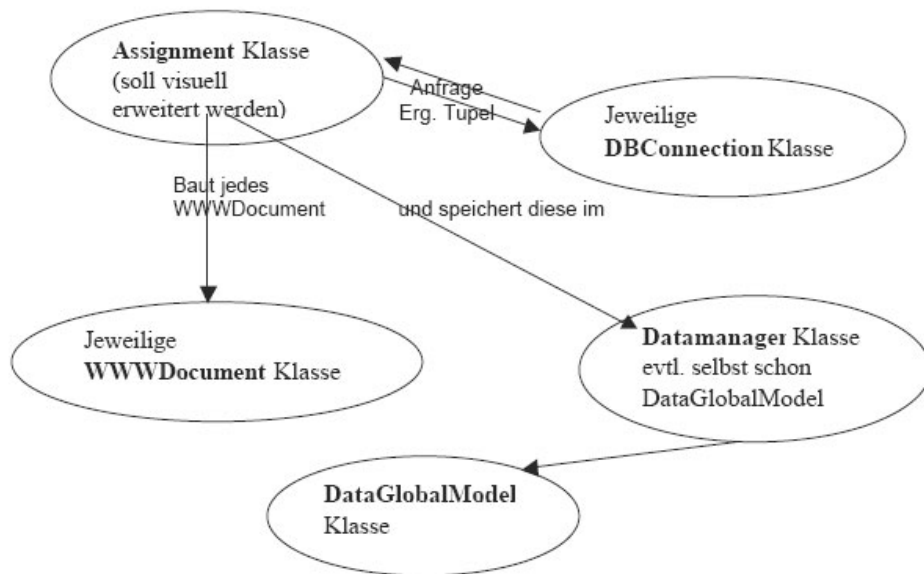


Abbildung 6: Visualisierung der Idee zum Datenmodell

Durch den Zusammenbau der Datenbasis für den Datenmanager nimmt der Bereich Assignment somit eine zentrale Rolle ein. Abbildung 7 zeigt eine weitere konzeptuelle Zeichnung des Zusammenspiels der Komponenten. Hier ist das Konzept bereits um die Idee des Visual Assignment erweitert.

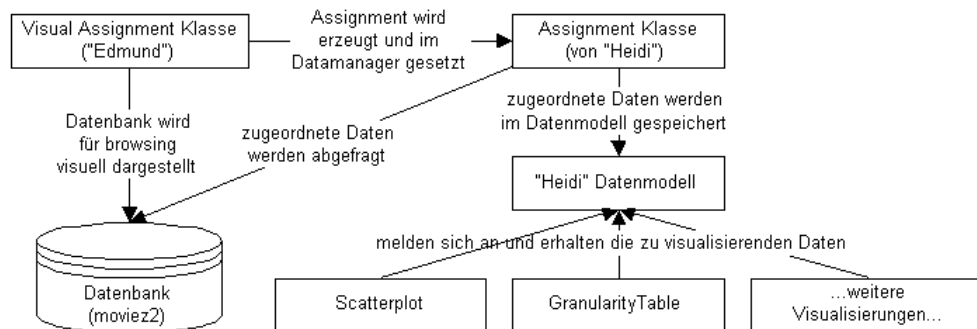


Abbildung 7: Visualisierung der Idee zum Datenmodell

3 Zusammenfassung

Das Grundkonzept des neuen Datenmodells ist, Anforderungen an Flexibilität und Leistungsfähigkeit zu verbinden und dabei eine möglichst einfache und leicht verständliche Struktur zu bewahren. So kann gewährleistet werden, dass auch zukünftige Arbeiten am Projekt, wie beispielsweise die Erweiterung um weitere Visualisierungen, leicht stattfinden können.

