INVISIP

Implementation eines Scatterplots zur Visualisierung von geo-räumlichen Metadaten

Bachelorarbeit vorgelegt von Fredrik Gundelsweiler

Universität Konstanz,

Fachbereich Informatik und Informationswissenschaft

1. Gutachter (Betreuer): Herr Professor Dr. Reiterer
2. Gutachter: Herr Professor Dr. Kuhlen

Abgabetermin: bis zum 17. September 2002 in drei gebundenen Versionen

Danksagung

Diese Arbeit entstand durch die Möglichkeit am INVISIP-Forschungsprojekt mitzuarbeiten. Deshalb möchte ich allen danken, die sich dafür eingesetzt haben und dieses Projekt letztendlich ermöglicht haben.

Ich möchte mich auch bei Herr Prof. Dr. Harald Reiterer bedanken, der mich zu jeder Zeit unterstützt hat. Vielen Dank auch an die Mitarbeiter seines Lehrstuhls für die konstruktiven Vorschläge, die Kritik und die angenehme Zusammenarbeit.

Außerdem möchte ich allen Dozenten und Lehrbeauftragten der Universität Konstanz danken, die mir ermöglicht haben meinen jetzigen Wissensstand zu erreichen.

Ich möchte mich auch bei meinen Kommilitonen bedanken, die mit mir das Studium bewältigt haben. Besonders herzlicher Dank geht dabei an meinen Mitstudenten Thomas Memmel, der mit mir in sehr guter Teamarbeit dieses Studium gemeistert hat. Ohne ihn hätte das Studieren wohl nur halb so viel Spaß gemacht.

Bedanken möchte ich mich auch bei Jutta Henßler, die mich in besonderer Weise bei meinem Studium und den zu meisternden Hürden unterstützt hat.

Vielen Dank.

Zusammenfassung (Deutsch)

Innerhalb des EU-Rahmenprojektes INVISIP (**In**formation **Vi**sualization For **Si**te **P**lanning) wurde der Prototyp eines Scatterplots entwickelt und implementiert. Nach einer kurzen Einleitung in die Materie, wird das vorherige Projekt INSYDER (**In**ternet **Sy**stéme **De R**echerche) und dessen Ziele kurz vorgestellt. Die Ergebnisse der Evaluation des INSYDER-Prototypen führen zu einigen Redesignvorschlägen, die im neuen INVISIP-System umgesetzt werden. Nach der Vorstellung von INVISIP werden die theoretischen Grundlagen dieser Arbeit umrissen. Die Grundlagen umfassen die Visualisierung von Daten und Forschungsansätze zum Scatterplot und zur enthaltenen Magic Lens. Der Kern dieser Arbeit widmet sich der visuellen Oberfläche des Scatterplots und dessen Implementierung in der Programmiersprache Java. Die entwickelte Visualisierung wird mit der Hilfe von Bildern erklärt. Grundlegende Konzepte und das INVISIP-Datenmodell werden im Anschluss besprochen. Zum Ende des vierten Kapitels wird die Implementation des Scatterplots und der Kommunikationsschnittstellen erläutert. Der letzte Teil der Arbeit stellt die aufgetretenen Probleme zusammen, die auch in einer vorläufigen Evaluation bestätigt wurden. Die gefassten Erkenntnisse und ein Ausblick bilden das Ende der Arbeit.

Abstract (English)

Within the EU project INVISIP (**In**formation **Vi**sualisation For **Si**te **P**lanning) a prototype of a scatterplot was developed and implemented. The first part of this work is an introduction to the field of the project. After this, the INSYDER (**In**ternet **Sy**stéme **De R**echerche) project and its goals are presented. The INSYDER system was evaluated and the results are the basis for the new development of the INVISIP framework. After the explanation of INVISIP, a summary of the theoretical basics is given. These basics include the visualization of spatial data and some research approaches concerning the scatterplot and the magic lens. The main part of this work is about the visual user interface of the scatterplot and its implementation in the programming language Java. The developed visualization is explained with pictures and the basic concepts are presented afterwards. At the end of the fourth chapter the implementation of the scatterplot and the communication interfaces are explained. The last part deals with known problems, a provisional evaluation and an outlook to the future of the project.

1 Motivation

In der heutigen Zeit müssen Menschen in Führungspositionen wichtige Entscheidungen treffen, die nicht nur sie selbst betreffen. Oft wird über den Fortschritt oder den Untergang ganzer Unternehmen entschieden, manchmal sogar über Leben und Tod. Die Entscheidungen werden aber nicht einfach so getroffen, sondern resultieren aus den vorhandenen Informationen. Der Grund für Fehlentscheidungen ist heute meistens falsche Information, ein Mangel an Informationen oder eine mangelhafte Darstellung der verfügbaren Informationen.

Um Daten zu verarbeiten benutzt der Mensch schon seit langer Zeit nicht nur seine eigenen mentalen Möglichkeiten, sondern auch physische Hilfsmittel. Eine einfache Art der Unterstützung der menschlichen Informationsverarbeitung ist zum Beispiel die Nutzung von Papier und Bleistift zur Lösung einer Rechenaufgabe. Mit diesen Hilfsmitteln lösen Menschen komplexere Rechenaufgaben um ein vielfaches schneller, als nur mit ihrem Gehirn. Ein weiteres bekanntes Beispiel für die Unterstützung der menschlichen Wahrnehmung ist der U-Bahnplan der Stadt London. Dieser hilft den Menschen ein mentales Modell der U-Bahn aufzubauen und die richtigen Fahrtentscheidungen zu treffen. Dies ist zwar ein recht einfaches, aber effizientes Hilfsmittel. Mittlerweile sind die meisten U-Bahnen großer Städte mit ähnlichen Fahrplänen ausgestattet.

Ein tragisches Beispiel für eine Fehlentscheidung ist der Absturz des Spaceshuttles Challenger [5]. Die Entscheidung, ob das Raumschiff an einem kalten Tag gestartet werden soll oder nicht, musste getroffen werden.

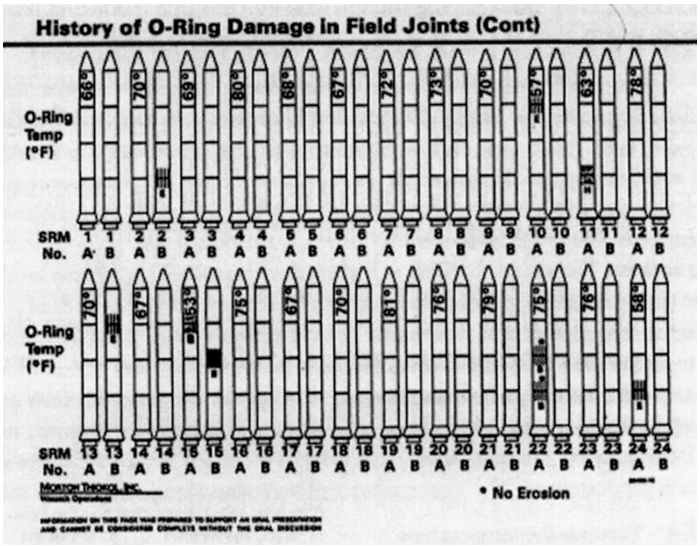

Abb. 1.1: Diagramm zur Überprüfung der Dichtungsringe, entnommen aus [5]

Diese Entscheidung hing vom Beschädigungsgrad der Dichtungsringe beim Start bestimmter Antriebsraketen ab. Wenn die Temperatur zu niedrig ist, werden die Dichtungsringe beschädigt und es kommt zu Fehlfunktionen. Abbildung 1.1 zeigt ein Diagramm, das benutzt wurde um die wichtige Entscheidung eines Starts zu treffen. Auf der Grafik sind alle Raketen in der Reihenfolge ihres Einsatzes abgebildet. Die Temperatur der Dichtungsringe ist nur in Textform in den Raketenspitzen angegeben. Der Grad des Schadens wird nicht offensichtlich und eine Legende fehlt auch. Auf der Basis dieser Grafik ist die Fehlererkennung kaum möglich.

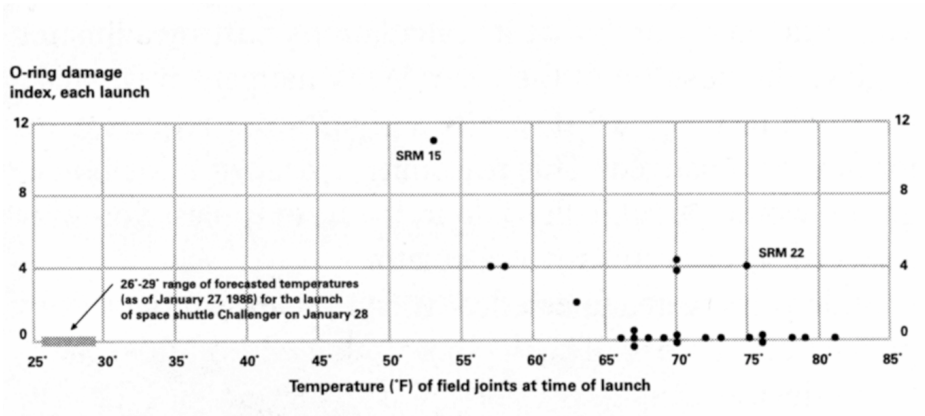

Abb. 1.2: alternative Visualisierung zur Überprüfung der Dichtungsringe, entnommen aus [5]

Wäre die Abbildung 1.2 benutzt worden, hätte der Start wahrscheinlich nicht stattgefunden. In dieser Grafik wird eine Art Scatterplot zur Visualisierung der Daten verwendet. Es ist sofort ersichtlich, dass ein Start der Raketen bei einer Temperatur unter 65° F, schwere Schäden verursacht. Hier führte die falsche Entscheidung zum Absturz des Raumschiffs und zum Tod der Besatzung. Die Visualisierung von Informationen ist elementar für die richtigen Entscheidungen. Es ist aber auch möglich, dass es aufgrund einer falschen Visualisierung zu Fehlentscheidungen kommt. Es gibt also richtige und falsche Wege, Daten in einem gewissen Aufgabenkontext zu visualisieren. Durch die Entwicklung der Computer, wurden mit der Zeit auch sehr komplexe Hilfsmittel entwickelt. Das Global Positioning System (GPS) und Navigationssysteme in Fahrzeugen sind zwei Beispiele für moderne, komplexe, entscheidungsunterstützende Hilfsmittel.

Werkzeuge zur Entscheidungsunterstützung werden auch in Unternehmen benötigt. Zum Beispiel im Managementbereich wird mit verschiedenen Arten der Informationsvisualisierung gearbeitet. Diagramme und Tabellen werden von Managern genutzt, um in der Unternehmenswelt die richtigen Entscheidungen zu treffen. Auch an

der Börse werden solche Hilfsmittel angewendet. Broker nutzen zum Beispiel die Portfolioanalyse, um Unternehmen in den Markt einzuordnen und auf dieser Basis Entscheidungen zum Kauf oder Verkauf von Unternehmensanteilen zu treffen.

Diese Arbeit befasst sich mit der Visualisierung von räumlichen Metadaten durch einen Scatterplot. Die Basis dieser Arbeit ist das INSYDER-Projekt. Dies ist eine visuelle Suchmaschine für das Internet. Der neue INVISIP-Framework soll im Gegensatz zu INSYDER ein System zur Visualisierung aller möglichen Arten von Daten zur Verfügung stellen. Bestandteil des Frameworks soll der in dieser Arbeit entwickelte Scatterplot sein. Bei der Entwicklung dieses Systems sind zunächst aber nicht alle Arten von Daten relevant. Der Fokus liegt auf der Anwendung des Systems zur Standortplanung.

2 Einleitung

Die Standortplanung ist vor allem für Gemeinden und Bauunternehmen von Relevanz. Es müssen Daten erhoben und Messungen durchgeführt werden, um die besten Standorte für kommerzielle oder private Gebäude, Straßen und andere Objekte zu ermitteln. Nach Verarbeitung und Auswertung dieser oft komplexen Informationen, können konkrete Entscheidungen leichter oder überhaupt erst gefällt werden. Zur Planung und Entscheidungsunterstützung werden aber Werkzeuge benötigt, die die Vielfalt der Informationen entsprechend aufbereiten und übersichtlich darstellen können. Ohne solche Werkzeuge ist man kaum in der Lage, die notwendigen Informationen sinnvoll zu organisieren und die richtigen Entscheidungen zu treffen.

2.1 Informationsquelle Internet

Das Internet hat sich in der heutigen Zeit als Informationsquelle etabliert. Alle möglichen Gruppen von Menschen und Organisationen benutzen das Internet als Recherchеinstrument. Vor allem bei Unternehmen hat Information, als wirtschaftliches Gut, eine große Bedeutung erlangt. So wird es auch für kleinere Unternehmen wichtig, Informationen über das Internet zu beschaffen. Oft hängen damit neue Märkte, Strategien oder Anpassungen an neue Strukturen zusammen. Ein Beispiel wäre die Einführung von E-Commerce in ein Unternehmen. In der heutigen Zeit werden die meisten Unternehmen ohne eine gute Informationsinfrastruktur und ohne eine computergestützte Zusammenarbeit mit den Marktteilnehmern vom Markt gedrängt [4]. Die Beschaffung von Informationen über das Internet funktioniert heutzutage hauptsächlich oder ausschließlich mittels der bekannten Internetdienste. Zu diesen zählen beispielsweise Suchmaschinen, Online-Thesauri, Informationsforen, Agenten, Informationsassistenten und andere Dienste. Aber auch Informationen, die nicht aus dem Internet stammen müssen in Organisationen be- und verarbeitet werden.

2.2 Erhebung und Visualisierung von Daten

Eine besondere Bedeutung haben die GIS (Geoinformationssysteme) in den letzten fünf bis zehn Jahren erlangt. Daten über unsere reale Welt werden auf die verschiedensten Arten erfasst, um die Menschheit bei der Forschung oder bei praktischen Problemlösungen adäquat zu unterstützten. Durch den technischen Fortschritt, vor allem im Hardwarebereich, ist es möglich geworden sehr komplexe Sachverhalte

bzw. Probleme zu berechnen. Es können sehr viele arithmetische oder logische Operationen in relativ kurzer Zeit durchgeführt werden. Die Kluft zwischen der realen Welt und deren Abbildung in Systeme wird mit dem Fortschritt von Forschung und Technik immer kleiner.

Zur Sammlung von Daten in unserer realen Welt werden heute noch Mittel eingesetzt, die vor Jahrzehnten entwickelt und erforscht wurden. Hierzu gehören zum Beispiel Kameras, mathematische Werkzeuge, Karten und vieles mehr. Im Informationszeitalter sind wir allerdings nun in der Lage, die gesammelten Daten anders aufzubereiten und zu speichern. Dies geschieht heute mit Hilfe der Computer. Messungen können automatisch durchgeführt werden und die Daten können in Datenbanken gespeichert werden. Bei der Erhebung von Daten in der realen Welt, werden immer Informationen unberücksichtigt bleiben, da die Komplexität der realen Welt nicht erfasst werden kann oder da dies zumindest zu aufwendig wäre. Es werden also nur die Daten gemessen und erhoben, die für relevant gehalten werden. Bei der Speicherung entsteht so ein Abbild eines Ausschnitts der realen Welt, wie sie zu einem bestimmten Zeitpunkt existierte. Den Ablauf und die Prozesse zeigt die folgende Grafik.

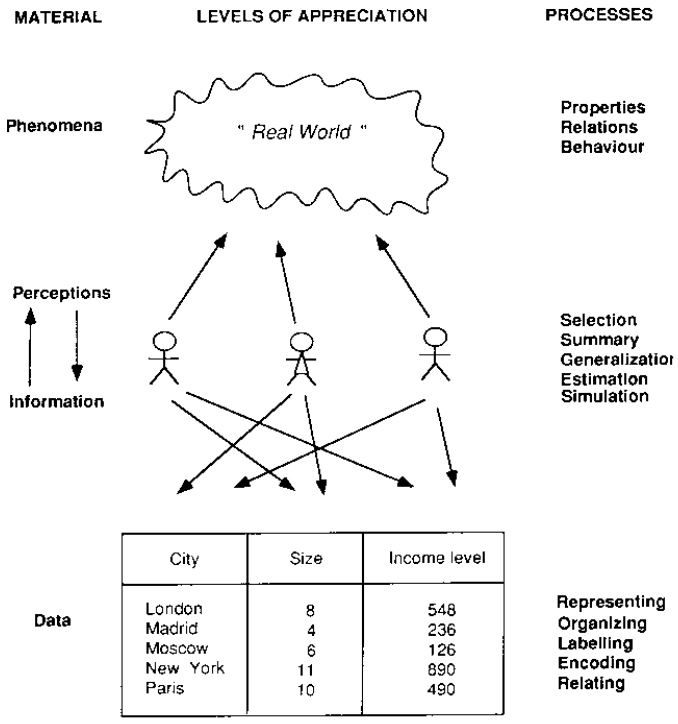

Abb. 2.1: entnommen aus [2]

Ein Teil der realen Welt mit bestimmten Eigenschaften, Beziehungen und Verhaltensweisen wird materiell als Phänomen dargestellt. Wir nehmen die Informationen wahr und ver- bzw. bearbeiten sie über verschiedene Prozesse. Die Prozesse bestehen aus Selektion, Zusammenfassung, Verallgemeinerung und anderen. Die resultierenden Informationen werden dann zum Beispiel in einer Datenbank abgespeichert. Die Daten gilt es dann zielgruppengerecht aufzubereiten und zu präsentieren. Im Beispiel sind die Objekte der realen Welt einige Großstädte. Die Daten, die repräsentativ gespeichert werden, sind der Name der Großstadt, die Größe und das Einkommenslevel. Hier ist eine einfache Tabelle als Repräsentationsform gewählt worden. Es würde sich jedoch auch ein Kreisdiagramm oder andere Möglichkeiten zur Visualisierung der Daten eignen. Es kommt auf den Anwendungskontext und die Benutzer an, was bevorzugt werden sollte.

Die Gebiete der Informationsvisualisierung und des Usability Engineerings werden heute für GIS und ähnliche Projekte immer bedeutender. Früher konnten Daten nur textbasiert dargestellt werden. Heute haben wir die Möglichkeiten, komplexe grafische Visualisierungen zu entwickeln, um Daten anschaulicher zu machen und sie dem Nutzungskontext bzw. dem Benutzer anzupassen.

Ein Problem der grafischen Visualisierung ist die Bewahrung der Übersichtlichkeit und Einfachheit. Es ist möglich, dass die Visualisierung zu komplex wird und der Anwender diese nicht mehr auf Anhieb versteht [3]. Eine einfache textbasierte Darstellung kann dann effektiver und effizienter sein als die grafische Darstellung. Die Daten sollen mit Hilfe der Visualisierung leichter und vor allem auch schneller vom Menschen verstanden und aufgenommen werden. Hierzu werden Metadaten verwendet. So müssen die möglicherweise auch komplexen Daten nicht mehr in ihrem vollen Umfang angezeigt werden, sondern nur noch gewisse extrahierte, wichtige, relevante Informationen über die einzelnen Daten oder Objekte.

2.3 Metadaten

Metadaten sind Daten über Daten. Sie beschreiben bestimmte Daten in allgemeinerer Form, so dass ein Benutzer die Übersicht behält und leichter mit den Daten arbeiten kann. In INVISIP sollen Daten über Objekte visualisiert werden, um das Finden und die Arbeit mit den Daten zu erleichtern. Die Testdatensätze mit denen bei der Entwicklung des INVISIP-Projekts gearbeitet wird, sind Datensätze aus dem alten

INSYDER. Sie enthalten Informationen über Webdokumente, sind also Daten über Daten. Enthalten sind beispielsweise die Größe des Dokuments in Kilobyte, die Anzahl Wörter, die URL und viele mehr.

Eine wissenschaftliche Definition von Metadaten gibt Diann Rush-Feja:

„Metadaten als Informationen über Daten die in einer Form gehalten werden, dass sie die Recherche, das Retrieval und die Nutzung der Primärdokumente ermöglichen, erleichtern und ggf. bestimmen.“ [Z1].

Weitere Definitionen ähneln sich zum großen Teil. Eine sehr einfache Definition liefert die Dublin Core Metadata Initiative. Dort werden Metadaten als „strukturierte Daten über Daten“ [I7] bezeichnet. Ein Objekt oder eine Ressource werden über Metadaten beschrieben. Wenn man sich als Objekt ein Buch vorstellt, dann wären geeignete Metadaten zur sinnvollen Beschreibung dieses Objektes Autor, Titel, Erscheinungsjahr, ISBN und beliebige mehr. Der Begriff „Metadaten“ ist zwar relativ neu, das zugrunde liegende Konzept aber ist schon älter. Es kam schon früher bei Bibliotheken vor, als vorhandene Bücher über Karteikarten beschrieben wurden. Schon damals diente das Konzept dazu, gesuchte Bücher schneller zu finden. Einige hundert nach bestimmten Kriterien sortierte Karteikarten lassen sich einfacher, schneller, effektiver und effizienter durchsuchen als einige hundert Bücher, die in Regalen stehen.

Eine weitere Definition kommt von Tim Berners-Lee, dem Vorsitzenden des World Wide Web Consortiums (W3C): „Metadaten sind maschinenlesbare Informationen über elektronische Ressourcen oder andere Dinge“ [Z2]. Ein großes Problem des Internets ist, dass es keinen einheitlichen Standard zur Beschreibung von Dokumenten gibt. Als Standard existiert zwar das Dublin-Core-Metadata-Element-Set [I7], es wird bei Internetdokumenten jedoch äußerst selten verwendet.

Metadaten werden also bei der Suche nach Dokumenten verwendet. Sie sind ein wesentlicher Grundstein für Retrievalsysteme. Bei der Entwicklung des INVISIP-Projekts spielen hauptsächlich geo-räumliche Metadaten eine Rolle. Das bedeutet, dass die Daten einen Raumbezug haben und Beschreibungsmerkmale über Geodaten beinhalten. Es geht also um die Visualisierung der Metadaten über Geodaten, die einen Raumbezug haben.

2.4 Information Retrieval Systeme

Systeme, die das Auffinden von Informationen/Dokumenten erleichtern und damit auch das Arbeiten, nennt man Information Retrieval Systeme (IRS). Von einer Vielzahl an Dokumenten sollen die für den Benutzer relevanten Dokumente gefunden werden. Ohne Hilfe würde man wahrscheinlich alle Dokumente durchlesen müssen, um diejenigen mit den relevanten Informationen zu finden. Ein IRS übernimmt die Aufgabe, nach den relevanten Dokumenten zu suchen. Da aber für das vollständige, korrekte Auffinden ein semantischer Bezug vom System verstanden werden muss, können heutige Computer diese Aufgabe nicht korrekt lösen. Die Künstliche Intelligenz (KI) ist, zumindest heute, noch nicht in der Lage die Semantik von Sprachen abzubilden und dem Computer verständlich zu machen. Es müssen also relevante Informationen aus den Dokumenten extrahiert werden. Das IRS kann jetzt nach bestimmten Kriterien auf der Dokumentenmenge suchen und ist in der Lage relevante Dokumente auszufiltern. Durch die sprachliche Komplexität und die Semantik werden dabei allerdings Grenzen gesetzt, so dass nicht immer alle relevanten Dokumente gefunden werden können. Wichtig ist trotz allem Effektivität und Effizienz des Systems, da der Benutzer schnell an seine gewünschten Informationen kommen soll. Mittels Precision und Recall lässt sich die Effizienz eines IRS bestimmen. Precision bezeichnet die Genauigkeit der Suche. Der Wert gibt das Verhältnis der relevanten, gefundenen Dokumente zu allen gefundenen Dokumenten an. Recall hingegen zeigt die Vollständigkeit einer Suche auf. Für diesen Wert wird das Verhältnis aller gefundenen, relevanten Dokumente zu allen im System vorhandenen, relevanten Dokumenten berechnet.

Eine Definition von IRS ist im folgenden aufgeführt.

„Die Informationen, die mit Retrievalsystemen verarbeitet werden, bestehen aus Dokumenten. Information Retrieval beschäftigt sich demnach mit der Repräsentation und Speicherung von und dem Zugriff auf Dokumente oder Dokumentstellvertreter. Ausgangspunkt ist der natürlichsprachige Text der Dokumente, Auszüge aus diesem Text oder Zusammenfassungen (Abstracts). Richtet man an ein Retrievalsystem eine Suchanfrage, so erhält man als Ergebnis eine Menge von Referenzen. Diese Referenzen verweisen den Nutzer des Retrievalsystems auf potentiell relevante Veröffentlichungen.“ [1]

2.4.1 Technischer Aufbau

Ein IRS besteht aus drei Komponenten, der Datenbank, dem Anwenderprogramm und dem Computer. Das Information Retrieval System kann auf bestimmte, gezielte Anfragen die gewünschten oder zugeordneten Informationen liefern.

Eine Datenbank ist ein System zur Speicherung von nach gewissen Prinzipien strukturierten Daten. Sie besteht aus einer Datenbasis und einem Verwaltungsprogramm. Um in der Datenbasis abgelegt werden zu können müssen die Daten in eine bestimmte Form gebracht werden. Hauptzweck der Datenbank ist die Daten zu verwalten und mittels Anfragen, nach bestimmten Kriterien gefilterte Daten, an den Anwender zurückzuliefern. Hierzu werden als Verwaltungsprogramme sogenannte Datenbankmanagementsysteme (DBMS) verwendet. Sie werden auf die Datenbasis aufgesetzt und steuern die Anfragen und das Arbeiten mit den Daten (z.B. die Benutzerverwaltung). Die zu steuernden Operationen sind Ändern, Einfügen, Lesen und Löschen.

Das Anwenderprogramm ist ein Computerprogramm, mit dem der Benutzer auf den Daten der Datenbank arbeitet und diese damit für den Anwender nutzbar macht. Jedes Anwenderprogramm kann die Daten unterschiedlich bearbeiten und jeweils auch unterschiedlich darstellen.

Die folgende Abbildung zeigt das grobe Schema eines solchen Systems.

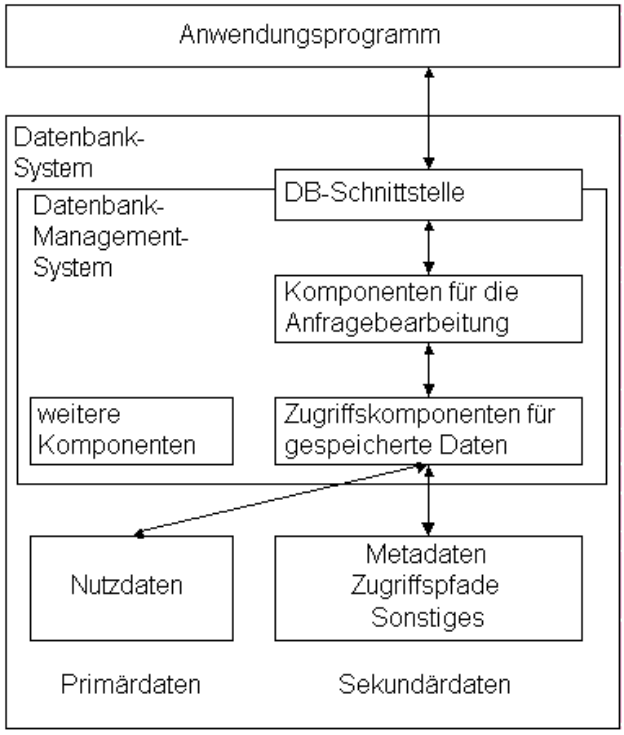

Abb. 2.2: vgl. Skript WS 2001 Marc Scholl

2.4.2 Beispiele für IRS

Ein für Internetbenutzer alltägliches Beispiel für Information Retrieval Systeme sind Internet-Suchmaschinen. Bekannte Beispiele sind unter http://www.google.de/, http://de.altavista.com/ und unter http://www.metager.de/ zu finden. Metager ist eine Beispiel für eine Metasuchmaschine. Die Suchanfrage des Benutzers wird an einige andere Suchmaschinen weitergegeben und die Ergebnisse werden von Metager aufgelistet. Auch die Suchmaschine INSYDER, das Basisprojekt für diese Arbeit, ist ein IRS. Weitere bekannte Projekte von Ahlenberg und Shneiderman (z.B. der Filmfinder [15]) werden in Kapitel 3.3 noch vorgestellt.

InGeo IC [I1] ist ein Metadateninformationssystem für Geodaten. Es handelt sich hierbei um ein Geoportal zur Suche jeglicher Art von Metadaten. Nach einer Registrierung kann das System über den Internetbrowser benutzt werden.

2.5 Forschungsprojekte und Prototypen

Grundlegend für diese Arbeit ist das Forschungsprojekt INSYDER. Der Scatterplot ist ein Hauptbestandteil des neuen INVISIP-Projektes, das momentan an der Universität Konstanz mitentwickelt wird. Im folgenden sollen die beiden Projekte vorgestellt und erläutert werden, sowie deren Einsatz und Ziele aufgezeigt werden.

2.5.1 INSYDER

Das Projekt INSYDER (**In**ternet **Sy**stème **de R**echèrche) [10, 11, 12] ist ein EU-Forschungsprojekt. Es wurde innerhalb des EU-Rahmenprogrammes ESPRIT Programm Software Technologien gefördert. Die Entwicklung begann im September 1998 in der Arbeitsgruppe Informationssysteme. Die länderübergreifende Zusammenarbeit bei der Entwicklung des EU-Projektes drückt sich durch die Zusammensetzung der Partner aus. Italien, Frankreich, England und Deutschland waren die beteiligten Länder. Ziel war die Entwicklung eines Informationsagenten für Business Intelligence Systems.

Eine Definition laut der Gartner Group von 1989:

„Business Intelligence is the process of transforming data into information and, through discovery into knowledge.” [Z3]

INSYDER ist eine Internet-Suchmaschine, die Daten über Dokumente aus dem World Wide Web sammelt und zu Informationen verarbeitet. Diese Informationen

werden für den Benutzer visualisiert, so dass dieser bei seiner Aufgabe unterstützt wird.

Ziele von INSYDER

Der INSYDER Prototyp wird zur Visualisierung und Evaluierung von Suchergebnissen aus dem World Wide Web verwendet. Kleinen und mittelständischen Unternehmen sollten relevante, für sie wichtige Informationen aus dem Internet zugänglich gemacht werden. Dabei soll den Entscheidungsträgern eine intuitiv gestaltete Benutzeroberfläche zur Verfügung stehen. Die Vorgehensweise zur Ermittlung relevanter Dokumente im Internet sieht ungefähr so aus. Der Anwender gibt dem System einige relevante Suchbegriffe. Daraufhin gibt INSYDER, wie eine Metasuchmaschine, die Suche weiter an verschiedene Suchmaschinen. Diese durchsuchen dann das World Wide Web (WWW) nach Daten bzw. Internetdokumenten, die die gewünschten Begriffe enthalten. Da der Anwender nicht mit zu vielen Informationen überhäuft werden soll, werden die gefundenen Informationen dann vom System nach bestimmten Kriterien gefiltert. Ein Filter ist zum Beispiel auch das Benutzerprofil des Anwenders. Mit diesem kann das System entscheiden, ob die gefundenen Daten für den Benutzer grundsätzlich überhaupt von Relevanz sind. Die Anwender können sich die Daten nach bestimmten Relevanzwerten, wie z.B. Sprache und Dokumentengröße, anzeigen lassen. Ein Ziel von INSYDER ist, die Informationen benutzergerecht zu visualisieren.

Design von INSYDER

Um die Abbildung der Daten auf bestimmte visuelle Formen gut und aussagekräftig zu machen, wurde festgestellt, welche Gesichtspunkte bei der Gestaltung der Benutzeroberfläche zu beachten sind. Zu Beginn des Projektes wurde eine Feldstudie durchgeführt, um den Nutzungskontext nach ISO 9241 zu kennen. Außerdem kamen der „human-centered design approach“ nach ISO 13407 und einige allgemeine empirische Studien zur Geltung. Als Ergebnis entstand das „5-T Environment“. Die fünf T’s bestehen aus den Datentypen (**T**ype of Data), der Zielgruppe (**T**arget users), der typischen Arbeitsaufgabe (**T**ypical Task), den technischen Umgebungsbedingungen (**T**echnical Environment) und dem **T**raining. Die Datentypen des INSYDER Projektes waren multidimensionale Daten in Form von Metadaten über Webdokumente. Als typische Benutzer wurden Bau- bzw. Konstruktionsexperten festgestellt. Außerdem noch Anfänger, die das Internet kennen, ein begrenztes Wissen über Suchma-

schinen haben und die bei der Informationssuche und der Benutzung von Retrievalsystemen noch wenige Kenntnisse haben. Typische Aufgaben sind die Situationen in denen ein System zur Informationssuche benötigt wird. Wichtig sind in diesem Zusammenhang die Informationsquellen (meistens Webseiten), die Informationsbedürfnisse der Benutzer (z.B. Informationen über neue Technologien oder Mitbewerber) und die vom Anwender erwartete Funktionalität. Die technische Umgebung ist der Computer mit einer begrenzten Prozessorleistung, Arbeitsspeicher und Bildschirmgröße.

Das Design von INSYDER entstand auf der Basis einer Feldstudie der Benutzer, ihrer Aufgaben und der technischen Umgebung. Dazu gehören INQUIRY von Georgia Tech [7], die Tile Bars von Xerox Parc, das Filmfinder Projekt der Universität von Maryland und das Projekt Envision von Virginia Tech. Es wurde darauf geachtet, dass die Visualisierungen sich möglichst an Grafiken orientieren, die den Anwendern aus ihrem Arbeitsumfeld schon bekannt sind. Ein „multiple view approach“ wurde gewählt. Dadurch ist der Benutzer immer in der Lage aus den möglichen Visualisierungen, die passende auszuwählen. Eine Visualisierung, die INSYDER benutzt, ist der Scatterplot. Dieser beeinflusste auch den Scatterplot der in dieser Arbeit noch vorgestellt wird.

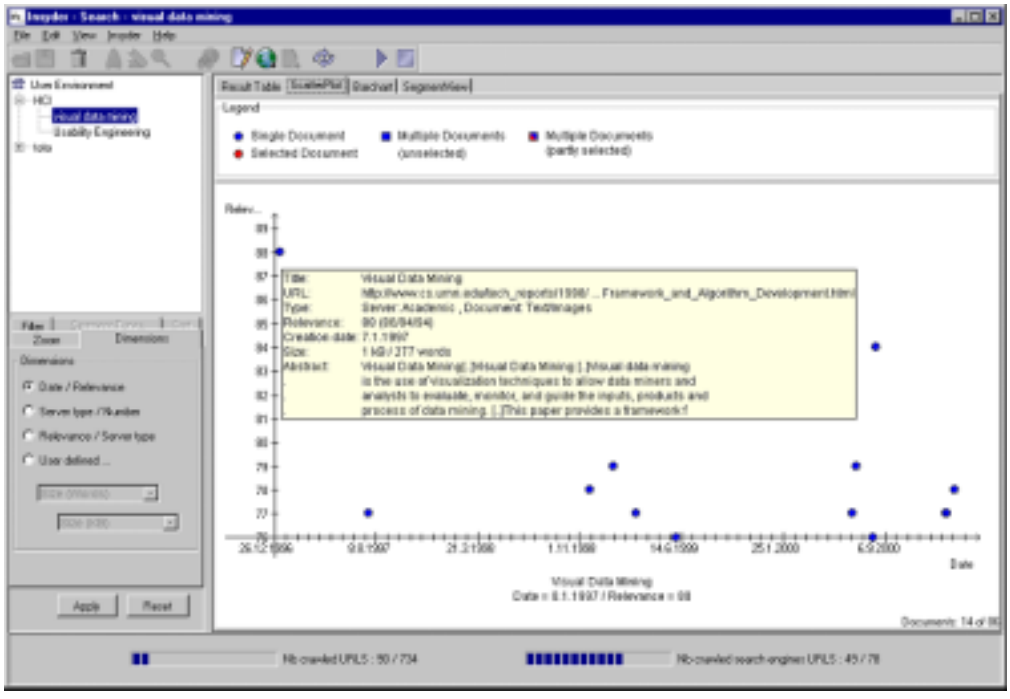

Abb. 2.3: INSYDER – Abbildung Scatterplot (Screenshot)

Man sieht hier die Achsen die mit den Werten Datum und Relevanz belegt sind. Über einen Tool Tip, wie hier im Bild, können weitere Informationen über bestimmte Dokumente eingeblendet werden. Die Dokumente sind als Punkte im Koordinatenkreuz visualisiert. Oben in der Legende sind die verschiedenen Punkttypen und deren Bedeutungen erklärt. Dokumente können selektiert sein, dann werden sie in rot dargestellt, andernfalls in blau. Das Problem, dass Dokumentenpunkte an den gleichen Stellen erscheinen können, wird dadurch gelöst, dass statt einem runden Punkt ein Quadrat als Darstellung erscheint.

Eine weitere grafische Komponente ist die Tabelle (engl.: Table). Im Split Window ist oben ein Tabelle untergebracht. Unten erscheinen, je nach Zeilenauswahl, die Dokumente selbst im Volltext.

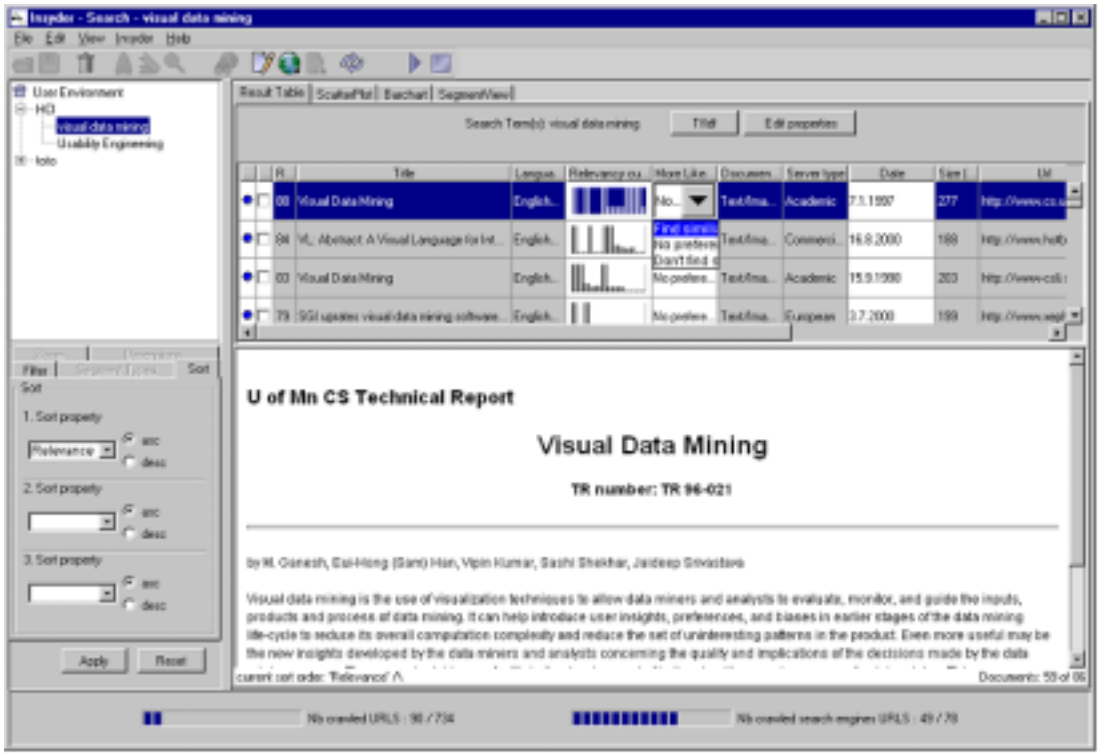

Abb. 2.4: INSYDER – Abbildung Table und Detailinformationen (Screenshot)

Die anderen Komponenten sind eine einfache Liste, Barcharts, Tilebars und Stacked Colums. Je nach Aufgabe oder persönlichen Präferenzen kann sich der Anwender für die entsprechenden Visualisierungen entscheiden.

Neben den Visualisierungen selbst, wurden auch die Interaktionen mit den verschiedenen Ansichten bedacht. Die Visualisierungen können interaktiv vom Benutzer in ihrer Größe oder sonstigen Parametern verändert werden. Diese Möglichkeit wird View Transformation genannt. Im INSYDER Projekt wurden drei verschiedene Ar-

ten von View Transformation eingesetzt. Location Probes benutzen bestimmte Orte von Visualisierungen, um zusätzliche Informationen im Kontext anzuzeigen.

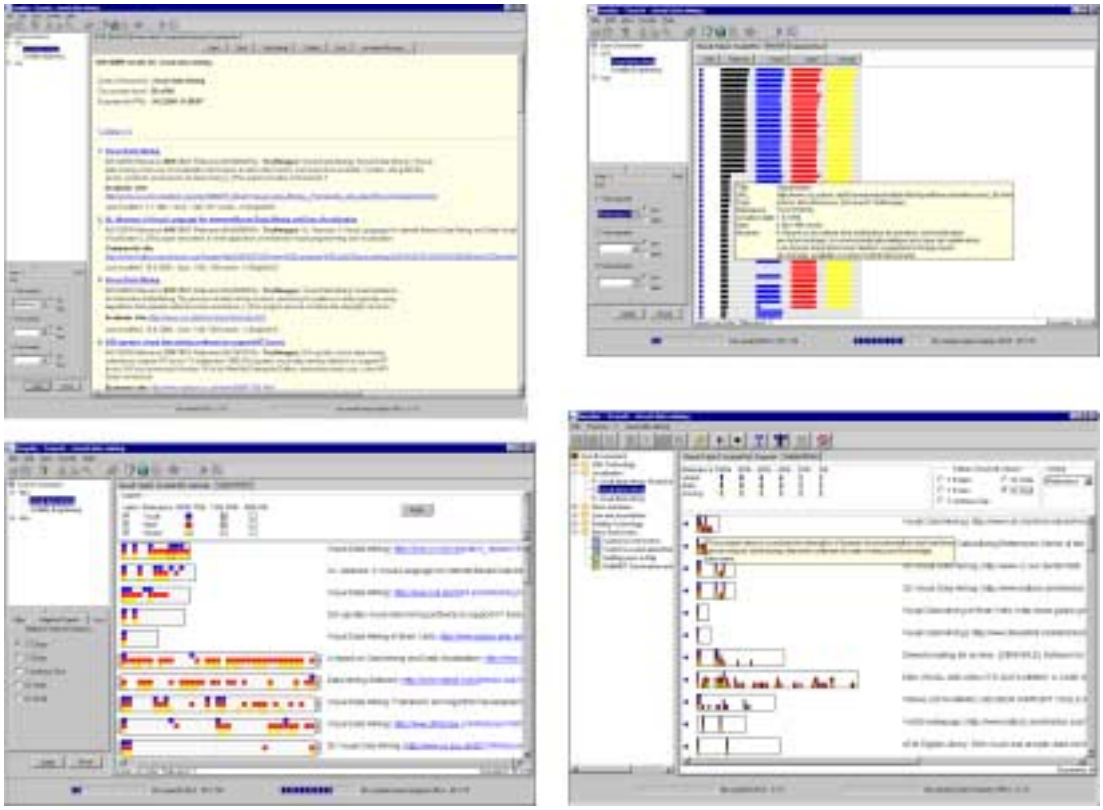

Abb. 2.5: INSYDER – Abbildungen der Liste (l.o.), Barchart (r.o.), TileBars (l.u.) und Stacked Colums (r.u.) (Screenshots)

INSYDER benutzt Tooltips und Pop-up Fenster als Location Probes. Mittels Viewpoint Controls ist es dem Benutzer möglich die Visualisierung an seine Anforderungen anzupassen. INSYDER benutzt im Scatterplot das Zooming Konzept, außerdem können die Achsen mit verschiedenen Werten belegt werden. Das Sorting ist das dritte verwendete Konzept von View Transformations. Die Darstellung der Dokumente kann mittels Klick auf den Tabellenkopf oder mit einer Drop-down Listbox nach den verschiedenen Werten sortiert werden. Alle Visuellen Komponenten benutzen die bekannten Konzepte des WIMP (Windows, Icons, Menus, Pointers) [8], wie beispielsweise die Direkte Manipulation. Außerdem werden die Konzepte des Zooming, details-on-demand und direkte Selektion verwendet, um dem Benutzer Kontrolle über das Abbilden der Daten auf die Visualisierungen zu geben.

2.5.2 INVISIP

Das Projekt INVISIP (**In**formation **Vi**sualization in **Si**te **P**lanning) [9, I2] wurde im Dezember 2001 gestartet und hat eine vorläufige Dauer von 25 Monaten. Es ist Bestandteil des IST (Information Society Technologies) Programms. In diesem Projekt geht es hauptsächlich um die Standortplanung. Ziel ist die an der Standortplanung (engl.: site planning) beteiligten Parteien, wie kommunale Ämter und Behörden, Planungsbüros, Datenerhebungsstellen, Bürger und andere Beteiligte, bei ihren Aufgaben zu unterstützten. Techniken der Informationsvisualisierung sollen genutzt werden, um Such- und Analyseaufgaben zu verbessern. Entscheidungen können auf der Basis eines existierenden Metadaten-Informationssystem für geographische Daten getroffen werden. Dieser Entscheidungsprozess soll durch Techniken der Informationsvisualisierung vereinfacht werden. Die vorgeschlagene Architektur des Systems ist generisch und kann deshalb in vielen Gebieten Anwendung finden. Die momentane Implementation ist jedoch auf die Standortplanung zugeschnitten. Ein grundlegendes Problem im Standortplanungsprozess ist die Suche nach Daten und deren Analyse. Um die Planungsobjekte zu analysieren und zu realisieren, werden speziell räumliche Daten benötigt.

INVISIP soll eine technische Plattform als Hilfe zur Verfügung stellen, um den Zugriff auf Informationen und die Bearbeitung der Daten im Standortplanungsprozess zu vereinfachen. Der Zugang zu geografischen, sowie demographischen Daten soll erleichtert werden. Im Zuge des Projektes werden verschiedene Visualisierungen entwickelt, um die Daten einfacher auszuwerten und damit Zeit zu sparen.

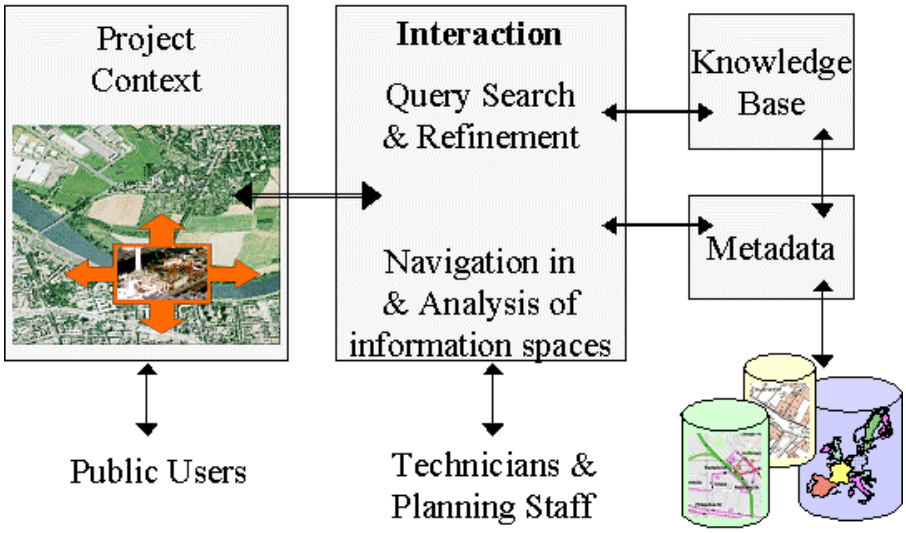

Abb. 2.6: INVISIP – Szenario (entnommen aus: www.invisip.de)

Kern des Systems ist ein Metadaten-Browser, der die Benutzer bei der Arbeit unterstützt. Suchanfragen können auf den gespeicherten Daten ausgeführt werden , um die gewünschten Informationen zu erhalten. Eine weitere Komponente des Systems ist die Knowledge Base. Sie bietet verschiedene Data Mining Techniken, mit deren Hilfe neu gesammelte Daten visualisiert und Informationen zu diesen zur Verfügung gestellt werden. Mit Hilfe der Knowledge Base sollen die Daten und deren semantische Beziehungen zueinander leichter verstanden werden. Dies alles soll eine schnelle und effiziente Standortplanung ermöglichen.