Durch das neue Datenmodell kann die INVISIP Softwareanwendung auf entfernte Datenquellen zugreifen und deren Metadaten visualisieren. Die vielfältigen Einsatzmöglichkeiten der Anwendung werden dabei

gerade dadurch sichtbar, dass Filmdatenbanken genauso visuell dargestellt werden können, wie Datenquellen mit geo-räumlichen Metadaten. Das Datenmodell kann leicht erweitert werden und kann auch künftig Basis weiterer Arbeiten sein, insofern das zugrunde liegende Design Pattern von künftigen projektangehörigen Programmierern beachtet wird und keine so genannten „Dirty Hacks“ dazu führen, dass die Strukturen des Datenmodells aufweichen. Dies wäre in der Konsequenz sicherlich für die Leistungsfähigkeit des Systems (siehe Abbildung 8, UML Diagramm) von Nachteil.

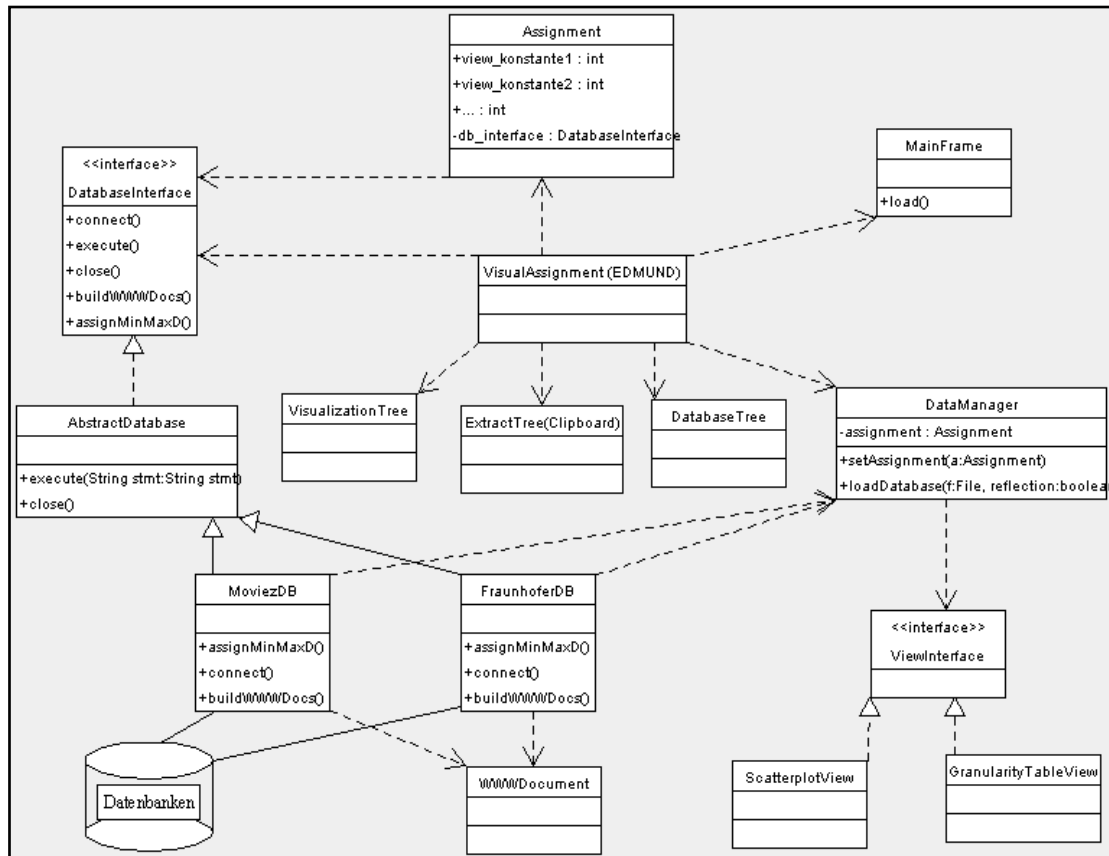


Abbildung 8: Datenmodell UML Diagramm

Ich möchte an dieser Stelle nochmals auf die Praktikumsarbeit von Herrn Gundelsweiler verweisen, der weitere Funktionen des Datenmodells implementiert hat. Während meine Aufgabe eher in der Anbindung an Datenquellen und im Aufbau der Datenbasis bestand, war Herr Gundelsweiler für den Zugriff seitens der Visualisierungen auf diese Daten zuständig (Gundelsweiler 2003). Hier spielten weiterhin Fragen der Organisation und des Software Designs eine sehr große Rolle. Darüber hinaus wurde beinahe parallel zu diesen Arbeiten von uns beiden gemeinsam das Tool zum „Visual Assignment“ (Gundelsweiler & Memmel 2003) entwickelt.

Alle drei Bestandteile verschmolzen im Frühjahr 2003 zu einer Einheit. Deren Funktionsweise und Aufbau kann mit Hilfe der auf der beiliegenden CD enthaltenen Version von INVISIP nachvollzogen werden.