Die Entwicklung des Projekts besteht aus drei verschiedenen Phasen: In der ersten Phase wird eine Technologieanalyse und Fallstudien in den Partnerländern durchgeführt. Die Ergebnisse werden als Input in nachfolgenden Projektschritten genutzt. Die folgenden Phasen beschreiben eine iterative Entwicklung. Das Wissen aus der Analysephase wird vertieft und als Prototyp implementiert. Die wichtigste Komponente bildet der Metadaten-Browser. Parallel wird ein Analyseinstrument und eine Datenintegrationskomponente entwickelt. Die Evaluationsphasen werden nicht nur genutzt, um die technische Funktionalität des Prototypen zu überprüfen, sondern auch um dessen Benutzbarkeit zu prüfen. Alle Komponenten sollen über geeignete Schnittstellen verbunden und in den INVISIP Framework integriert werden.

Beteiligt an diesem Projekt sind das Fraunhofer IGD, die Forschungsgruppe von Herr Prof. Dr. Reiterer aus dem Fachbereich Computer und Information Science der Universität Konstanz, die tim GmbH (ein Planungsbüro), die Stadt Wiesbaden und weitere Partner[1]. Die Hauptaufgaben der Universität Konstanz liegen in der Visualisierung der Metadaten, einer visuellen Anfrageformulierung und des Designs der Benutzeroberfläche des INVISIP-Systems.

Diese Arbeit entstand als Teil des INVISIP Projektes am Lehrstuhl von Herr Prof. Dr. Reiterer. Ein Scatterplot wurde in der Programmiersprache Java, als Visualisierung für den INVISIP Framework, implementiert. Gleichzeitig wurde eine Tabellenvisualisierung für das System entwickelt [6].

---

[1] Weitere Partner sind: **Kungl Tekniska Högskolan** (KTH), eine große technische Hochschule in Schweden; **Inregia AB** eine Firma mit ca. 40 Experten in der Planung, regionalen Wirtschaftsentwicklung, Transprt und natürliche Ressourcen; **IMA**, ein Institut für angewandte Mathematik und Computerforschung; **D'Appolonia** (DAPP), eine Ingenieursfirma; die italienische Stadt **Genova**; die **University of Cracow** (UoC)

2.5.3 INSYDER Evaluation und Redesign

Diese Arbeit konnte auf der Basis von INSYDER aufbauen, da dort schon die benötigten Visualisierungen, wie z.B. der Scatterplot, benutzt wurden. Der Quellcode, die Art der Implementation und das logische Design des INSYDER Scatterplots schlossen allerdings eine Wiederverwendung aus. Die Evaluation von INSYDER in der Arbeit von Thomas Mann [13] mit ca. 40 Benutzern zeigte eine sehr unterschiedliche Bewertung der Visualisierungen. Den Benutzern wurden bestimmte Aufgaben gestellt, die sie mit Hilfe von INSYDER lösen sollten. Die Visualisierungen konnten, wenn sie nicht in der Aufgabenstellung festgelegt waren, frei vom Benutzer gewählt werden. Zur Auswahl standen die Visualisierungen HTML-List, ResultTable, Scatterplot, BarGraph und SegmentView. Für diese Arbeit sind hauptsächlich die Bewertungen des Scatterplots von Bedeutung, der in der Evaluation keine guten Werte erzielte. Der Vorteil der Darstellung von zwei Relevanzdimensionen auf einen Blick wurde von den Anwendern kaum erkannt. Bei der Benutzung kamen die wenigsten Anwender mit dem Scatterplot zurecht, allerdings wurde dieser auch am wenigsten genutzt. 35 Prozent der Benutzer fanden den Scatterplot überladen und zu komplex. Außerdem schnitt der Scatterplot als am wenigsten selbstbeschreibend ab. Deshalb war er auch nicht sehr intuitiv zu bedienen. Als Verbesserungen vorgeschlagen wurde eine Relevanzanzeige der Dokumente, obwohl die Relevanz nicht auf einer Achse als Dimension festgelegt ist. Also noch eine feste dritte Dimension. Wenn beim Scatterplot mehrere Dokumente auf einer Position liegen, wurden diese Dokumentengruppen als Quadrate angezeigt. Dies verursachte bei einigen Benutzern Verständnisprobleme. Gewünscht wurde deshalb eine Alternative für die Anzeige von Dokumentengruppen. Die Symbole sollten also überdacht werden. Als die Unterstützung der Komponente zur Lösung der Aufgabenstellung zu bewerten war, schnitt der Scatterplot auch schlecht ab. 40 Prozent der Benutzer sahen es als Zeitverschwendung an mit dem Scatterplot zu arbeiten und wählten eine Alternative zur Bearbeitung der Aufgaben. Für einige Benutzer war die Entscheidung, welche Achsenbelegung gewählt werden soll, ein Problem. Insgesamt schnitt der Scatterplot in der Benutzerzufriedenheit der Visualisierungen, bei Anfängern wie auch bei Experten, am schlechtesten ab.

Die Evaluationsergebnisse der Tabelle fielen recht gut aus, jedoch wurden hier auch einige Überlegungen zur Verbesserung angestellt [9]. In der Arbeit INVISIP –

Implementation einer tabellenbasierten Visualisierung für geo-räumliche Metadaten[6] wurde eine sogenannte Granularity Table als visuelle Alternative zur bisherigen Super Table implementiert.

In dieser Arbeit wird ein entwickelter Scatterplot vorgestellt, der an das neue INVISIP System angepasst ist. Die Verbesserungsmöglichkeiten, die oben erwähnt sind, wurden vorerst nur zum Teil realisiert. Wichtig war in erster Linie eine gut funktionierende Visualisierung, die die Möglichkeiten der Java-Technologie ausnutzt und eine gute Basis für eine Weiterentwicklung bietet. Als eine Idee zur Verbesserung des Scatterplot im Bezug auf die Erweiterung um eine dritte Dimension, wurde ein Magic Lens Filter implementiert. Diese kann wie eine Lupe über die im Scatterplot angezeigten Dokumente gelegt werden. Je nach Einstellungen der Linse werden dann die entsprechenden Dokumente aus der Anzeige ausgefiltert. Weitere Ideen zur Verbesserung und Weiterentwicklung des Scatterplots sind im fünften Kapitel aufgeführt. Die Abbildung 2.7 zeigt die beiden Visualisierungen.

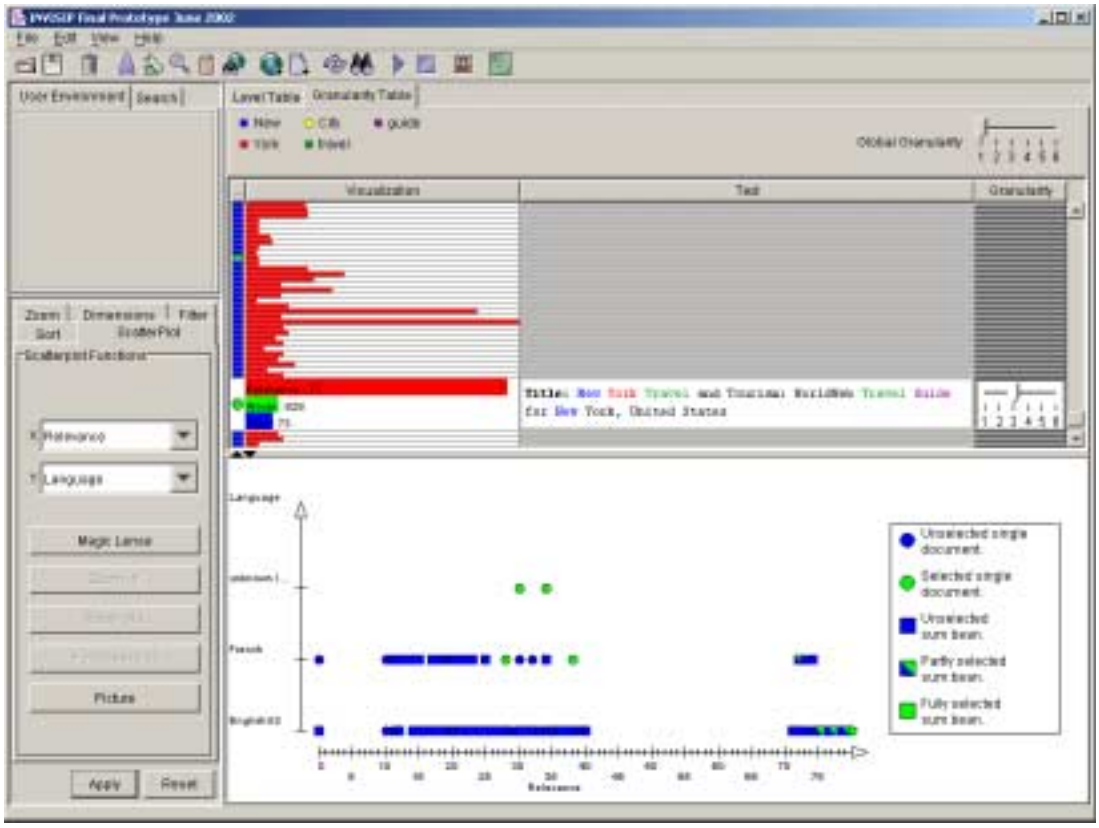

Abb. 2.7: INVISIP – Scatterplot und Granularity Table (Screenshot des fertigen Projektes)

Oben im Splitpane ist die Granularity Table und unten der Scatterplot zu sehen. Beide Visualisierungen können einzeln oder zusammen angezeigt werden, indem der vertikale Balken des Splitpanes nach oben oder nach unten bewegt wird. So ist der

Benutzer in der Lage gleichzeitig mit Scatterplot und Tabelle zu arbeiten. Er kann die Vorteile beider Visualisierungen im Bezug auf bestimmte Aufgabenstellungen nutzen, ohne ständig zwischen den Visualisierungen umschalten zu müssen. In der vergrößerten Tabellenzeile sieht man in der rechten Spalte den sogenannten Granularitäts-Slider, mit dem die Art und Menge der angezeigten Informationen über das Dokument in jeder Zeile reguliert werden kann. Außerdem wird der Titel eines selektierten Dokumentes angezeigt. Unten im Scatterplot werden die Dokumente, gemäß der Legende, als Punkte angezeigt. Die x-Achse des Scatterplots ist mit der Dimension Gesamtrelevanz (engl.: relevance) belegt. Auf der y-Achse ist die Sprache (engl.: language) der Dokumente abgetragen. Insgesamt sind hier 600 Dokumente visualisiert. Ein weiterer Gesichtspunkt ist die Kommunikation zwischen der Granularity Table und dem Scatterplot. Im Bezug auf die Benutzbarkeit und den Mehrwert ist die Frage zu klären, welche Mehrwerteffekte erzielt werden können, indem die beiden Komponenten sich gegenseitig beeinflussen.

3 Grundlagen und Theorie

3.1 Visualisierung von geo-räumlichen Metadaten

Der entwickelte Scatterplot soll zur Visualisierung geo-räumlicher Metadaten eingesetzt werden. Wie im ersten Kapitel beschrieben, sind geo-räumliche Metadaten vereinfachte Abbildungen von Objekten mit einem Raumbezug. Geo-räumliche Daten sind meistens mehrdimensional. Dabei sind nicht nur die uns bekannten Dimensionen des Raumes oder die Zeitdimension gemeint. Alle Eigenschaften der Objekte der realen Welt können jeweils in einer Dimension festgehalten werden. Also wären die Dimensionen eines Steines zum Beispiel seine Abmessungen, sowie Farbe, Gewicht, Oberflächenbeschaffenheit, usw.. Bei der Darstellung von mehreren Dimensionen gleichzeitig, wie im Scatterplot, müssen die Skalenniveaus sinnvoll kombiniert werden, da die Lage eines Objektes von beiden Dimensionen abhängt. Dies ist teilweise schwierig, da nicht immer ein logischer Zusammenhang der Dimensionen gegeben ist. Im Gebiet der Geowissenschaften wurden die wichtigsten Eigenschaften und Bezüge von Geodaten festgelegt. Geoobjekte sind durch ihren Raumbezug definiert. Dieser Raumbezug besteht hauptsächlich aus der Geometrie der Objekte, d.h. deren Lage im Raum. Ein Beispiel ist die genaue Beschreibung der Lage eines Stadtzentrums per GPS Koordinaten. Geoobjekte können aber auch Beziehungen untereinander haben. Angenommen zwei Städte wären jeweils als Geoobjekte in einem Modell abgebildet. Beide Städte haben bestimmte Eigenschaften wie Ausdehnung, Einwohnerzahl und andere. Zusätzlich kann eine Beziehung der Städte zueinander bestehen. Dies nennt man eine topografische Beziehung. Ein Beispiel für eine solche Beziehung wäre die Entfernung der beiden Stadtzentren. Durch bestimmte fachliche Eigenschaften der Geoobjekte wird auch noch deren Thematik beschrieben. Das heißt worauf die Daten Schlüsse zulassen und in welchen Gebieten sie angewendet werden können. Der zeitliche Bezug geht auf die Veränderungen der Objekte zurück, die sich in einer gewissen Zeit abspielen. Ein Geoobjekt kann sich in seinen drei Hauptpunkten (der Geometrie, Topologie und Thematik) über einen gewissen Zeitraum verändern [24].

Die Visualisierung von ein- und multidimensionalen Daten kann logischerweise in einer oder mehreren grafischen Dimensionen erfolgen. So können Daten beispielsweise eindimensional dargestellt werden, wie dies bei einer Dateiliste auf der Konso-

le eines Computers (im Beispiel Windows 2000 Remote Konsole zu einem Linux Server) realisiert ist.

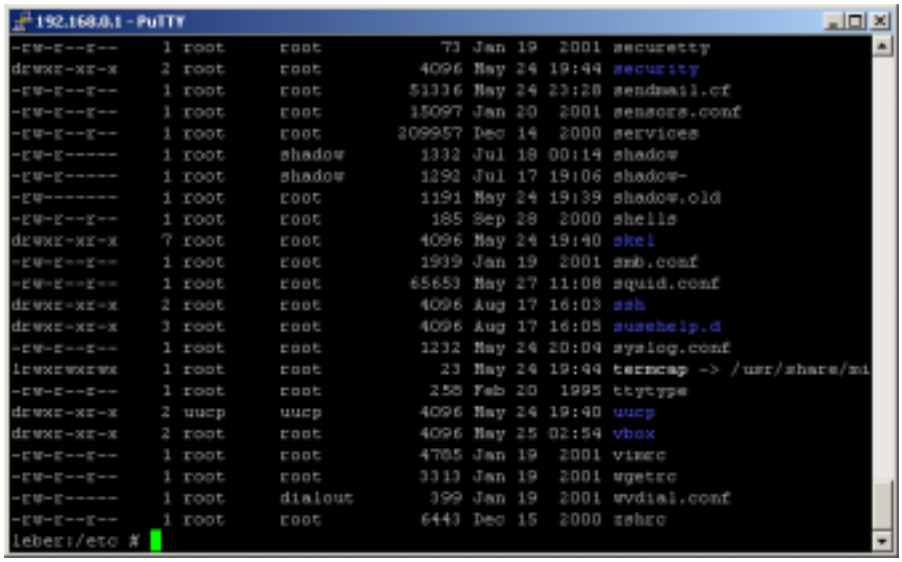

Abb. 3.1: 1D Dateiliste auf Konsole (Screenshot)

Hier werden Informationen über die Dateien eines Verzeichnisses durch eine einfache Liste, also eindimensional, visualisiert. Jede Zeile beschreibt eine Datei des Verzeichnisses. Eine Datei hat die Eigenschaften Lese- und Schreibrechte (für Besitzer, Gruppe und alle anderen), Besitzer der Datei, letztes Änderungsdatum und Dateiname. Durch Farben sind hier noch besondere Eigenschaften von einigen Dateien hervorgehoben.

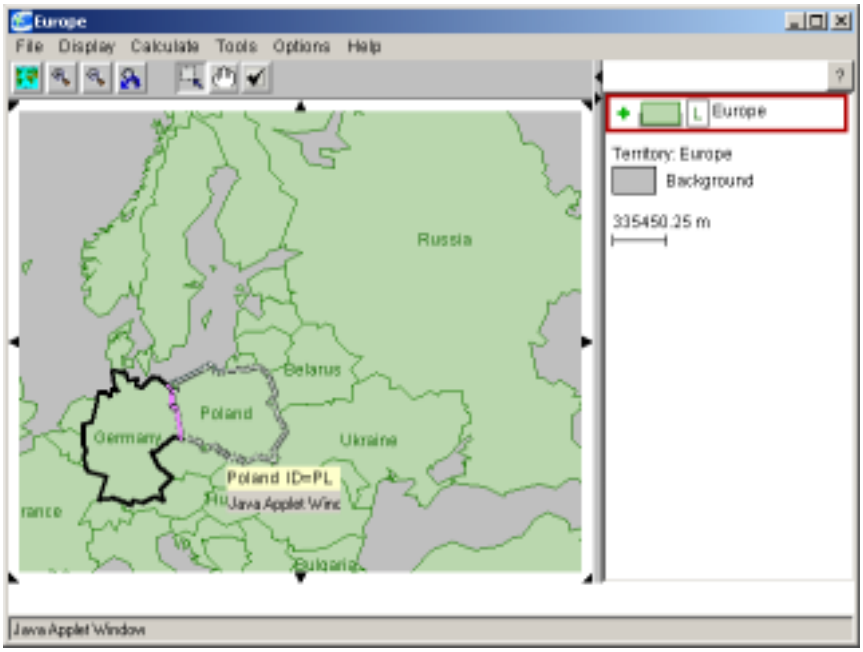

Abb. 3.2: 2D CommonGIS Java Applet (Screenshot)

Ein typisches Beispiel für eine zweidimensionale Visualisierung sind Landkarten. CommonGIS [17] ist ein System, um kartographische Geodaten darzustellen und Berechnungen anzustellen. Die verschiedenen, dargestellten Länder haben hier wieder sehr viele Eigenschaften, die mit Werten belegt sind. Außer den geografischen Maßen und dem Umfang, sind Entfernungen, Einwohner, Städte, Straßen, Flüsse, Klima und vieles mehr weitere Informationen über die Länder. Visualisiert ist hier nur ein Teil der Informationen, da es zu komplex wäre zu viele Daten der Realität abzubilden und da jedes System eine bestimmte Anwendungsdomäne hat.

In heutigen Geoinformationssystemen kommen schon häufig dreidimensionale Visualisierungen vor. Praktisch sind diese für eine gute Übersicht oder wenn das reale Objekt relativ exakt als Modell abgebildet werden soll, um damit zu arbeiten. Dies ist bspw. in der Medizin der Fall. Medizinstudenten können mit Hilfe des Visible Human Explorers den menschlichen Körper studieren. Andere Beispiele kommen aus der Chemie. Moleküle können in einer dreidimensionalen Umgebung visualisiert und ihr Verhalten simuliert werden. MolSurf[I4] in Abb. 3.3 ist ein Projekt der Universität Erlangen.

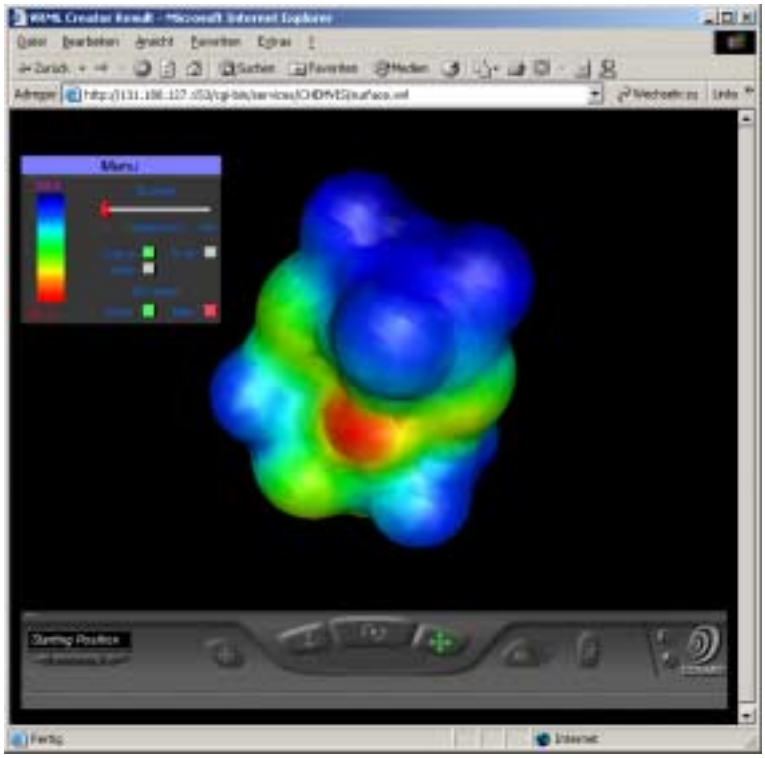

Abb. 3.3: 3D MolSurf, VRML Abbildung von Molekülen, Oberfläche (Screenshot)

Mittels des Cursors kann das Molekül in alle Richtungen gedreht werden. Der rote Slider mit der Beschriftung Surface, oben im Menu, lässt bei Betätigung zunehmend

die Oberfläche transparenter erscheinen, bis nur noch die Struktur (Abb.3.4) zu sehen ist. Die Struktur des Moleküls wird in der bekannten Schreibweise der Chemie mit Hilfe eines Tools zusammengebaut. Ein Programm setzt dann die Struktur in VRML um, damit ein VRML-fähiger Browser das Molekül visualisieren kann. In diesem Beispiel bestehen die multidimensionalen Daten aus den Strukturdaten der Moleküle.

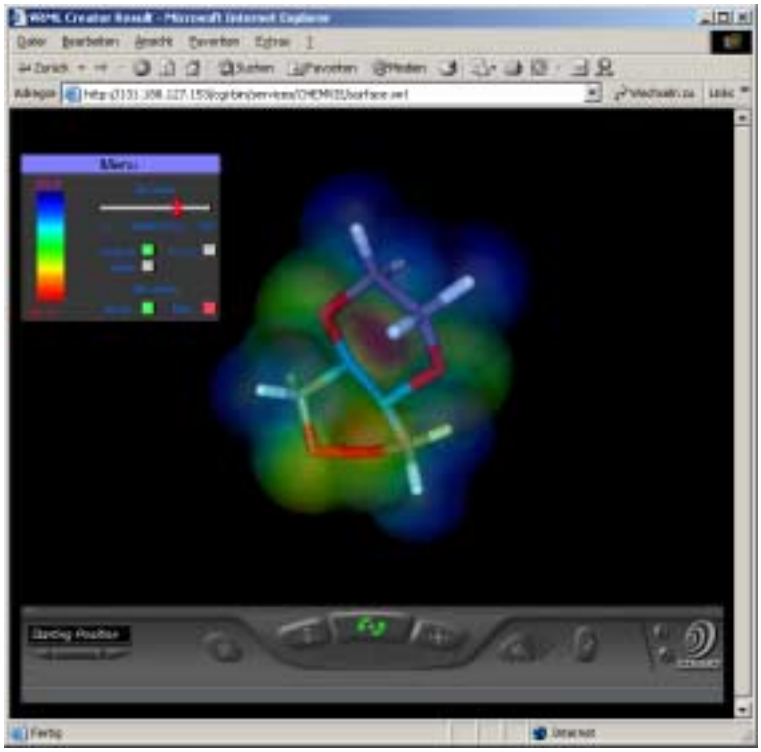

Abb. 3.4: 3D MolSurf, VRML Abbildung von Molekülen, Struktur (Screenshot)

Der INVISIP Scatterplot ist im momentanen Zustand ein Beispiel für die Visualisierung multidimensionaler Daten auf zwei wählbare Dimensionen. Eine dritte Dimension kommt hinzu, wenn eine Magic Lens Filter kreiert wird. Die momentan zu Testzwecken benutzten Daten, sind alte Daten aus gespeicherten Suchvorgängen des INSYDER Prototypen, also Metadaten von Internetdokumenten. Der in dieser Arbeit vorgestellte Scatterplot ist darauf ausgelegt, eine Vielzahl an Dokumenten darzustellen. Er kann dazu verwendet werden, um auf großen Datenbeständen Suchen auszuführen. Allgemein treten bei großen Datenmengen oder komplexen Berechnungen sehr schnell Performanceprobleme auf. Deshalb dürfen z.B. bei MolSurf nur eine eingeschränkte Anzahl Atome zum Bau des eigenen Moleküls verwendet werden. Obwohl heute die Computer technisch sehr viel weiterentwickelt sind als noch vor zehn Jahren, gibt es bei der Visualisierung von komplexeren Objekten Probleme mit der Geschwindigkeit und dem Speicherbedarf. Dies ist vor allem der Fall, wenn Da-

ten über das Internet bezogen werden. Bisherige kommerzielle Internet-Suchmaschinen zeigen die gefundenen Ergebnisse als einfache Trefferliste an. Bei INSYDER werden die Trefferlisten und gefundenen Seiten ausgewertet. Metadaten über die relevanten Dokumente werden gebildet. Mit diesen können die Visualisierungen die Dokumente darstellen und mit Links auf sie verweisen. Für eine Visualisierung von Rohdaten müssen diese erst gesammelt werden. Dann werden sie, je nach Visualisierung, in Beziehung zueinander gesetzt und in Skalierungen eingeteilt. Eine geeignete Struktur für die Daten muss gefunden werden, um diese dann abzuspeichern. Erst dann kann die Abbildung auf visuelle Strukturen erfolgen. Der INVISIP Scatterplot bekommt seine Daten beispielsweise im XML-Format. Die Metadaten beschreiben die zu visualisierenden Objekte in XML (Extended Markup Language).

3.2 Informationsvisualisierung (Information Visualization)

Man kann Visualisierungen als anpassbare Abbildungen betrachten. Die gewonnenen, zu visualisierenden Daten werden auf visuelle Formen abgebildet und schließlich vom Menschen wahrgenommen [5, 18].

Das Hauptproblem beim Design ist geeignete, ausdruckstarke und effektive Metaphern zu finden, die abstrakte Daten auf eine visuelle Form abbilden. Mit diesem Problem beschäftigt sich auch das Referenzmodell der Informationsvisualisierung [5].

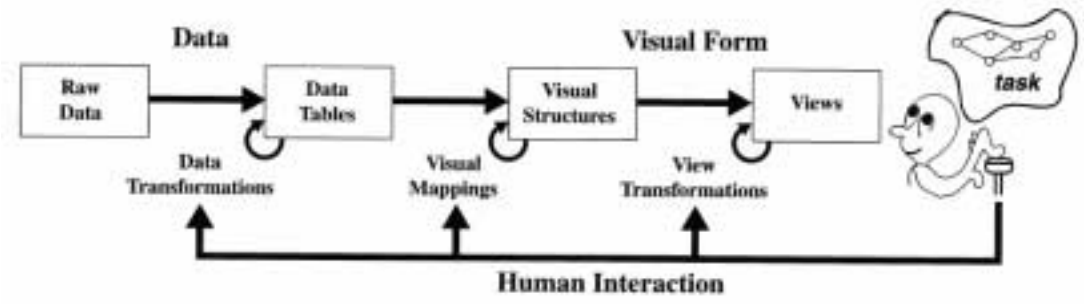

Abb. 3.5 : Das Referenzmodell der Informationsvisualisierung;
entnommen aus [5]

Von den Rohdaten zeigen Pfeile in Richtung Benutzer. Bei jedem Schritt werden Transformationen an den Daten vorgenommen. Die Pfeile, die vom Benutzer zu den Transformationen zeigen, sollen verdeutlichen, dass der Benutzer die Möglichkeit hat,

die Transformationen anzupassen. Während Rohdaten noch in jeder erdenklichen Form vorliegen können, werden die auf die Form von Data Tables gebrachten Daten in Beziehung zueinander gesetzt. Außerdem kommen durch die Beziehung meist Metadaten hinzu. Um die Daten visualisieren zu können ist es nötig, geeignete Skalierungen zu finden. Hierzu werden die aus der Statistik bekannten Skalierungen Nominal-, Ordinal- und quantitative Skalierung verwendet. Es gibt auch noch weitere Skalierungen, wie das Zusammenfassen von Objekten in bestimmte Klassen. Diese Skalierung wird Intervallskalierung genannt. Eine Klasse umfasst dann einen bestimmten Wertebereich und die Objekte können je nachdem, in welchen Bereich sie fallen, zugeordnet werden. Die Data Tables sind sehr gut auf Datenbanktabellen übertragbar. Zur Abbildung dieser Daten auf visuelle Strukturen benutzen wir den Raum. Wichtig bei dieser Transformation ist, dass die Logik der Daten bewahrt wird. Es muss auch darauf geachtet werden, dass der Mensch die dargestellten Informationen möglichst gut aufnehmen kann. Benutzeraufgaben und Arbeitsumgebung machen oft unterschiedliche Darstellungsarten der gleichen Daten notwendig. Die letztendliche Visualisierung entsteht erst bei der letzten Transformation. Die statischen visuellen Strukturen werden, durch grafische Parameter und mögliche Veränderungen durch den Benutzer, zu interaktiven Visualisierungen. Als View Transformations kommen hier die Location Probes (z.B. die Magic Lens), Viewpoint Controls (z.B. Zooming) und Verzerrungen zum Einsatz. Sie machen es durch bestimmte Konzepte möglich, mehr Informationen besser und benutzerfreundlicher darzustellen. Der INVISP Scatterplot benutzt einige dieser Konzepte. Dem Anwender wird so zusätzlich die Möglichkeit zur Änderung der Visualisierung gegeben.

3.3 Forschungsansätze zum Scatterplot

Grundlage für den Scatterplot ist das mathematische Koordinatensystem. Dieses kommt ursprünglich aus der Geometrie. Die Geometrie wird benutzt, um den Raum darzustellen bzw. den Raum in einem Modell zu simulieren. Wir nutzen hier die Geometrie, um mehrdimensionale Daten darzustellen.

Der INVISIP Scatterplot basiert visuell, nicht aber von der Implementierung her, auf dem Scatterplot des INSYDER Projektes. Die Entwickler des INSYDER Scatterplots haben sich hauptsächlich von zwei Projekten inspirieren lassen. Das erste Projekt heißt Envision [14] und wurde von Virginia Tech entwickelt. Es handelt sich um ein

visuelles System zur Informationssuche. Envision ist ein Prototyp einer digitalen Bücherei für Computerliteratur. Das System wurde nach den Prinzipien des User-Centered Designs entworfen. Benutzer können die Computerliteratur durchsuchen und es stehen verschiedene Wege zur Ansicht zur Verfügung. Envision beinhaltet ein flexibles System zur Analyse der Dokumentenstrukturen. Titel, Autor, Erscheinungsjahr und andere Daten werden den Dokumenten entnommen und können in beliebige Beschreibungsschemata übersetzt werden. In Abbildung 3.6 sieht man auf der linken Seite Eingabefenster, mit deren Hilfe man die Suchanfrage bestimmen kann. Autoren, Titelwörter und Inhaltswörter können angegeben werden.

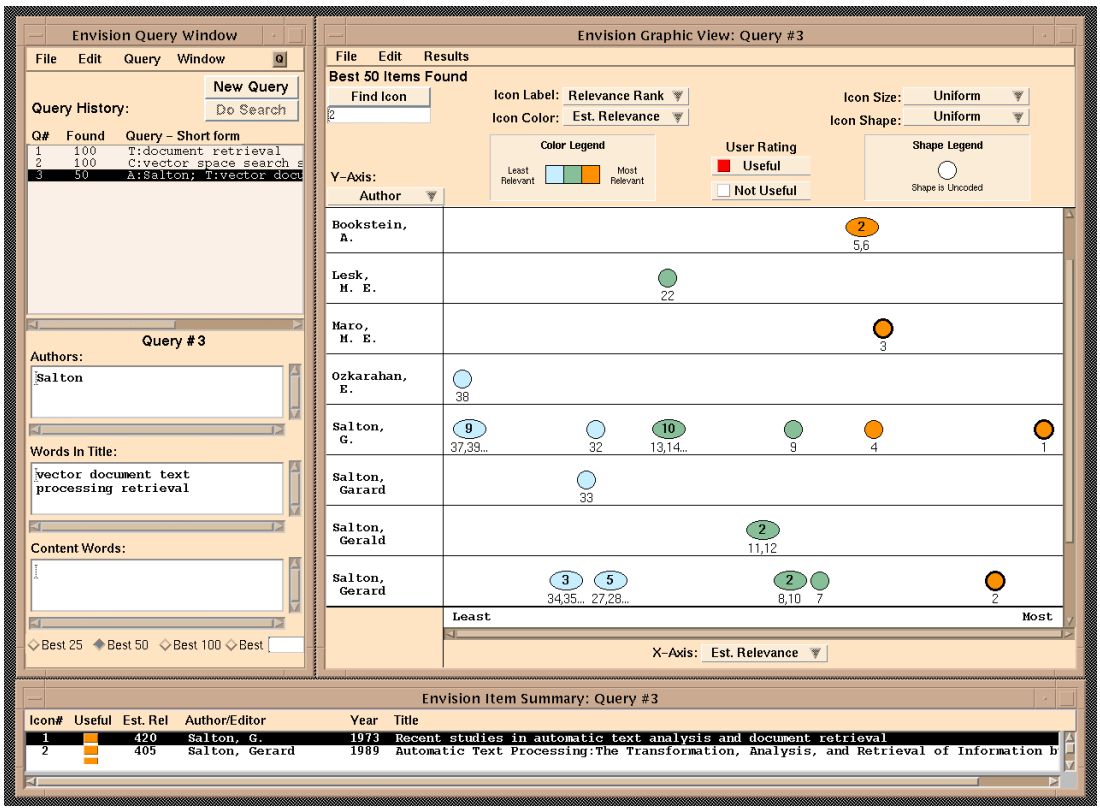

Abb. 3.6: Envision von Virginia Tech entnommen aus [I5]

Die Ergebnisse werden dann in einer Art Koordinatensystem auf der rechten Seite angezeigt. Die x-Achse ist mit dem Relevanzwert des Buches belegt und die y-Achse mit den Autoren. Wie beim INSYDER Scatterplot können die Achsenbelegungen geändert werden. Zum Beispiel könnte auf die x-Achse das Publikationsjahr gelegt werden, um neue von alten Büchern zu unterscheiden. Gesamtrelevanz der Dokumente ist in drei Stufen unterteilt, die immer sichtbar sind (siehe Color Legend). So kann ein Benutzer auch ohne eine der Achsen mit der Relevanz zu belegen, die Relevanz der Bücher einschätzen.

Das zweite Projekt das Inspirationen zum INSYDER Scatterplot lieferte, war der FilmFinder [15], der von der Universität Maryland entwickelt wurde. Dieses Tool ist zur Suche in einer großen Filmdatenbank geeignet. Die verschiedenen Filmgenres sind verschiedenen Farben zugeordnet (s. Abb. 3.7). So kann der Benutzer immer die für ihn interessanten Filmkategorien überblicken. In Abbildung 3.7 ist die y-Achse mit dem Popularitätswert belegt und auf der x-Achse sind die Erscheinungsjahre der Filme abgetragen. Die Achsenbelegung kann im FilmFinder nicht verändert werden, erst Spotfire [19], eine nachfolgende Version, behebt dieses Manko und lässt den Benutzer auch die Achsenbelegungen ändern.

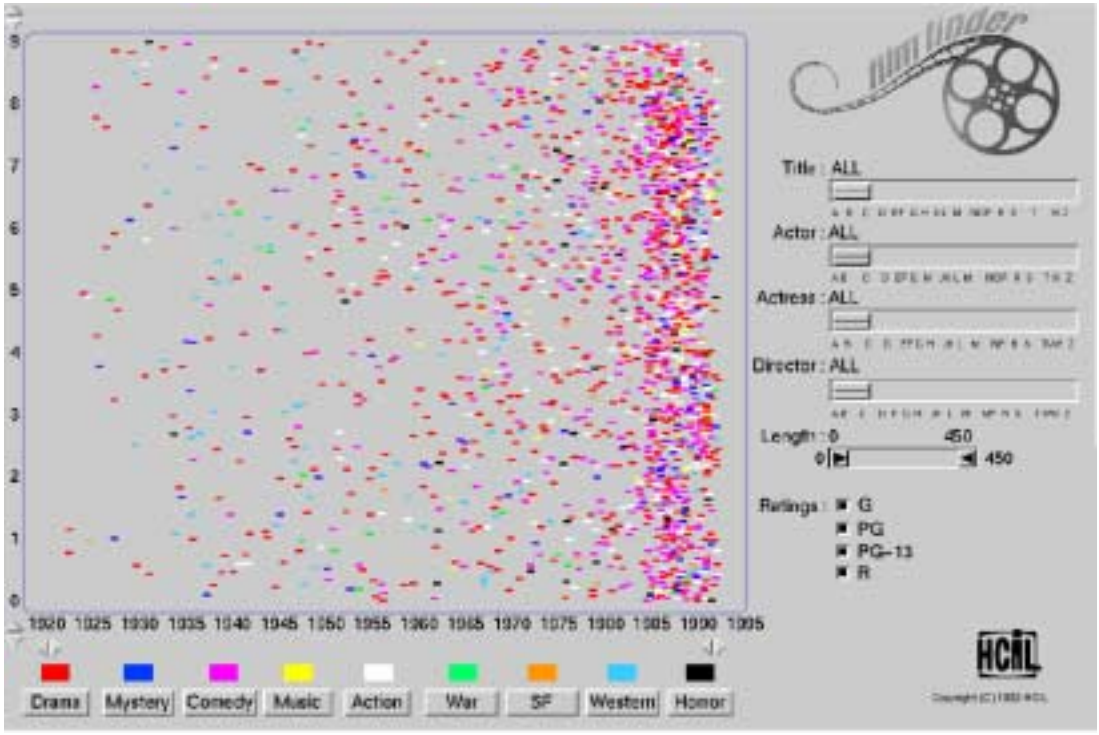

Abb. 3.7: FilmFinder aus [15]

Eine große Rolle spielen in dieser Visualisierung die Alphaslider auf der rechten Seite. Mit diesen können die angezeigten Filme, um Filme mit ungewünschten Eigenschaften, reduziert werden.

3.4 Forschungsansätze zur Magic Lens

Eine der Erweiterungen, die in den Scatterplot des INVISIP Projektes eingebaut wurden, ist die Magic Lens. Die Magic Lenses [20, 21, 22, 23] wurden 1993 in den USA, im Xerox PARC in Palo Alto, von Eric A. Bier, Ken Fishkin und Maureen C. Stone entwickelt. Die Xerox Corporation ließ ihr Produkt als Trade Mark eintragen und ein Logo (s. Abb. 3.8) wurde entwickelt. Das Logo zeigt schon die Grundfunkti-

onalität einer Magic Lens. Über einen Bereich, hier ein Teil des Wortes ‚Magic', wird eine Linse gelegt, die die Sichtweise des darunter liegenden Bereichs auf eine bestimmte Art und Weise verändert.

Abb. 3.8: Logo von Magic Lenses ™

Auf der linken Seite des Logos wird die Schrift durch die Linse transparenter gemacht und ein Stück vergrößert. Die rechte Linse vergrößert die Schrift nicht, sondern zeigt nur noch die Randlinien der Buchstaben an. Wie man hier sieht, können die Linsen in Form und Veränderungen variieren. Eine weitere, komplexere Eigenschaft der Linsen, ist deren Kombinierbarkeit.
Eine Magic Lens ist also eine Linse, vergleichbar mit einer Lupe, die über einen Bereich gelegt werden kann. Sie verändert dann die Sicht des Bereich aufgrund der ihr zugewiesenen Eigenschaften. Eric A. Bier, Ken Fishkin und Maureen C. Stone beschreiben Magic Lens Filters so:

„Magic Lens filters are a user interface tool that combine an arbitrarily-shaped region with an operator that changes the view of objects viewed through the region. The operator can be quite general, accessing the underlying data structures of the application and reformatting the information to generate a modified view." [20]

Der im INVISIP Scatterplot implementierte Magic Lens Filter wurde auf der Basis der Forschungsarbeiten zu Magic Lenses [20, 21, 22, 23] entwickelt. Die Vorteile der Linsen liegen auf der Hand. Wenn über eine vorhandene Ansicht eine Linse gelegt wird, kann diese eine modifizierte Ansicht im Kontext anzeigen. Für den Scatterplot war es außerdem wichtig, Ordnung und Übersicht zu behalten. So kann mit einem Magic Lens Filter bei zu vielen Informationen, die ungewünschte Information ausgeblendet werden, ohne die Achsenbelegung zu ändern. Umgekehrt kann es auch sein, dass bei einer Anwendung durch den Einsatz von Magic Lenses weitere Informationen angezeigt werden sollen. Bei einer elektronischen Straßenkarte können z.B. in der Originalansicht nur die Hauptstraßen angezeigt werden. Wird eine Magic Lens

über die Karte gelegt, werden auch die kleinen Straßen und evtl. noch mehr angezeigt. Abbildung 3.9 zeigt ein solches Beispiel.

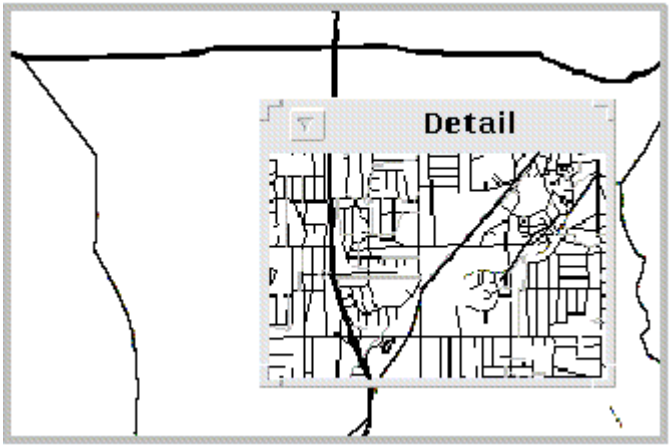

Abb. 3.9: Magic Lens zur Detailansicht im Kontext einer Anwendung, entnommen aus [22]

Weitere Vorteile sind eine einfache Erzeugung visueller Makros und die Möglichkeit, die Funktionalität der Linsen auf unterschiedliche Anwendungen und Informationen zu übertragen.

Ein weiteres sehr schönes Beispiel zur Implementation einer Magic Lens ist in Abbildung 3.10 zu sehen. Das Beispiel wurde von der Xerox Corporation 1996 in Form eines Java Applets implementiert.

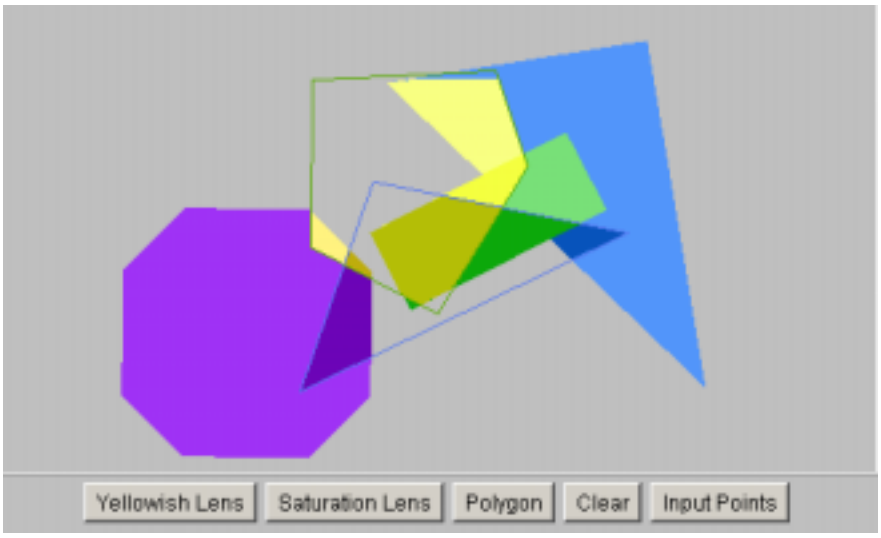

Abb. 3.10: Magic Lens [I6] (Screenshot)

Das Fünfeck mit dem leicht grünen Rand ist eine Yellowish Lens. Sie wandelt die Farbe aller Formen, die in ihrem Bereich liegen in gelb um. Das Dreieck mit dem blauen Rand ist eine Saturation Lens. Sie verdunkelt, bzw. sättigt die Farben der Formen, die in ihrem Bereich liegen. In der Mitte sieht man sehr schön, was bei einer

Kombination der beiden Linsen geschieht. Zuerst wurde die fünfeckige Yellowish Lens über einen Teil des grünen Rechtecks gelegt. Die überdeckten Bereiche erscheinen gelb. Als nächstes wurde noch die dreieckige Saturation Lens darüber gelegt. Die überdeckten zuvor gelb gefärbten Bereiche erscheinen jetzt in dunklem gelb.

Magic Lenses eigenen sich gut zur Erzeugung von Dynamic Queries. Durch sie kann der Benutzer direkt im Kontext der Anwendung dynamische Anfragen generieren. Die direkte Manipulation ist intuitiv und kann dazu benutzt werden, die Anfragen dynamisch veränderbar zu machen. Durch diese Techniken können die angezeigten Informationen schnell auf die relevanten reduziert werden. Durch die Kombination mehrerer Linsen können Anfragen einfach kombiniert werden. Die Kombination zweier oder mehrerer Linsen funktioniert nach einem einfachen, recht intuitiven Prinzip. Die unterste Linse bekommt den Originalinput an Informationen und gibt an die darüber liegende Linse die, je nach Eigenschaften der Linse, geänderten Informationen weiter.

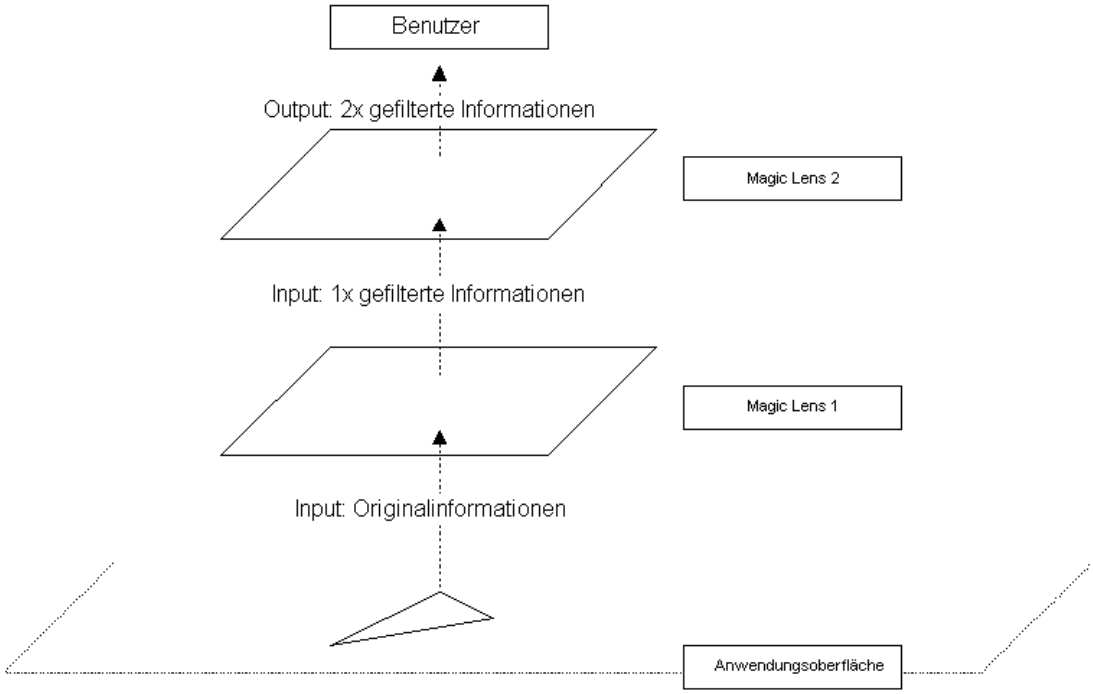

Abb. 3.11: Kombination von Magic Lens Filters

Auf der Basis der oben vorgestellten Projekte wurde der Magic Lens Filter des INVISIP Scatterplots entwickelt und anschließend implementiert. Es können mehrere Magic Lens Filter im Scatterplot erzeugt und kombiniert werden.