4 Anhang – Screenshots INVISIP / VisMeB



Abbildung 9: INVISIP GranularityTable in unterschiedlichen Stufen

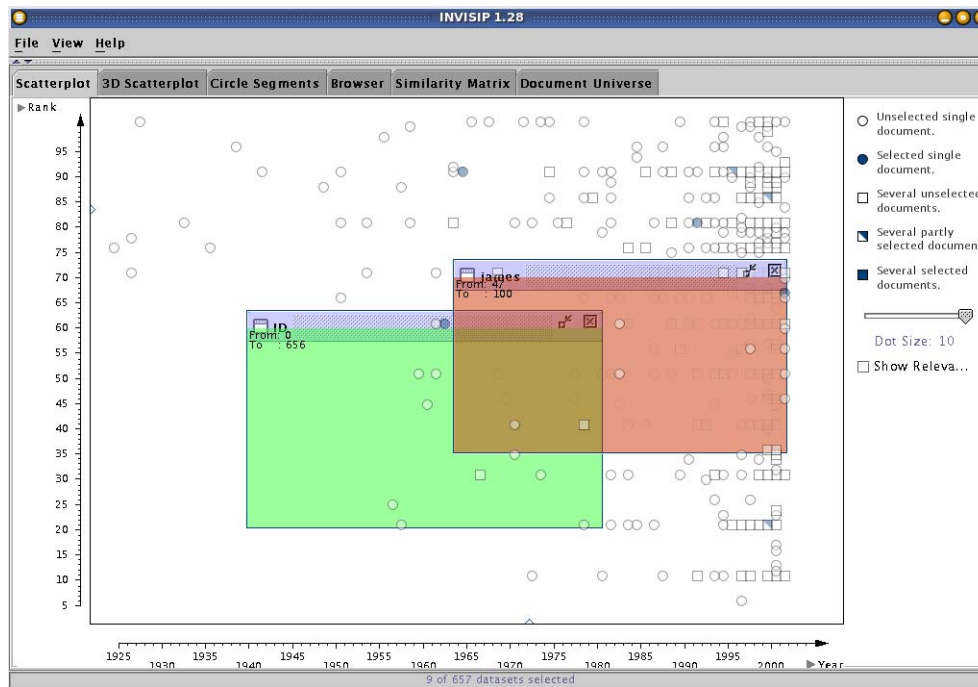


Abbildung 10: INVISIP 2D Scatterplot mit MagicLenses

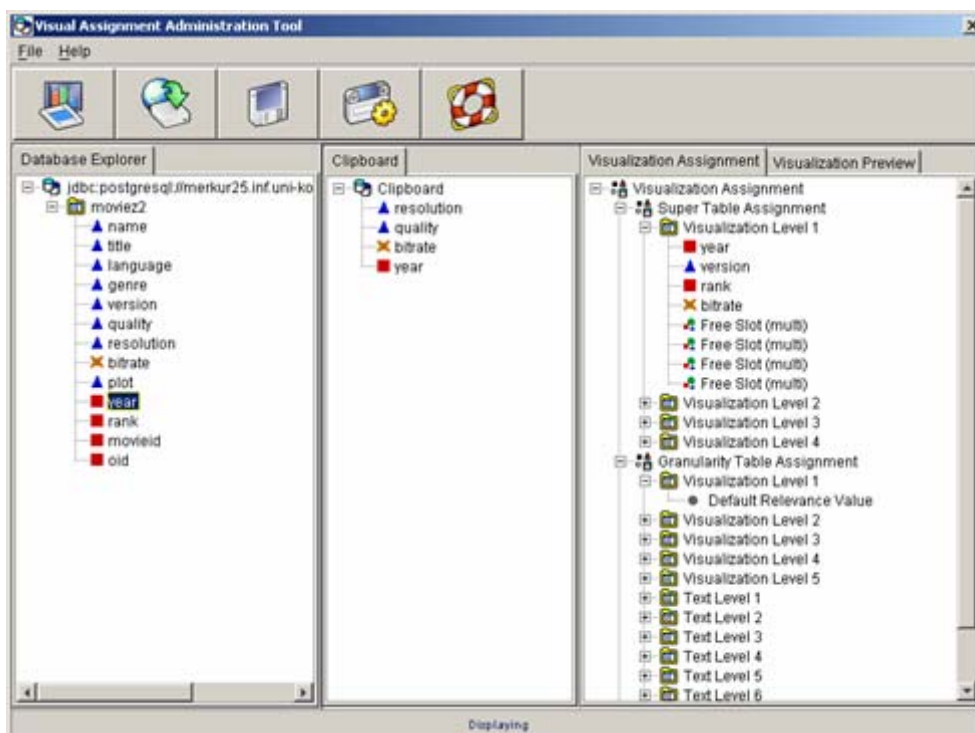


Abbildung 11: INVISIP 2D Scatterplot mit MagicLenses

5 Literaturverzeichnis

- Gamma, E; Helm, R.; Johnson, R; Vlissides John – “Design Patterns” – Addison Wesley Verlag, 1994.
- Gundelsweiler, Fredrik – „Bericht zum Master-Projektpraktikum im Rahmen des Forschungsprojekts INVISIP“ – Hausarbeit, Universität Konstanz, 2003.
- Gundelsweiler, Fredrik – „Implementation eines Scatterplots zur Visualisierung von georäumlichen Metadaten“ – Bachelorarbeit, Universität Konstanz, 2002.
- Gundelsweiler, Fredrik; Memmel, Thomas – „Projekt Edmund“ – Vorlesung „Visuelle Suchsysteme“, Hausarbeit, Wintersemester 2002/2003, Universität Konstanz, 2003.
- Mann, T. M.; Reiterer, H. – „Case Study: A Combined Visualization Approach for WWW-Search Results” - IEEE Information Visualization Symposium 1999 Late Breaking Hot Topics Proceedings
- Mann, Thomas M - “Visualization of Search Results from the World Wide Web“ – Disseratation, Universität Konstanz, Fachbereich Informatik und Informationswissenschaft, 7.10.2001.
- Mommel, Thomas – „Implementation einer tabellenbasierten zur Visualisierung von georäumlichen Metadaten“ – Bachelorarbeit, Universität Konstanz, 2002.
- Müller, F.; Klein, P.; Reiterer, H.; Eibl, M. - Visualization of Metadata using the SuperTable+Scatterplot. "Information und Mobilität", in: Hammwöhner R., Wolff C., Womser-Hacker C. , UKV Universitätsverlag Konstanz, "Schriften zur Informationswissenschaft", Konstanz, p. 147-163, "8. Internationales Symposium für Informationswissenschaft ", 2002
- Reiterer H., Mußler G., Mann T., Handschuh S. - "INSYDER - An Information Assistant for Business Intelligence" – 23st Annual International ACM SIGIR 2000 Conference on Research and Development in Information Retrieval", 2000