4 Der Scatterplot des INVISIP Projekts

4.1 Start des Projekts

Das Projekt begann mit einer konkreten Aufgabenstellung. Der INSYDER-Prototyp sollte einem Redesign inklusive einer eventuellen Neuprogrammierung unterzogen werden. Die neuen, verbesserten Komponenten sollten im INVISIP Projekt eingesetzt werden. Das Programmierteam bestand anfangs aus drei Studenten von denen einer nach kurzer Zeit das Team verließ. Regelmäßige Treffen mit den Betreuern des Projektes wurden abgehalten. Alle Überlegungen, Gedanken und Ergebnisse wurden dort nochmals überdacht und präsentiert. Nachdem das INSYDER Projekt installiert, untersucht und benutzt worden war, wurde ein erstes Brainstorming abgehalten, um die ersten Verbesserungsvorschläge zu erfassen.

4.1.1 Brainstorming

Die Ergebnisse sind hier in drei Punkte unterteilt, die GUI (Graphical User Interface), die Datendarstellung und die Benutzerdialoge. Die Menueinträge in der Menuleiste der Benutzeroberfläche wurden teilweise als unlogisch kritisiert. Sie sollten für die Anwender selbsterklärender sein. Auch ist der Toolbar des GUIs keine Funktionalität zugeordnet. Das Layout der Tabbed Panes sollte verbessert und die Wortwahl eindeutiger und passender ausgesucht werden. Der zweite Kritikpunkt am INSYDER ist die Art der Datendarstellung. Die bestehenden Sichten im Tabbed Pane, dem Hauptfenster der Anwendung, sollten durch eine Tabelle (Supertable) ersetzt werden. In Einbezug der Idee aus [9] ist der Einbau eines Granularitäts-Sliders über der Tabelle vorgesehen. Für die Supertable und den Scatterplot, die beiden Hauptvisualisierungen, ist ein permanentes Splitpane vorgesehen. Durch die Einführung neuer Darstellungskomponenten sollte die Darstellung der Dokumente in der Tabelle verbessert werden. Eine detaillierte Textansicht und die Darstellung aller Granularitätsstufen sollten in der Supertable verwirklicht werden. Die neue, erweiterte Supertable wurde in ‚Granularity Table' umbenannt. Schließlich wurde die Neuimplementation des Scatterplots einer Überarbeitung bevorzugt. Diese Entscheidung fiel aufgrund des unübersichtlichen Quelltextes, der ohne Konzepte straight-forward programmiert worden war. Der Scatterplot hätte nur mit übermäßigem Zeitaufwand aus dem INSYDER Projekt herausgelöst werden können. Außerdem enthielt der Quellcode keine Kommentare und es war schwer nachzuvollziehen, welche Vorgehensweise die

Programmierer verfolgt hatten. Außer der Neuimplementation des Scatterplot, sollte dieser noch mit der Supertable bzw. Granularity Table interaktiv und über Komponentenkommunikation verknüpft werden. Der letzte Kritikpunkt am INSYDER sind die Benutzerdialoge. Die Führung des Benutzers bei der Eingabe einer Suche war nicht intuitiv und schwer nachzuvollziehen. Weitere Verbesserungsvorschläge, die beim Brainstorming entstanden, waren die eventuelle Einführung eines Wizards und mehr Feedback für den Benutzer.

4.1.2 Programmspezifikation

Der nächste Schritt im Projektablauf war die Programmspezifikation. Hier wird festgelegt, welche Ziele gesetzt werden, wie diese erreicht werden sollen und welche äußeren Umstände Einfluss auf das Projekt haben. Im folgenden werden die erarbeiteten Punkte der Programmspezifikation aufgelistet.

Zielbestimmungen und Aufgabenstellung

Die Musskriterien und damit auch die Aufgabenstellung für das Projekt sind ein Redesign der Benutzeroberfläche des bestehenden INSYDER Forschungsprototypen. Dabei steht die Implementation neuer, sowie die Übernahme und Verbesserung alter Visalisierungsmethoden im Vordergrund. Für die Arbeit ‚INVISP - Implementation eines Scatterplots zur Visualisierung von geo-räumlichen Metadaten' war die Aufgabe den Scatterplot des INSYDER Forschungsprototypen neu zu programmieren, in den INVISIP Framework einzubauen und mit der Granularity Table zu verknüpfen.

Abgrenzungskriterien

Im Vordergrund stehen Visualisierungsmethoden und die Benutzbarkeit des Programms. Das Endprodukt soll ein funktionsfähiger Scatterplot zur Darstellung von Metadaten sein. Nicht so wichtig ist die Programmperformance, da diese auch von anderen Komponenten abhängig ist und nachträglich noch verbessert werden kann.

Produkteinsatz

Der Anwendungsbereich des Scatterplots ist vorerst ein lokaler Rechner. Der Scatterplot im Framework des INVISIP Projektes ist auf einem lokalen Computer lauffähig. Als Testpersonen kommen zur Zeit Studenten und Mitarbeiter des Fachbereichs zum Einsatz. Das Produkt wird in einer durchschnittlichen Büroumgebung eingesetzt.

Produktumgebung und Entwicklungsumgebung

Die entwickelte Software wird als Applikation in der Programmiersprache JAVA entwickelt und ist wegen der interpretierenden Virtual Machine (VM) plattformunabhängig. Zur Entwicklung werden visuelle Entwicklungsumgebungen benutzt. Dazu gehören der JBuilder 6 von Borland, BlueJ und VisualAge von IBM. Die geschätzte Hardwareanforderung für die Entwicklung und das Produkt ist ein IBM kompatibler Personalcomputer mit zur Zeit durchschnittlicher Ausstattung. Ungefähr geschätzte Anforderung: Pentium III 500 gleichwertiger Prozessor, 256 MB Arbeitsspeicher, Maus, Tastatur und eine Grafikkarte mit mindestens 16 MB Grafikspeicher. Die zur Entwicklung verwendeten Betriebssysteme sind SuSe Linux 7.3 und die MS Windows Betriebssysteme 98 und 2000. Als Hardware stehen Personal Computer mit Intel Pentium III und AMD Athlon Prozessoren mit einem Arbeitsspeicher von mindestens 512 MB und mit 64 MB 4xAGP Grafikkarten zur Verfügung. Die Rechner verfügen außerdem über einen DSL Zugang. Als Versionsverwaltungssystems ist CVS eingerichtet. Die Programmierer können lokal ihr Projekt vom Server herunterladen, sowie update am Projekt vornehmen. Dies geschieht alles über das Internet.

Produktfunktionen

Das GUI des Programmes besteht aus dem Application Window. Dies ist das Hauptanwendungsfenster. In Java-Swing programmierte Komponenten, wie Jpanels, MenuBar und MenuItems, die ToolBar, TabbedPanes und andere Komponenten werden zur Oberflächenprogrammierung eingesetzt.

Visualisierungen

In der Endversion des Projektes sollen Metadaten von Internetdokumenten, die aus alten Testdaten des INSYDER Prototypen stammen, eingelesen werden und mit dem Scatterplot und der Granularity Table visualisiert werden können.

Beschäftigungsdaten

Der Projektbeginn war am 1. November 2001. Die Implementation des Scatterplot wurde im Dezember begonnen und war Ende Juni abgeschlossen.

Benutzungsschnittstelle

Als Benutzungsschnittstelle für den Scatterplot und das gesamte INVISIP Projekt ist eine WIMP Benutzeroberfläche vorgesehen. Es wurden insbesondere Menusteuerungen, Fenster, Dialoge und Frames zur Entwicklung der Oberfläche benutzt. Die Bedienung des Produktes erfolgt durch Maus und Tastatur.

Testszenarien

Getestet und evaluiert wurde der Stand des Projektes von drei Expertenbenutzern. Es sollen auch noch Studenten von nicht naturwissenschaftlichen Fakultäten eingeladen werden, um das Produkt zu testen und Verbesserungsmöglichkeiten zu finden.

4.1.3 Konkrete Aufgabenstellung

Die konkrete Aufgabenstellung für die Neuimplementation des Scatterplots umfasst die folgenden Punkte und wurde vollständig erfüllt.

Es soll eine grafische Darstellung eines Koordinatensystems, angelehnt an den INSYDER Scatterplot, entwickelt werden. Die Achsendefinitionen müssen variabel sein. Weiterhin werden die Datenwerte auf den Achsen abgebildet. Der Datenraum und die Achsen werden entsprechend der darzustellenden Werte eingeteilt.

Beide Achsen müssen in der Lage sein, Zahlenwerte, Buchstabenwerte und bool'sche Werte anzuzeigen. Die Start- und Endwerte der auf den Achsen abgetragenen Daten passen sich automatisch an die vom Datenmodell übergebenen Datensätze an. Die Intervalle werden also berechnet und je nach eingelesenen Werten, werden die Minima und Maxima der Achsen bestimmt. Auch die Einteilung der Major- und Minorticks auf den Achsen werden, entsprechend der anzuzeigenden Daten, passend berechnet.

Die vom Datenmodell gelieferten Daten selbst, sollen als grafische Objekte im Scatterplot dargestellt werden. Zum Test werden XML-Testdatensätze eingelesen. Alle Punkte sollen mit der linken Maustaste selektiert und deselektiert werden können. Der bool'sche Wert Selektion muss dann entsprechend im Datenmodell geändert werden. Optisch muss erkennbar sein, ob ein grafisch dargestelltes Datenobjekt selektiert ist oder nicht. Bei manchen Achseneinteilungen kann es außerdem geschehen, dass mehrere Datenpunkte auf eine Stelle fallen und dann nicht mehr als einzelne Punkte identifiziert werden können. Um dieses Problem zu lösen, muss eine sogenannter Multipoint erstellt werden, die alle enthaltenen Punkte grafisch auf eine andere Weise darstellt. Diese neue Art von Datenpunkt soll auch darstellen, wie viele Punkte enthalten sind und welche bzw. wie viele davon selektiert sind.

Ein weiteres Feature des Scatterplots soll eine Zoomfunktion sein. Diese soll genutzt werden können, um Bereiche des Scatterplots vergrößert darzustellen und die Daten besser zu untersuchen. Mit einem Auswahlrechteck soll eine Fläche mit der linken Maustaste aufgezogen werden können. Durch einen Zoom müssen sich dann die

Achsen des Scatterplots an die neuen Werte anpassen. Es muss möglich sein, mehrere Male zu Zoomen und danach den Weg der Zooms wieder zurück zu verfolgen. Eine Zoomhistory ist also notwendig.
Für den Scatterplot soll zusätzlich ein Magic Lens Filter implementiert werden. Dieser kann über einen Bereich des Scatterplots gelegt werden und filtert die darunter liegenden Daten nach bestimmten Kriterien. Es soll auch möglich sein, mehrere Filter zu kombinieren bzw. übereinander zu legen.
Der fertige Scatterplot soll in das INVISIP Projekt eingebunden werden und lauffähig sein. Die Kommunikation mit dem Datenmodell und den anderen Komponenten soll implementiert werden. Außerdem sollen der Scatterplot und die Granularity Table jeweils auf Benutzeraktionen reagieren, die in den jeweils anderen Visualisierungen ausgelöst werden.

4.2 Vorstellung des entwickelten Projektes

4.2.1 Der Scatterplot

Das vorläufige Endprodukt des im Rahmen des INVISIP Projektes entwickelten Scatterplots, soll hier kurz und anhand von Bildern erläutert werden. Außerdem werden die angewendeten Konzepte des Scatterplots und dem Magic Lens Filter erklärt. Die vorläufige Endversion des Scatterplots dieser Arbeit ist auf den Abbildungen 4.1 ff. zu sehen. In Abbildung 4.1 sieht man den Scatterplot mit 600 eingelesenen Testdatensätzen. Aus programmiertechnischen Gründen schneiden sich die beiden Achsen nicht im Ursprung. Dies wurde bei einem Projekttreffen jedoch auch nicht als Problem angesehen.

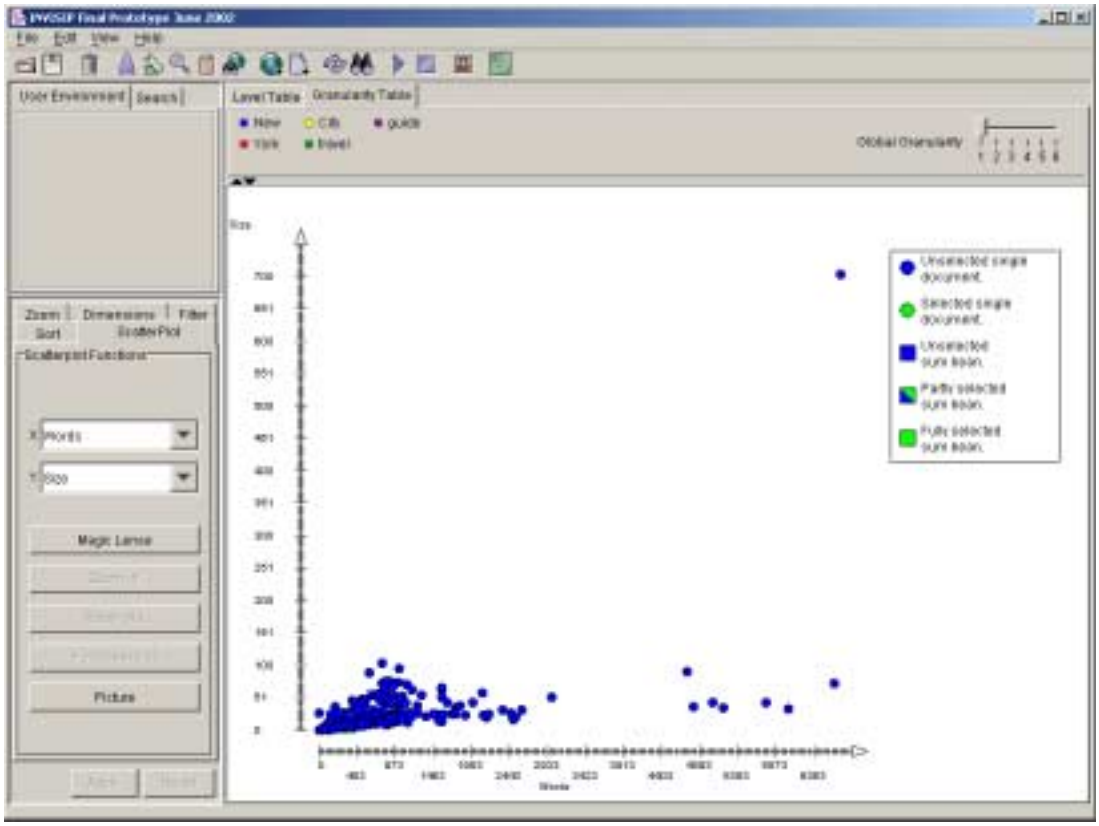

Abb. 4.1: Der Scatterplot mit 600 eingelesenen Testdatensätzen

Der grundsätzliche Aufbau besteht aus dem Hauptfenster mit einer Menubar. Über die darunter angebrachte Iconbar sollen sich in zukünftigen Entwicklungsstadien einige Funktionen steuern lassen. Von Bedeutung für den Scatterplot sind dann nur noch das Tabbed Pane mit dem Namen ‚ScatterPlot' (links unten) und das Hauptfenster. Die Achsenbelegung ist standardmäßig so wie in Abbildung 4.1 eingestellt. Auf der x-Achse wird die Anzahl der Wörter angegeben und auf der y-Achse die Größe der Dokumente in Kilobytes. Die Standardwerte können natürlich ohne weiteren großen Aufwand im Quelltext geändert werden. Dadurch, dass der Scatterplot zwei Dimensionen gleichzeitig anzeigen kann, sieht man schon auf einen Blick, dass bei den eingelesenen Testdatensätzen fast alle Dokumente eine geringe Anzahl an Kilobytes und weniger als 2500 Wörtern haben. In den ‚Scatterplot Functions' im Tabbed Pane kann die Achsenbelegung geändert werden, ein Magic Lens Filter hinzugefügt werden oder die Zoomfunktion bedient werden. Zusätzlich besteht die Möglichkeit ein Hintergrundbild in den Scatterplot zu laden. Zum Einfügen des Hintergrundbildes öffnet sich der standardmäßige Dateibrowser des jeweiligen Betriebssystems und es kann bequem eine Bilddatei ausgewählt werden. Diese wird dann im Hintergrund des Scatterplots angezeigt (Abb. 4.2). Falls die Größe des gewählten

Bildes nicht den Maßen des Hintergrundes entspricht, werden die Bildmaße so angepasst, dass das Bild passt. Gegebenenfalls wird das Bild, wie im Beispiel, etwas verzerrt.

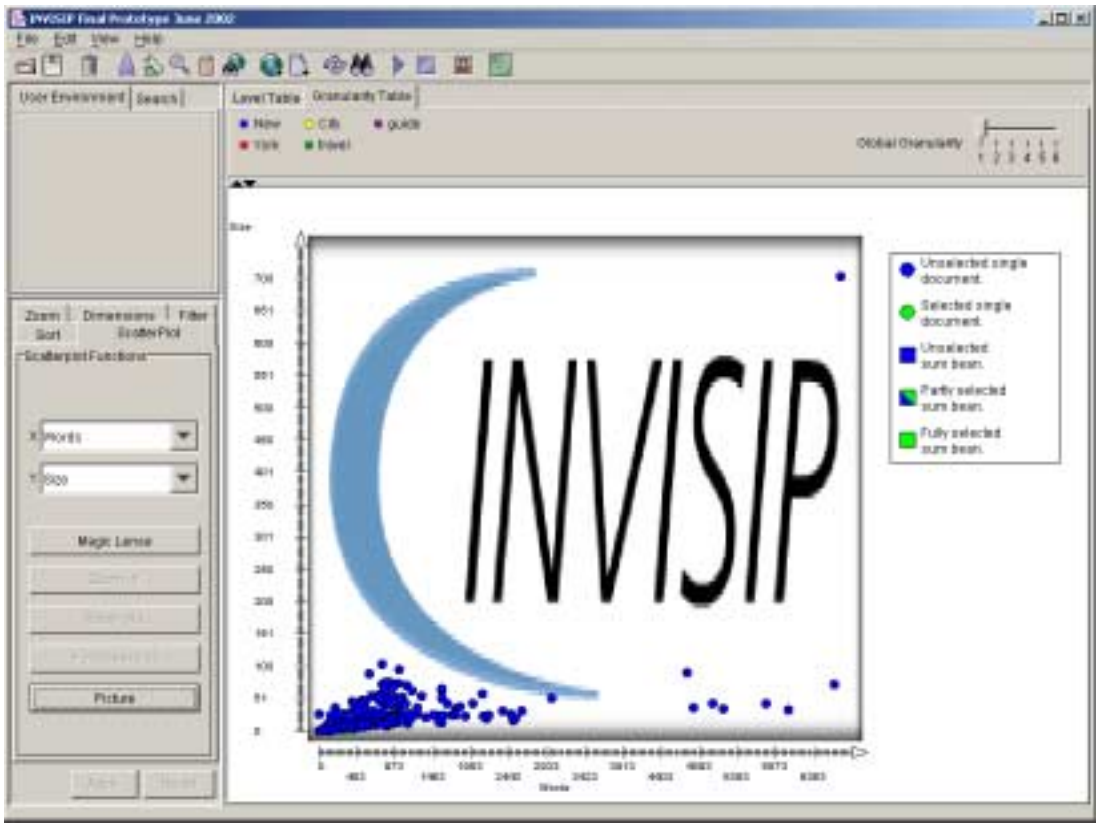

Abb. 4.2: Der Scatterplot mit dem INVISIP Logo als Hintergrundbild

Im Hauptfenster befindet sich auf der rechten Seite eine Legende mit den Bedeutungen der Dokumentensymbole. Wie auch die Standardachsenbelegungen können auch die Standardfarben ohne weiteres angepasst werden. Ein runder, blauer Punkt heißt, dass das Dokument nicht selektiert ist und keine weiteren Dokumente auf die gleiche Position fallen. Ein grüner, runder Punkt repräsentiert ein selektiertes, einzelnes Dokument. Die Quadrate wurden, wie im INSYDER Scatterplot, als Repräsentationsform für mehrere, auf einer Stelle liegende Dokumente verwendet. Wenn also mehrere Dokumente auf einer Koordinate liegen, werden aus mehreren runden Punkten ein Quadrat. Diese sogenannten Multipoints können entweder keine selektierten Punkte enthalten, dann werden sie ganz blau dargestellt. Wenn sie einen oder mehrere selektierte Punkte enthalten, dann werden die zur Hälfte blau und zur anderen Hälfte grün dargestellt. Wenn alle Dokumente, die zu einem Multipoint zusammengefasst sind, selektiert sind, dann ist die Darstellung im Scatterplot grün.

Bei geänderter Achsenbelegung sind die verschiedenen Variationen gut zu sehen. Leider kann es auf den Achsen zu Schwierigkeiten mit der Beschriftung kommen, da die Länge der verschiedenen Namen und die Anzahl der Beschriftungen dynamisch ist. So weiß man nicht, wie lang die Beschriftung werden kann und muss diese nach einer gewissen Länge abschneiden. Es kann also geschehen, dass nicht die ganze Beschriftung angezeigt wird. Mit einem Mouse-Over Effekt wird aber dann, trotz abgeschnittener Beschriftung, die ganze Beschreibung angezeigt (Abb. 4.3).

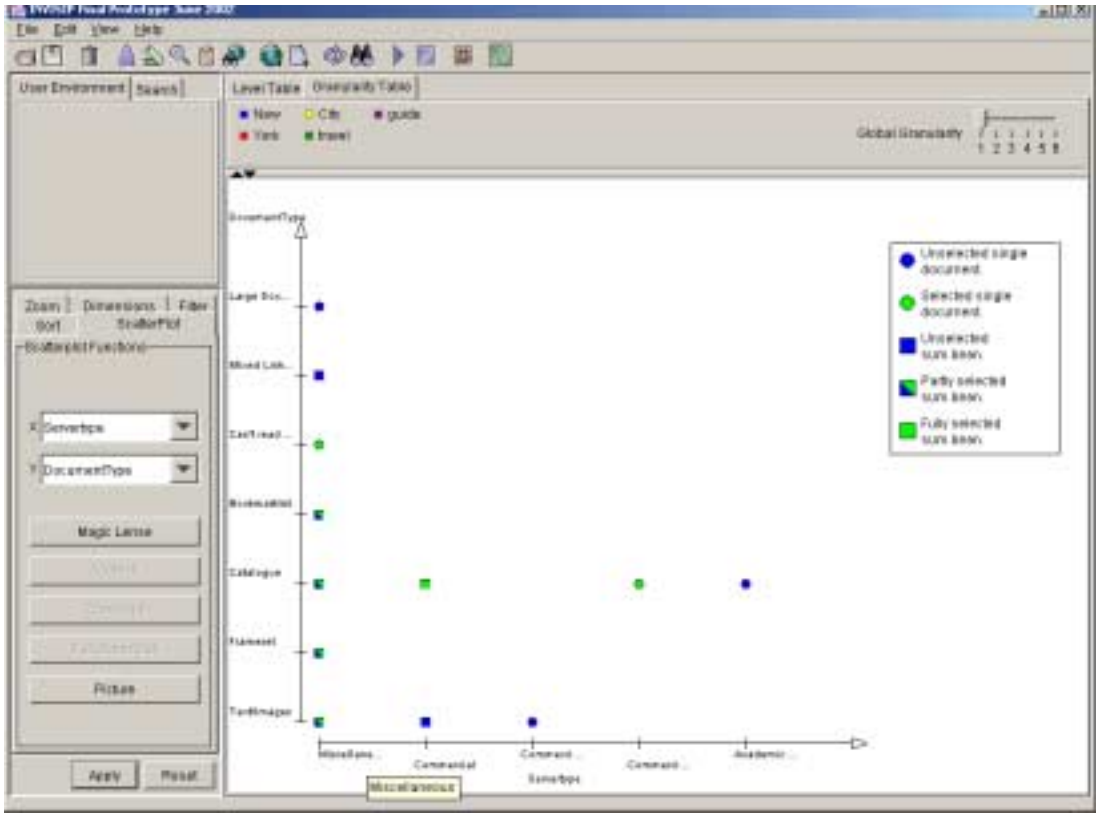

Abb. 4.3: Der Scatterplot mit dem INVISIP Logo als Hintergrundbild

Der Mauszeiger befindet sich über dem ersten Wert der x-Achse. Durch das kleine, gelbe Label wird dem Benutzer die ganze Beschriftung ‚Miscellaneous' angezeigt, obwohl die Beschriftung auf der Achse abgeschnitten wurde. Die y-Achse ist jetzt mit den ‚Document Type' Werten und die x-Achse mit den ‚Server Type' Werten belegt.

4.2.2 Magic Lens Filter, Zooming und Interaktion im Scatterplot

Durch einen Magic Lens Filter kann jetzt noch eine weitere Dimension hinzugeschaltet werden. Zur Erzeugung kann der Button ‚Magic Lens' im Tabbed Pane ‚ScatterPlot' geklickt werden. Es wird ein Konfigurationsdialog (Abb. 4.4) eingeblendet, mit dessen Hilfe ein neuer Magic Lens Filter erzeugt werden kann.

blendet, mit dessen Hilfe ein neuer Magic Lens Filter erzeugt werden kann. Zum gleichen Dialog gelangt man, wenn man über der Oberfläche des Scatterplots die rechte Maustaste klickt und den Menupunkt ‚Magic Lens' auswählt.

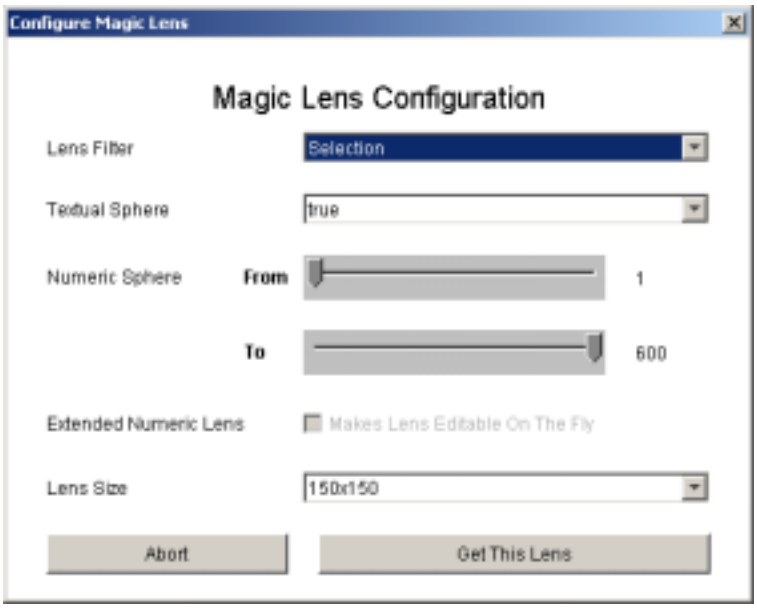

Abb. 4.4: Der Magic Lens Konfigurationsdialog

Die hier zu Testzwecken erzeugte Linse filtert die selektierten Dokumente aus, da der ‚Lens Filter' über die Combo Box auf ‚Selection' eingestellt wurde. Es stehen also die Werte ‚true' und ‚false' als anzeigbar zur Verfügung. Hier wurde ‚true' gewählt. Es werden jetzt also alle nicht selektierten Dokumente unter der Linse ausgeblendet. Die ‚Numeric Sphere' kann nicht bedient werden, da bool'sche Werte (wie die Selektion) hier auch unter die ‚Textual Sphere' fallen. Wenn ein numerischer Wert als Linsenfilter gewählt wird, kann der Filterbereich mit den beiden Slidern eingestellt werden. Unter dem Punkt ‚Extended Numeric Lens' kann eine numerische Linse auch nach Erzeugung noch geändert werden. Dazu muss die Checkbox aktiviert werden. Zum Schluss kann noch die Größe der Linse bestimmt werden, indem eine der drei angebotenen Größen in der Combo Box ausgewählt wird. Die Größen des Magic Lens Filters können ohne große Änderungen im Quelltext angepasst werden. Über den ‚Abort' Button wird das Dialogfenster geschlossen und die Erzeugung des Magic Lens Filters wird abgebrochen. Durch einen Klick auf ‚Get This Lens' wird eine entsprechend konfigurierte Linse im Scatterplot angezeigt und der Dialog geschlossen. Der Dialog ist nur zu Testzwecken gedacht und genügt keinesfalls einer benutzerfreundlichen Bedienung.

In Abbildung 4.5 ist ein Magic Lens Filter angezeigt. Die Linse kann durch gedrückt gehaltenen linken Mauszeiger im Scatterplot verschoben werden. Sie wurde genau über das linke untere Eck des in Abbildung 4.3 dargestellten Scatterplots geschoben.

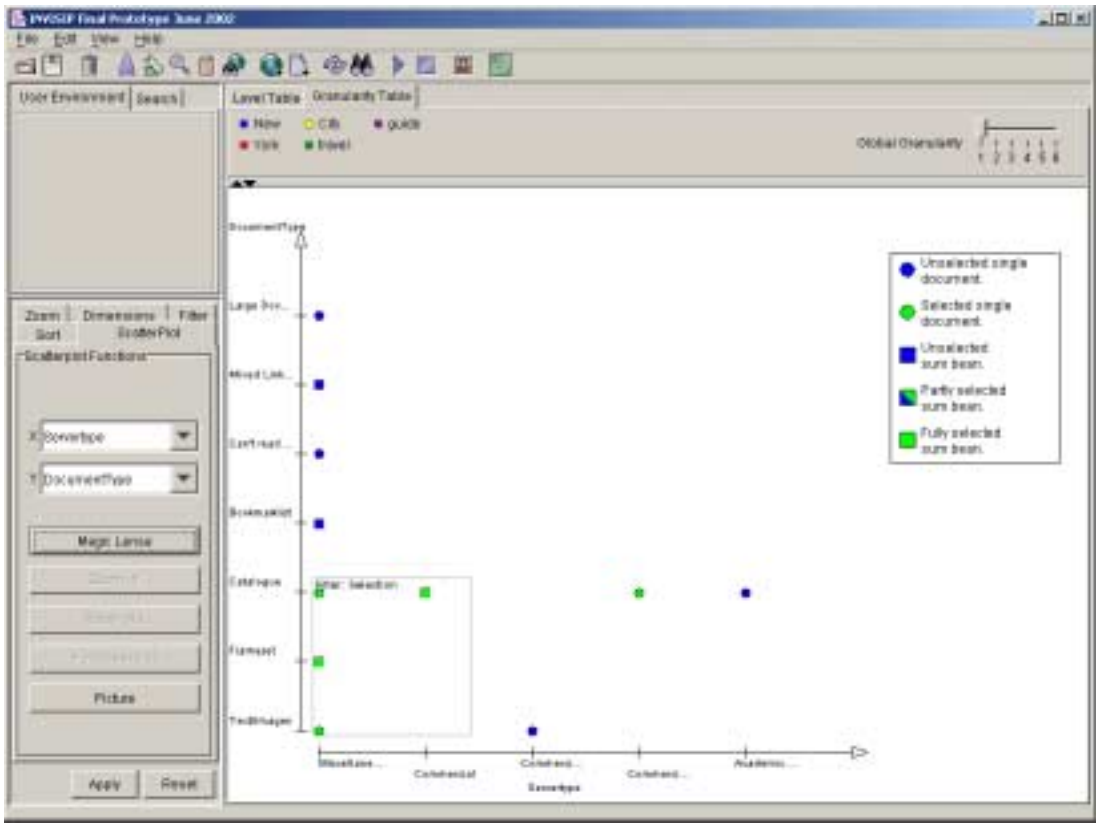

Abb. 4.5: Scatterplot mit Magic Lens Filter

Man sieht sehr schön die Funktion der Linse. Während in Abbildung 4.3 noch alle Dokumente angezeigt wurden, werden in 4.5 unter der Linse liegende Dokumente nur noch angezeigt, wenn sie selektiert sind. Aus den Multipoints werden je nach enthaltenen Dokumenten einzelne, selektierte Dokumente oder komplett selektierte Multipoints. Blaue, nicht selektierte Multipoints und nicht selektierte normale Dokumente fallen ganz weg. Mischformen können demnach nicht auftreten.

Der Magic Lens Filter kann aber auch als Übersichtswerkzeug dienen. Wenn man sich noch mal das Startbild in Abbildung 4.1 ansieht, bemerkt man die große Punktewolke links unten im Scatterplot. Eine geeignet konfigurierte Linse lässt hier, durch Hinzunahme einer weiteren Filterdimension, mehr Übersichtlichkeit zu. Dokumente mit ungewünschten Eigenschaften werden ausgeblendet. Ein Beispiel wäre eine Linse mit einem Filter für die Sprache der Dokumente. Wenn man die Abbildung 4.1 mit der Abbildung 4.6 vergleicht, sieht man, wie aus der Punktwolke sehr viele unrelevante Dokumente ausgefiltert werden. Der benutzte Magic Lens Filter

zeigt hier nur die Dokumente in französischer Sprache an. Alle anderen werden ausgeblendet.

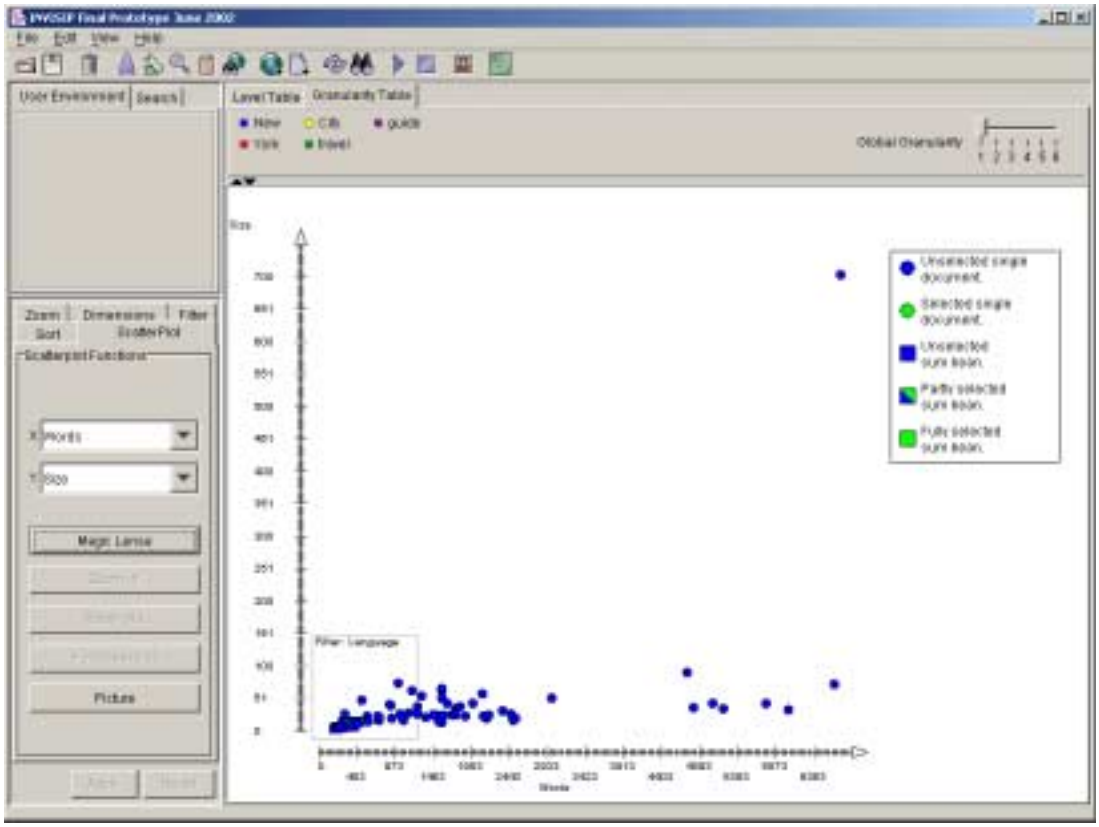

Abb. 4.6: Scatterplot mit Magic Lens Filter

Zusätzliche Linsen können den Filter noch verbessern. So werden dynamische Anfragen generiert, die grafisch durch Überschneiden der verschiedener Linsen abgeändert und kombiniert werden können. Nimmt man eine Magic Lens hinzu, erweitert man die Anfrage. Beim Löschen einer Magic Lens wird auch die Suchanfrage wieder allgemeiner. Dies entspricht dem Konzept der Dynamic Queries. In Bild 4.7 sieht man, wie die Anfrage, nur französische Dokumente einzublenden, mit einer zweiten Magic Lens erweitert wurde. Durch die Kombination werden jetzt unter beiden Linsen nur die selektierten Dokumente in französischer Sprache angezeigt. Es verbleiben nur noch zwei relevante Dokumente in der Anzeige.

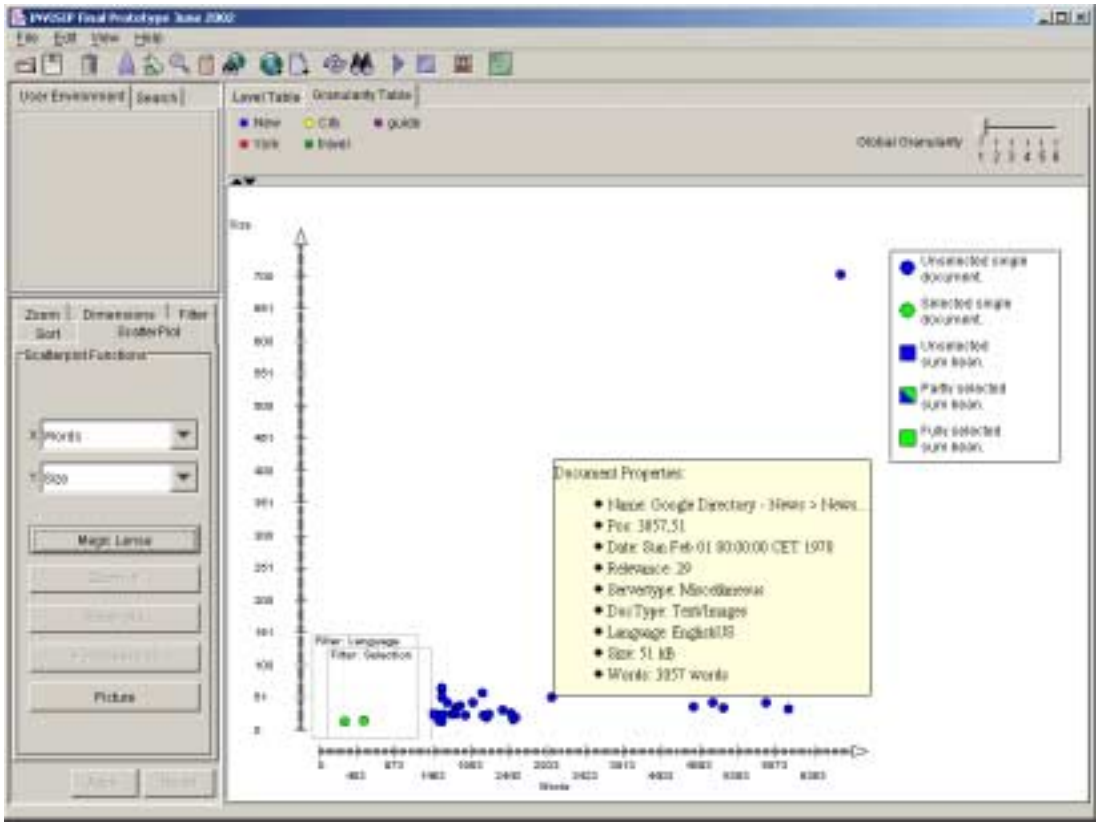

Abb. 4.7: Scatterplot mit zwei Magic Lens Filtern

Außer der Magic Lens werden im Scatterplot noch weitere Location Probes angewendet. Durch überfahren eines Punktes mit dem Mauszeiger wird ein Pop-Up Fenster mit genaueren Informationen angezeigt (Abb. 4.7). Außerdem kann mit einem Klick der rechten Maustaste ein Kontext-Pop-Up Menu angezeigt werden.

Die View Transformations werden auch noch um eine Zoomfunktion erweitert. Mit deren Hilfe ist es möglich, interessante Punktewolken näher zu betrachten. Diese Art der View Transformation wird Viewpoint Control genannt.

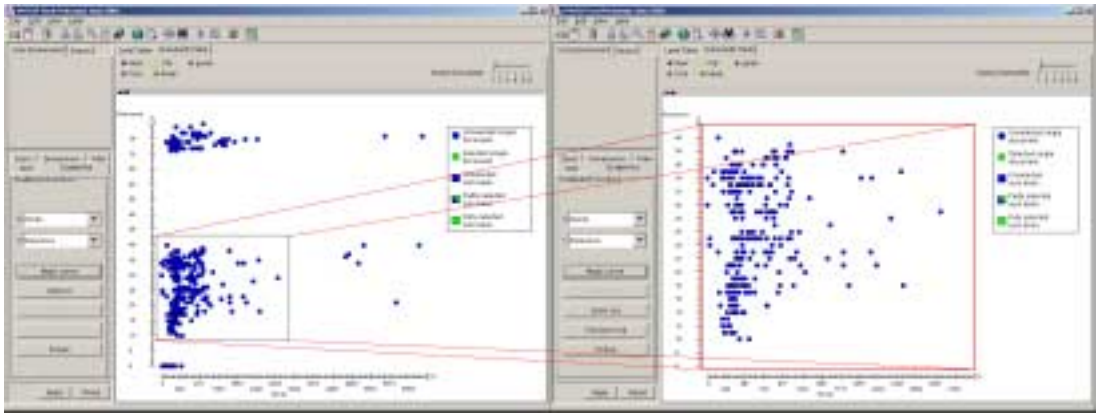

Abb. 4.8: Scatterplot mit Zooming Effekt

Rechts in Abbildung 4.8 ist der Scatterplot mit den Achsenbelegungen ‚Words' und ‚Relevance'. Mit der Maus wurde ein Rechteck aufgezogen. Dazu muss die linke Maustaste gedrückt gehalten werden und dann die Größe des Rechtecks mit der Maus aufgezogen werden. Ist ein Rechteck vorhanden, kann die Funktion ‚Zoom In' entweder aus dem Menu links im Tabbed Pane oder im Kontextmenu der rechten Maustaste angeklickt werden. Der Zooming Effekt vergrößert das gewählte Rechteck auf die gesamte Fläche des Scatterplots (Abb. 4.8). So werden die optischen Abstände der Punkte zueinander geringer und sie können besser untersucht werden. Wenn einmal gezoomt wurde, kann entweder weiter hineingezoomt werden oder eine Stufe zurückgesprungen werden.

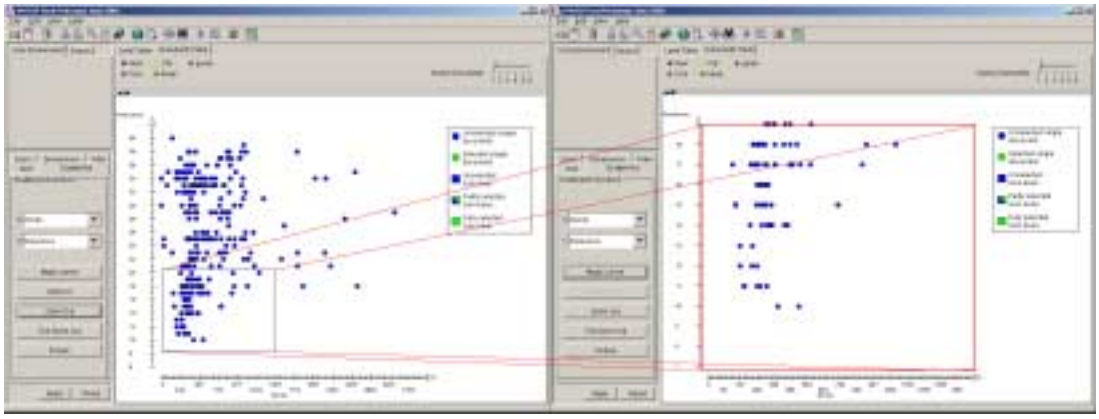

Abb. 4.9: Scatterplot beim zweiten Zoom

Mit der Betätigung des ‚Full Zoom Out' Buttons steht eine weitere Möglichkeit zur Verfügung. Der Scatterplot wird dann in den Zustand zu Beginn, ohne Zoom versetzt. Allerdings fallen durch einen Zoom einige Dokumente aus dem Sichtbereich. Das Problem ist, dass der Benutzer dies nach einem Zoom nicht mehr erkennen kann. Die Erweiterung, die im Ausblick dieser Arbeit beschrieben ist, zeigt Punkte, die wegen eines Zooms aus der Visualisierung fallen, am Rand der Matrix des Scatterplots an. Dies ist eine zusätzliche View Transformation, eine Art der Distortion (dt.: Verzerrung), die hier als Problemlösung genutzt werden kann.

Der Scatterplot bietet auch noch weitere Möglichkeiten. Zum Beispiel kann eine Selektion ganzer Punktehaufen erfolgen. Durch das Aufziehen eines Rechtecks können die im Rechteck liegenden Punkte über das Kontextmenu (Aufruf mit der rechten Maustaste) alle selektiert oder deselektiert werden. In Abbildung 4.10 sieht man, wie die Selektion eines Punkteclusters vorgenommen wurde. Diese Selektion wirkt sich

auch auf die tabellarische Visualisierung über dem Scatterplot aus. Die im Scatterplot selektierten Dokumente werden auch in der Tabelle als selektiert angezeigt.

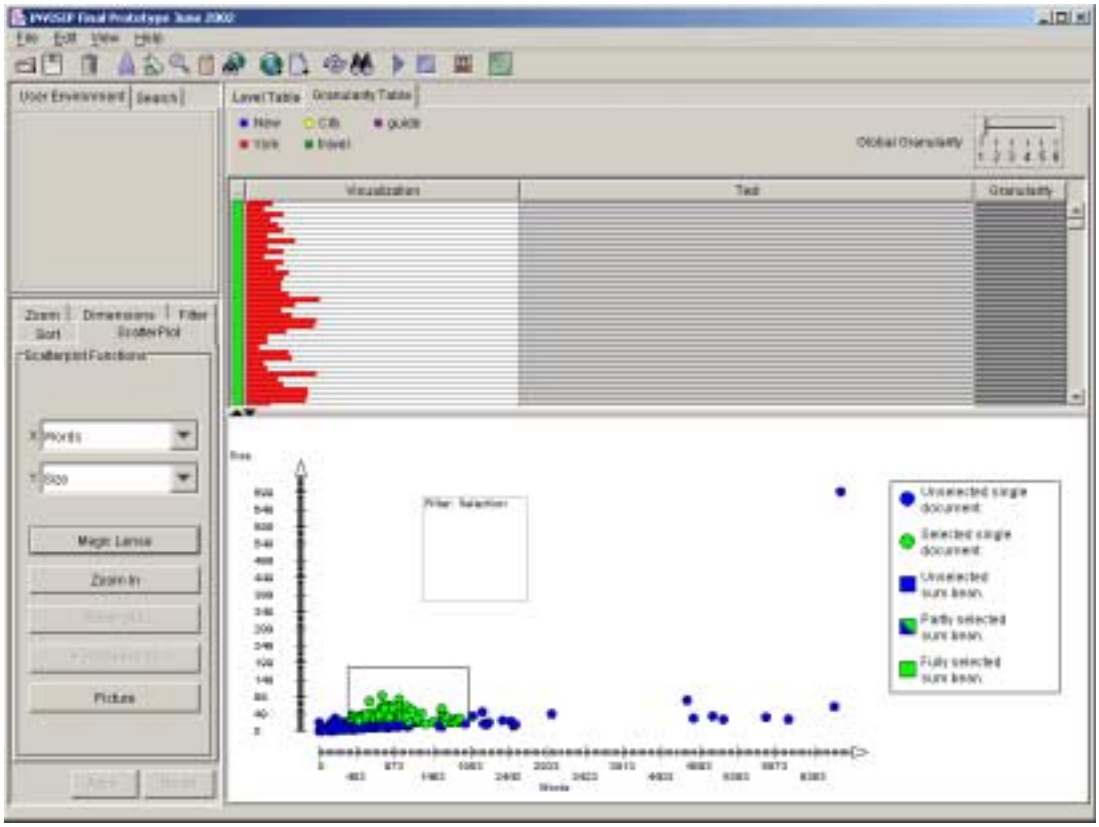

Abb. 4.10: Scatterplot und Selektion

4.2.3 Interaktion des Scatterplots mit der Granularity Table

Parallel zum Scatterplot wurde die Granularity Table [6] entwickelt. Im INVISIP-Prototypen sind beide Komponenten, der Scatterplot und die Granularity Table, in einem Splitpane untergebracht.

Bei der Entwicklung der Granularity Table wurde eine Idee zur Visualisierung von Quelle [9] verfolgt. Jede Tabellenzeile repräsentiert ein Dokument. Die Zeilen haben jeweils sechs verschiedene sogenannte Granularitätslevel, die einzeln mittels eines Sliders in der rechtesten Spalte verändert werden können. In der ersten, der Selektionsspalte sind die Dokumente als runde Punkte visualisiert. Hier können die Dokumente angeklickt und damit selektiert werden. Die zweite Spalte mit dem Header Visualization zeigt im ersten Level die Gesamtrelevanz an. Die Textspalte zeigt genauere Informationen je nach Granularitätslevel an.

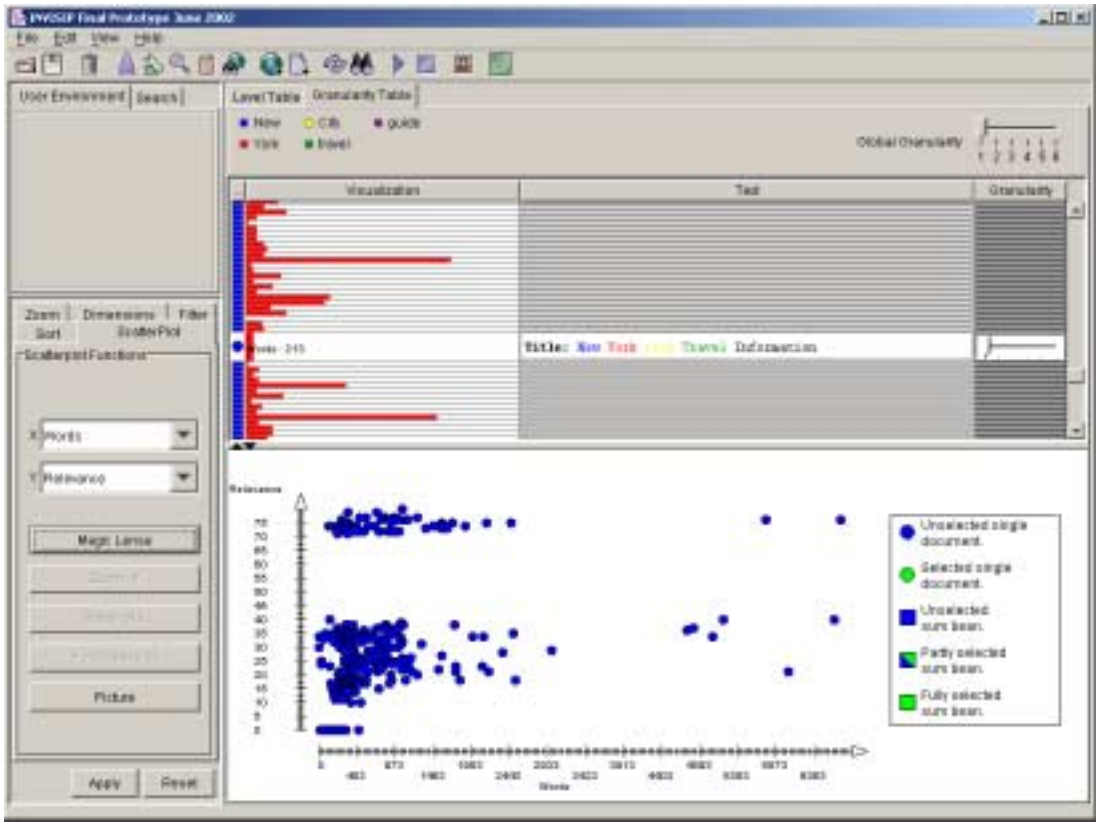

Abb. 4.11: Scatterplot und Granularity Table

Über dem Tabellenheader befindet sich noch eine Legende mit den Suchworten auf der linken Seite. Auf der rechten Seite ist der globale Granularitätsslider. Mit diesem kann der Anwender die Level alle Dokumente in der Tabelle verändern, ohne jeden einzelnen Slider der jeweiligen Zeile zu bedienen. Zu Beginn sind alle Tabellenzeilen in der ersten Stufe. Das bedeutet, dass nur die Gesamtrelevanz der Dokumente in der Visualisationsspalte angezeigt wird. Außerdem sind in diesem ersten Level die Zeilen so niedrig, dass der Titel nur beim Mouse-Over Effekt angezeigt wird. Dieser Mouse-Over Effekt ist eine vereinfachte Table Lens und vergrößert die Höhe der Tabellenzeile, über der sich der Mauszeiger gerade befindet (Abb. 4.11). Wie schon oben bei den Abbildungen des Scatterplots zu sehen war, kann auch die Granularity Table allein angezeigt werden. Dies wird durch einen Klick auf den schwarzen, nach unten zeigenden Pfeil im Splitpane erreicht (Abb. 4.12).

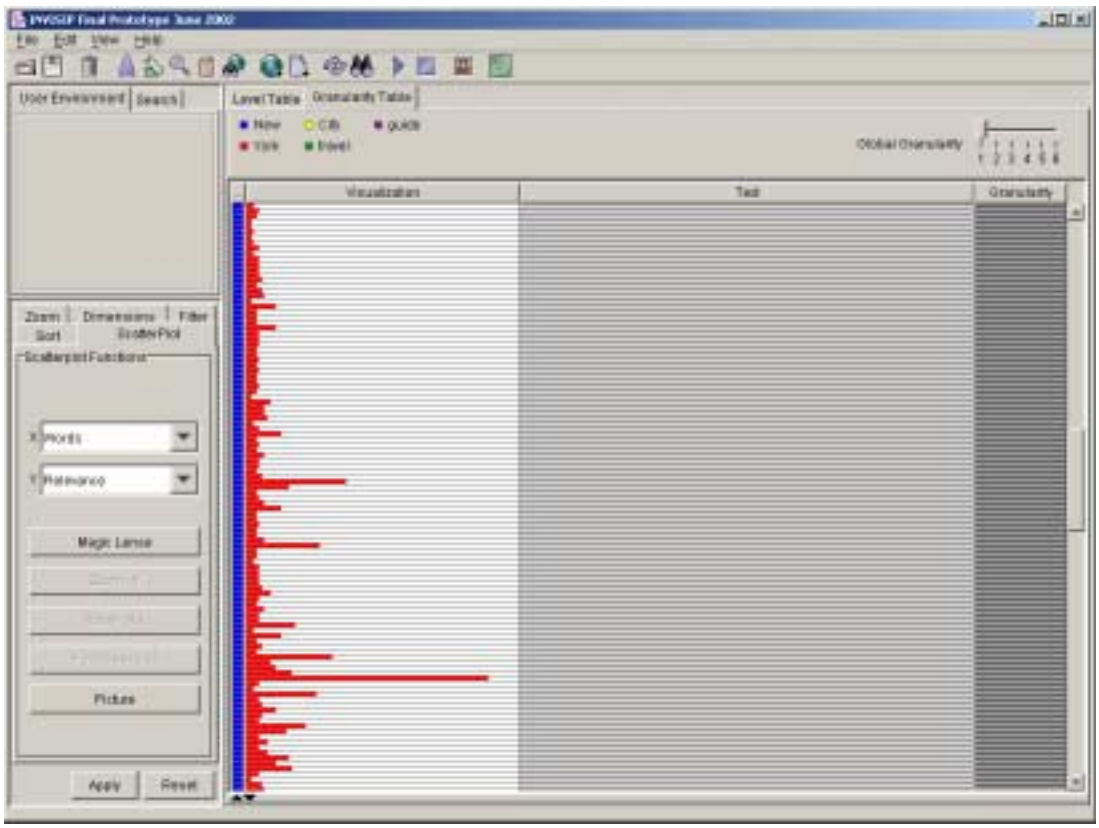

Abb. 4.12: Scatterplot und Granularity Table

Über einen Klick auf den Spaltenkopf mit dem Namen ‚Visualization' werden die Dokumente entsprechend ihrer Relevanzen aufsteigend sortiert, nach einem weiteren Klick absteigend. Die folgenden Abbildungen zeigen die verschiedenen Levelzustände der Tabellenzeilen. Man sieht wie die Anzeige der Textspalte mit zunehmendem Level immer Größer wird. Es werden immer mehr Informationen angezeigt, bis schließlich im sechsten Level der Volltext des Dokuments zu lesen ist. Die Visualizationspalte zeigt verschiedene grafische Informationen zum Dokument an. Im ersten Level die Gesamtrelevanz als roten Balken. Vom zweiten bis zum vierten Level werden Gesamtrelevanz, Wörterzahl und Größe in Kilobytes angezeigt. Im fünften Level wird eine Tilebar für jedes Dokument angezeigt. Die verschiedenen Relevanzverteilungen der Schlüsselwörter sind dadurch zu sehen. Außerdem bietet diese Visualisierung eine Navigation über den Klick an irgendeine Stelle der Tilebar. Sind zum Beispiel interessante Schlüsselwortgruppierungen an einer Stelle der Tilebar zu sehen, kann der Anwender einfach auf die Stelle in der Tilebar klicken. Der Volltext in der Anzeige der Textspalte scrollt dann automatisch an die gewünschte Stelle. Im sechsten Level wird zur besseren Ansicht des Volltextes die Visualizationspalte mit der Textspalte zusammengelegt.

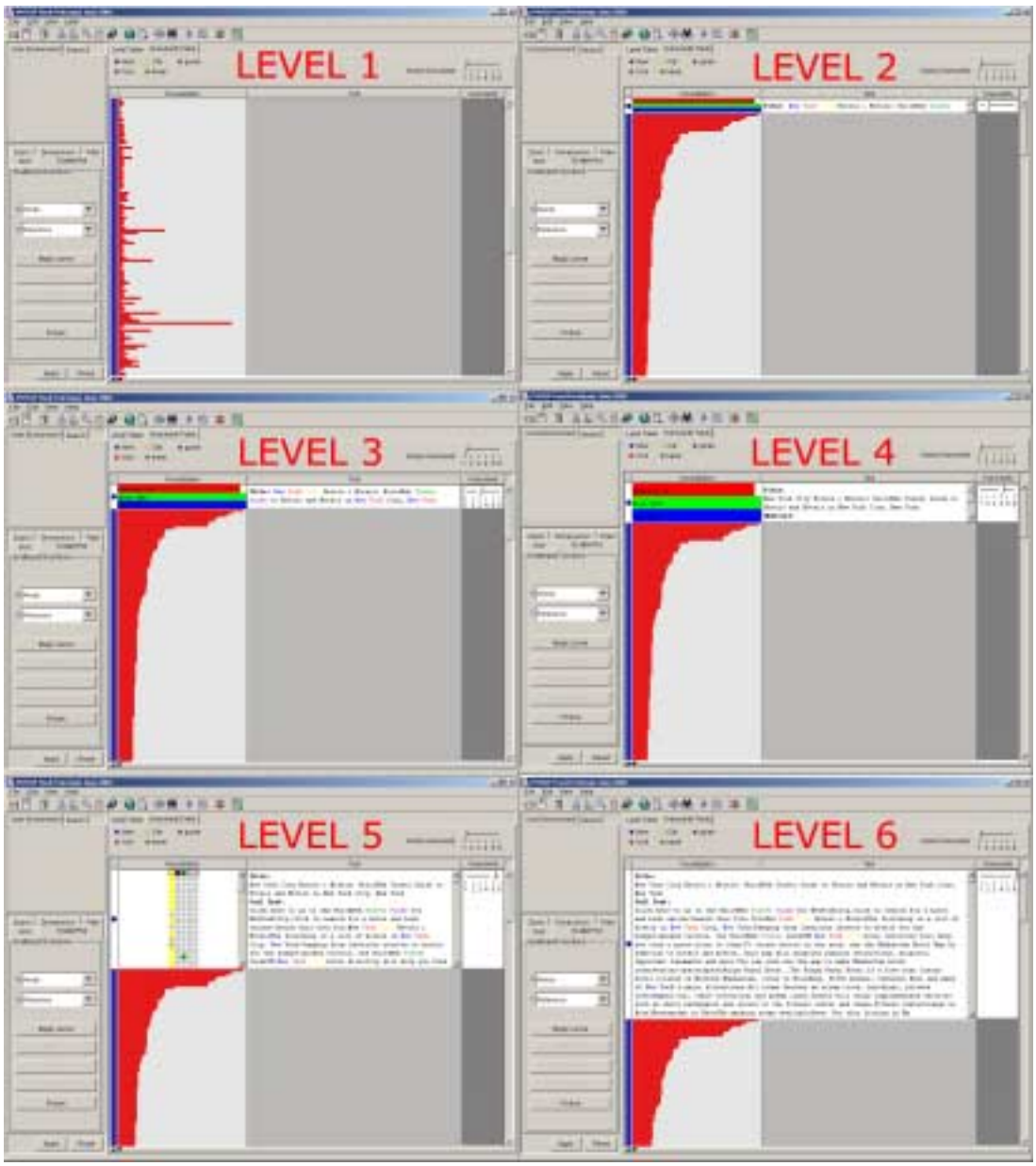

Abb. 4.13: Granularity Table; oberste Zeile in allen sechs Granularitätsstufen
(die rote Schrift ‚Level 1 bis 5' gehört nicht zum Programm, sie wurde nachträglich hinzugefügt und zeigt nur den Level der jeweiligen Abb. an)

Die gleichzeitige Anzeige der Granularity Table und des Scatterplots ermöglichen dem Anwender mit beiden Visualisierungen zu gleicher Zeit zu arbeiten. So können ohne Umschaltungen die Vorteile beider Visualisierungen genutzt werden. Ist für eine Aufgabe nur eine der Visualisierungen relevant, kann diese im ganzen Splitpane allein angezeigt werden. Zusätzlich können weitere Mehrwerteffekte durch die Kommunikation der beiden Visualisierungen entstehen. Wenn ein Dokument im Scatterplot selektiert wird, so ist es bei gleichzeitiger Benutzung der Visualisierungen sinnvoll, dass das gleiche Dokument auch in der Granularity Table selektiert

erscheint (umgekehrt gilt das auch). Diese Interaktion der beiden Komponenten wurde in diesem Projekt so implementiert. Bei der Selektion eines Dokumentes im Scatterplot wird das selektierte Dokument auch in der Tabelle als selektiert angezeigt. Außerdem rückt es an die oberste Stelle der Tabelle, damit es der Anwender sofort im Blick hat. Weitere übergreifende Funktionen werden beim Zooming und beim Einsatz der Magic Lens eingesetzt. Fallen Dokumente durch einen Zoom aus der Ansicht des Scatterplots, dann wird die Zeilenhöhe dieser Dokumente in der Tabelle auf das mögliche Minimum (entspricht einem Pixel) reduziert.

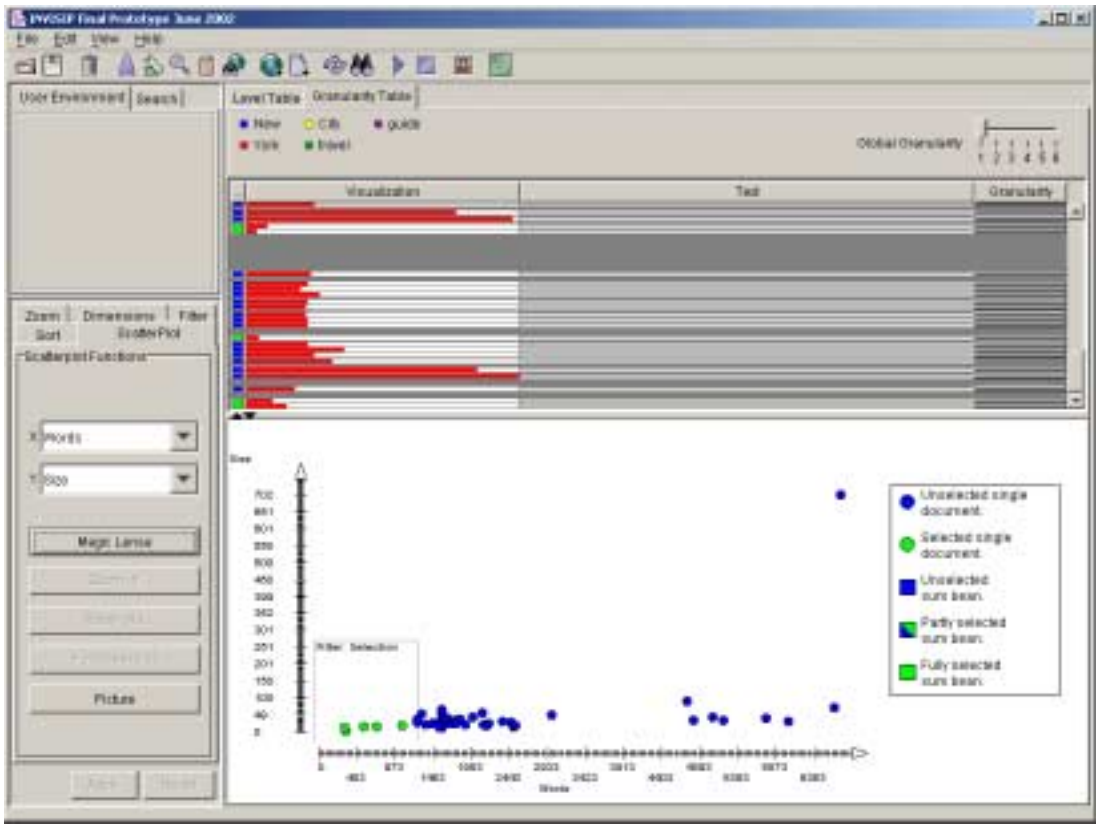

Abb. 4.14: Interaktion zwischen Granularity Table und Scatterplot

Die gleiche Kommunikation zwischen Tabelle und Scatterplot findet bei der Filterung der Dokumente durch einen Magic Lens Filter, wie in Abbildung 4.14 statt. Sind Punkte durch die Magic Lens ausgeblendet, werden die Zeilenhöhen der entsprechenden Dokumente in der Tabelle auf einen Pixel verkleinert.

Eine weitere Interaktion ist durch einen Rechtsklick auf einen Punkt im Scatterplot möglich. Dadurch wird ein Kontextmenu angezeigt, das einen Granularitätsslider (genau wie in der Granularity Table) enthält. Wird dieser verschoben, rückt in der Tabelle das entsprechende Dokument in den Kontext und ändert seine Granularität je nach Einstellung des Sliders im Scatterplot.

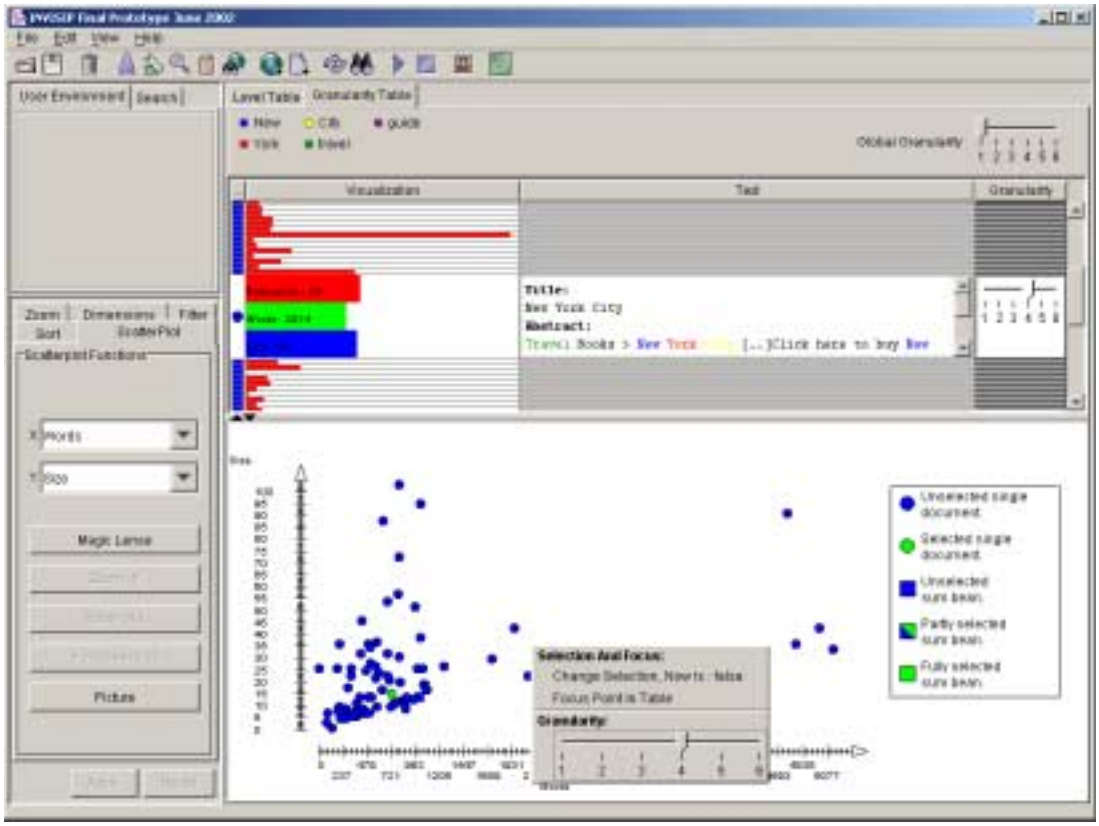

Abb. 4.15: Interaktion zwischen Granularity Table und Scatterplot

4.3 Grundlagen für die Implementation

Der INVISP-Prototyp sollte aus den Komponenten des Datenmodells, dem Scatterplot und der Granularity Table bestehen. Das Datenmodell wurde von einem Student schon vor Beginn des Projektes entwickelt. In den ersten Treffen wurden die bisherigen Projekte vorgestellt und Verbesserungsvorschläge für den INSYDER-Prototypen erarbeitet. Parallel zur Implementation des Scatterplots wurde auch die Granularity Table programmiert. Eine Basis des Projektes waren die Erfahrungen aus der Programmierung des INSYDER-Prototypen. Der Scatterplot des INSYDER bot leider keinen Re-Use des Quellcodes an, da die Programmierung weder auf wichtigen Konzepten der Programmiersprache Java basierte noch übersichtlich genug war. Auch die Kommentare waren mager oder gar nicht vorhanden. Ein solches Projekt, wie den INSYDER-Scatterplot, in ein anderes Projekt zu integrieren, stellte sich als zu komplexe Aufgabe heraus. Die Programmierung war nicht modular organisiert und die Möglichkeiten der Objektorientierung wurden nicht bedacht. Somit war jeglicher Quellcode des INSYDER-Scatterplots mit sehr vielen anderen Programmteilen, die nichts mit dem Scatterplot zu tun hatten, verstrickt und verflochten. Es war also kaum möglich ein Projekt zu generieren, das nur den lauffähigen Scatterplot des

INSYDER-Prototypen enthält. Hinzu kam das neue Datenmodell, das als Basis für das INVISIP-Projekt entwickelt wurde. Der alte Scatterplot hätte sehr umständlich umgebaut werden müssen, um das Datenmodell zu benutzen. Eine Neuimplementation des Scatterplots war also unumgänglich. Die Programmierung des gesamten Projektes stellte sich nach ersten Entwicklungen als komplexer und schwieriger heraus als erwartet. Das Datenmodell unterstützte die Visualisierungen nicht in dem benötigten Maß. So mussten Performanceverluste durch weitere Berechnungen der Daten in den Visualisierungen hingenommen werden, da das Datenmodell nicht die gewünschten Daten liefern konnte. Von großer Bedeutung war die Teamarbeit in diesem Projekt. Die Komponenten und Schnittstellen mussten aufeinander abgestimmt werden. Die Funktionalitäten und Programmiertechniken, sowie auch der Schreibstil des Quellcodes mit Kommentaren, wurden abgesprochen. Dies gewährleistete eine saubere Programmierung, die jederzeit von allen Programmierern ohne Verständnisprobleme eingesehen werden konnte. Der erste Versuch der Programmierung des neuen Scatterplots schlug fehl, da eine straight-forward Programmierung versucht wurde. Das Resultat war eine ähnliche Unübersichtlichkeit wie beim INSYDER-Scatterplot. Der Neubeginn richtet sich nach den objektorientierten Konzepten und Software Patterns der Programmiersprache Java.

4.3.1 OO-Programmierung in Java

Im Gegensatz zur modularen Programmierung kommt in der OO (Objekt-Orientierung) die Idee der abstrakten Klassen hinzu. Einige Klassen sind normal implementiert, die anderen sind abstrakt. Das heißt, sie können nicht instantiiert werden und enthalten Methoden und Variablen, die von anderen Klassen benutzt werden können. Das Konzept der Vererbung wird oft benutzt, um implementierte Klassen von abstrakten Klassen erben zu lassen. Die erbenden Klassen übernehmen dann alle Eigenschaften der Oberklassen. Mittels der Vererbung können Eigenschaften übernommen geändert, hinzugefügt oder auch abgelegt werden.

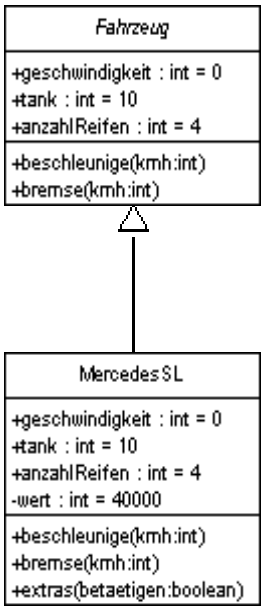

Abb. 4.16: UML Vererbung zwischen Klassen

In Abbildung 4.16 wird das Konzept der Vererbung und der Abstraktion eingesetzt. Die Klasse Fahrzeug ist abstrakt. Von ihr können die konkreten Fahrzeugklassen abgeleitet werden. Ein Beispiel wurde hier abgeleitet, die Klasse mit dem Namen MercedesSL. Jede von Fahrzeug abgeleitete Klasse erbt alle Eigenschaften der Fahrzeugklasse. So übernimmt die Klasse MercedesSL zum Beispiel die Methoden ‚beschleunige()' und ‚bremse()', sowie die Variablen ‚geschwindigkeit', ‚tank' und ‚anzahlReifen'. Hinzu kommen die Bedienung der Extras über die Methode ‚extras()' und die Variable ‚wert', die den Gesamtwert des Fahrzeugs angibt. Eine weitere Funktion der Klassen ist die Kapselung der Datentypen. Je nach Bedarf können so eigene Klassen mit den benötigten Variablen, Methoden und Eigenschaften entworfen werden. Die Funktion und die benutzten Datentypen in den Klassen können versteckt werden. Objekte kommunizieren miteinander und haben Auswirkungen aufeinander. Durch Aufruf von Methoden ändern Objekte ihre Eigenschaften und Zustände. Die Kommunikation erfolgt meistens über Interfaces (Schnittstellen). Die Vererbung ist ein wichtiges und grundlegendes Konzept der OO-Programmierung in Java.

Die statische und dynamische Bindung der Datentypen sind zwei weitere Eigenschaften der Objektorientierung. Während die statische Bindung vom Compiler übernommen wird, erfolgt die dynamische Bindung zur Laufzeit des Programms. Bei der dynamischen Bindung wird der Typ der instantiierten Variable zur Laufzeit festgelegt. In einer IF-ELSE Verzweigung kann einer Variablen je nach Bedarf eine andere

Instanz zugewiesen werden. Der Nachteil der dynamischen Bindung ist ein geringer Zeitverlust. Außerdem kann nicht mehr so gut vorhergesagt werden wie das Programm reagiert, da die Komplexität ansteigt. Der Polymorphismus (dt.: Vielgestaltigkeit) bedeutet in Java, dass ein Objekt mehrere Gestalten annehmen kann. Dies geschieht durch die Mehrfachvererbung über Interfaces. Implementiert eine Klasse ein Interface, das heißt sie muss bestimmte Methoden anbieten, dann ist diese Klasse von einem bestimmten Typ von Klassen. Eine Klasse kann mehrere Interfaces implementieren, kann also die Gestalt mehrerer Typen von Klassen haben.
Die abstrakte Kopplung ist das Basiskonzept der Software-Patterns und wird auch im Scatterplot eingesetzt. Die Kopplung der Klasse A an Klasse B funktioniert über eine abstrakte Klasse AB.

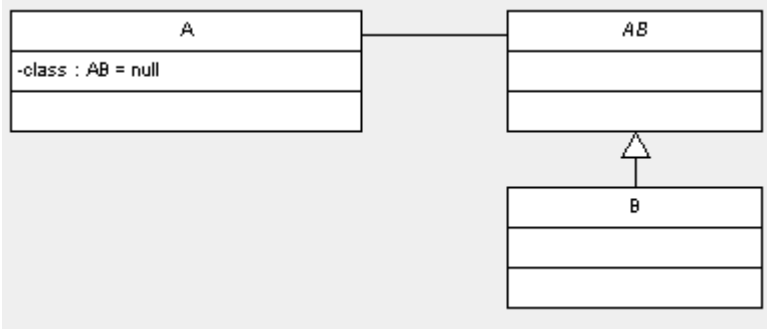

Abb. 4.17: UML der abstrakten Kopplung

Die Klasse A muss sich nicht um die Subklassen von Klasse AB kümmern, da sie die Methoden der Subklassen über die Klasse AB kennt. Welche Subklassen also der Klassenvariablen AB in Klasse A zugeordnet werden, ist in Klasse A nicht von Interesse. Auf dieser Basis bauen die beiden wichtigsten Software Patterns auf. Das Unification Pattern und das Separation Pattern [25,26]. Grundlegend für das Verständnis dieser Konzepte ist das System der Template- und Hookklasse.

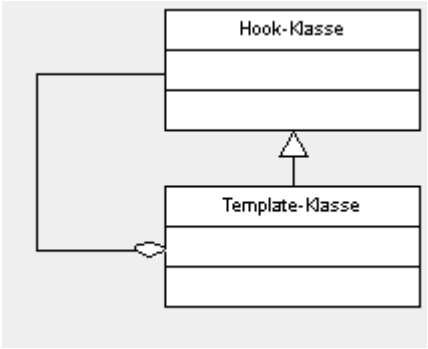

Abb. 4.18: UML der abstrakten Kopplung

Die Templateklasse ist normalerweise konkreter als die Hookklasse und enthält keine variablen Teile. Sie kann von der Hookklasse abgeleitet sein, aber nicht umgekehrt. Die Hookklasse enthält die Hookmethode(n), die potenziell variable Teile enthalten und mittels Vererbung geändert werden können. Beim Unification Pattern fallen die Hook- und die Templateklasse zusammen zu einer Klasse. Eine einzige Klasse übernimmt also die Aufgaben für Hook- und Templateklasse. Beim Separation Pattern werden Hook und Template getrennt. Ein Beispiel ist die abstrakte Kopplung aus Abb. 4.17. Die Klasse A ist die Templateklasse und enthält keinen anpassbaren Quellcode. Die Hookklasse ist AB. Sie kann Methoden enthalten, die evtl. nicht implementiert sind und angepasst werden können, indem eine Subklasse gebildet wird. Mit dieser angepassten Subklasse arbeitet nun Klasse A, ohne von der Veränderung Kenntnis nehmen zu müssen.

4.3.2 Verwendete Software Patterns

Durch den Einsatz von Software-Patterns können Programme so gestaltet werden, dass zu späterem Zeitpunkt leichter Änderungen möglich sind. Dies wird auf der Basis des Separation Patterns erreicht. Die Komponenten, die möglichst unabhängig sein sollen, werden so gut es geht voneinander entkoppelt.

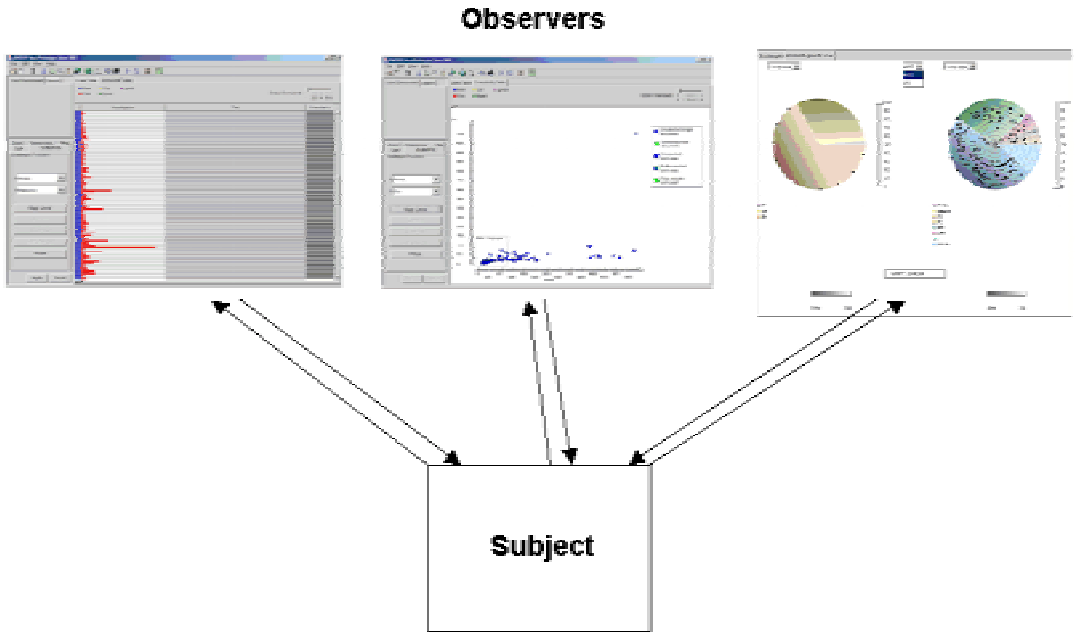

Abb. 4.19: Trennung von Sichten und Daten durch das Observer Pattern

Das in diesem Scatterplot und im INVISIP-Framework eingesetzte Observer Pattern ist auf der Basis des Separation Patterns aufgebaut. Dieses Pattern ist das wichtigste

Pattern des Projektes, da die Möglichkeit geschaffen wird, alle Visualisierungen vom Datenmodell zu entkoppeln. Die Granularity Table, der Scatterplot und andere Visualisierungen können getrennt entwickelt werden und anschließend mit dem Datenmodell verbunden werden.

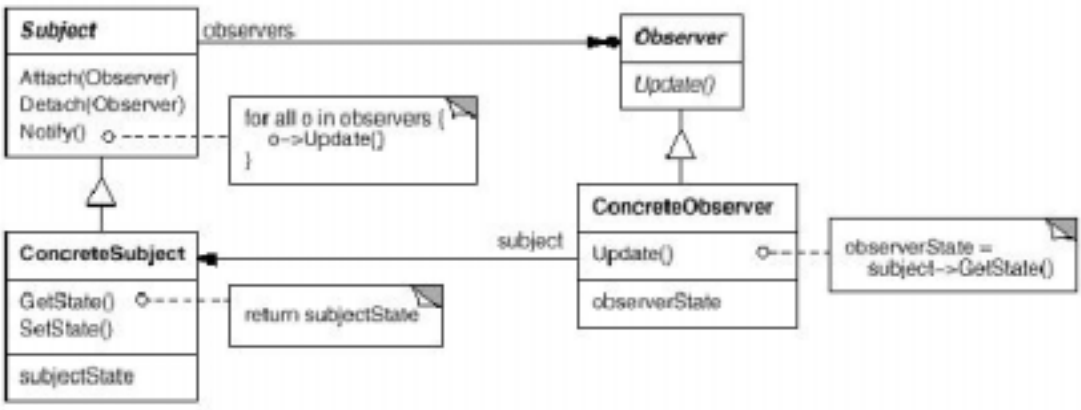

Abb. 4.20: UML Struktur des Observer Patterns, entnommen aus [25]

Das Observer Pattern wird auch Model-View-Controller Pattern genannt. Das MVC Konzept ist schon seit längerer Zeit ein Begriff in der Informatik. Das Eventmodell der Programmiersprache Java ist beispielsweise darauf aufgebaut. Das bei Java Beans eingesetzte Pattern ist allerdings eher ein Model-View Pattern, da der Controller Teil des Models ist und nicht in einer eigenen Klasse agiert. Das Observer Pattern wird auch innerhalb des Scatterplots eingesetzt. Durch die Zuordnung der verschiedenen Klassen des Scatterplots zu den Komponenten der Abbildung 4.20 erkennt man die Struktur. Die DefaultMainModel entspricht dem ConcreteSubject und die ScatterMatrix dem ConcreteObserver. Die Interfaces Subject und Observer entsprechen dem InterfaceMainModel und dem ScatterMatrixInterface (siehe auch Abb. 5.5). Es gibt zwei verschiedene Varianten des Observer Patterns. Die Variante, die auch beim Scatterplot eingesetzt wurde, ist das Push Model. Das Subject sendet hier alle benötigten Informationen zu den Observern. Dabei spielt es keine Rolle, ob der jeweilige Observer die Informationen benötigt oder nicht. Die Alternative ist das Pull Model. Bei dieser Variante sendet das Subject nur die Information einer Änderung an die Observer. Die Observer sind nun selbst dafür zuständig die benötigten Informationen für eine Aktualisierung zu erhalten. Das Pull Model wird im Konzept des Datenmodells des INVISIP-Frameworks angewendet.

Das Singleton Pattern ist ein weiteres Software-Pattern, das häufig eingesetzt wurde. Durch die Objektorientierung (OO) ist es in Java möglich, beliebig viele Instanzen

von Klassen zu erzeugen. Manchmal möchte man jedoch, dass nur eine einzige Instanz einer Klasse vorhanden ist. Wenn zum Beispiel mehrere Visualisierungen mit einem Datenmodell arbeiten sollen, macht es keinen Sinn eine zweite Instanz des Datenmodells zu erzeugen. Sonst würden die Visualisierungen nicht mehr mit dem gleichen Modell arbeiten und Veränderungen bzw. Aktualisierungen würden nur noch die Visualisierungen erhalten, die am richtigen Datenmodell angekoppelt sind. Um sicher zu gehen, dass man nur mit der einen Klasseninstanz arbeitet, kann das Singleton Pattern angewendet werden. Dabei wird über den Klassenkonstruktor sicher gestellt, dass nur eine Instanz erzeugt wird. Existiert einmal eine Instanz, dann merkt dies der Konstruktor beim Aufruf. Er gibt die vorhandene Instanz zurück und erzeugt keine neue. Damit ist gesichert, dass nur eine Instanz erzeugt und auch verwendet wird.

Im Datenmodell von INVISIP ist noch ein weiteres Software-Pattern eingesetzt worden. Es nennt sich Builder-Pattern. Dieses Pattern wird eingesetzt, wenn Komponenten aus Klassen variabel zusammengestellt werden sollen. Die Konstruktion eines komplexeren Objektes wird dabei von seiner Repräsentation getrennt. Dadurch kann der gleiche Konstruktionsprozess zu verschiedenen Repräsentationen der gebauten Objekte führen. Die Klasse DataElementBuilder des Datenmodells entspricht dem ConcreteBuilder. Das genaue Zusammenspiel der Komponenten des Datenmodells wird in dieser Arbeit nicht erläutert, da dies zu weit führen würde.

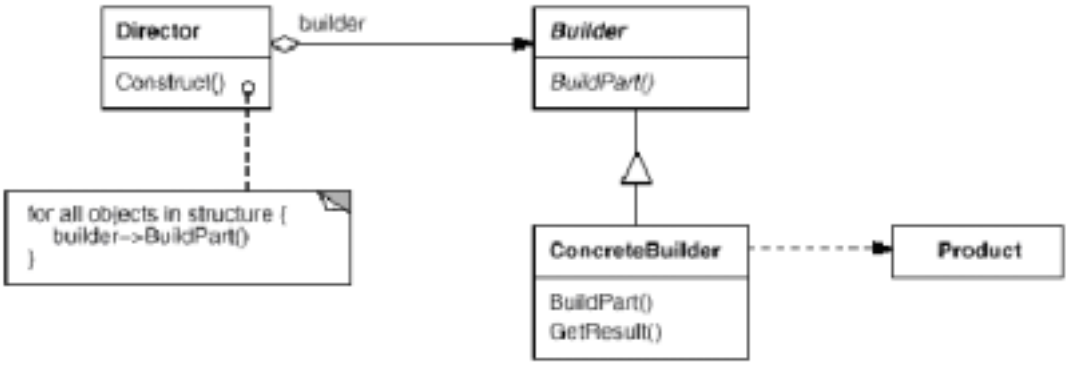

Abb. 4.21: UML Struktur des Builder Patterns, entnommen aus [25]

4.3.3 Komponententechnologie Java-Swing

Swing ist neben AWT eine von SUN Microsystems entwickelte Java-Bibliothek zur Entwicklung von Benutzeroberflächen. Sie gehört zum Java-Standard und enthält vorgefertigte Komponenten, wie Button, Slider, Scrollbar, Window, Menu und viele

mehr. Die Entwickler müssen nicht bei jeder Neuentwicklung die Benutzeroberfläche neu programmieren, sondern sie können sich der Bibliotheken bedienen. So können die Oberflächen schneller und kostengünstiger entwickelt werden. Im Gegensatz zu AWT enthält Swing mehr Komponenten. Diese unterscheiden sich gegenüber den AWT-Komponenten im Umfang ihrer Funktionen und der Programmierung. Während AWT-Komponenten von Hand programmiert werden müssen, gibt es bei Swing die Möglichkeit, visuelle Programmierung mit Hilfe eines GUI zu verwenden. Ein Beispiel wäre der JBuilder von Borland, mit dessen Hilfe Oberfläche visuell programmiert werden können. Alle im mittleren Fenster sichtbaren Objekte sind Java Beans. Sie können visuell erstellt und angepasst werden. Der Quelltext wird automatisch angepasst, wobei teilweise viele unnötige Anweisungen verbleiben. Eine spätere Überarbeitung des Quelltextes ist notwendig, da die funktionale Programmierung nicht oder nur unzureichend von grafischen Designoberflächen wie dem JBuilder-Designer unterstützt wird.

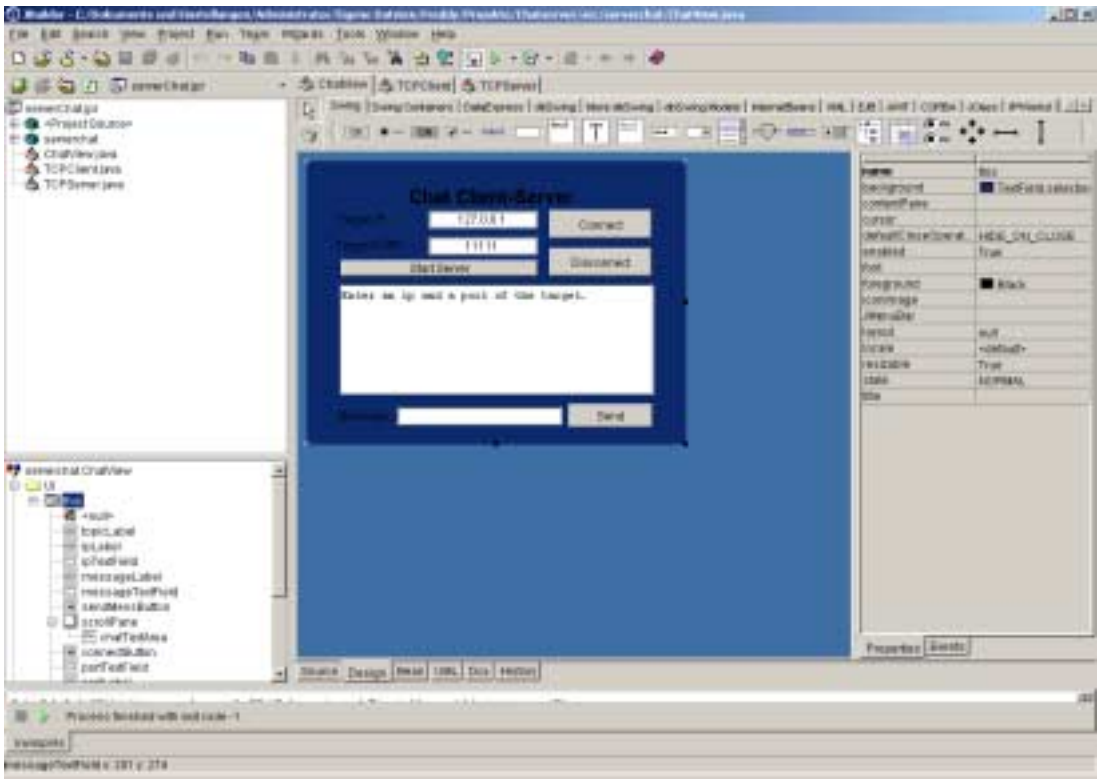

Abb. 4.22: Entwicklung einer Oberfläche mit dem Designer des JBuilders von Borland (Screenshot)

Die visuelle Programmierung funktioniert, weil die Swing-Komponenten sogenannte Properties (dt.: Eigenschaften) haben, die in den jeweiligen Komponenten intern gespeichert werden (Datenkapselung). Auch ohne die Konzepte der Programmiersprache Java würden Probleme auftreten. Ein erheblicher Unterschied besteht auch zwi-

schen den Eventmodellen von AWT und Swing. Sie sind für die Behandlung von auftretenden Aktionen, wie z.B. Maus-Events, zuständig.

Visuelle Swing-Komponenten zur Programmierung von Oberflächen werden Java Beans genannt. Eine normale Java Bean besteht aus drei Hauptteilen, dem sichtbaren Teil, der Schnittstelle zum Model der Bean und Methoden zur Verknüpfung der Bean mit anderen Beans. Der sichtbare Teil besteht einfach aus der sichtbaren Komponente (z.B. eine JList). Die JList kann ein Model haben, um die Daten abzuspeichern. Hier existiert in der Javabibliothek ein schon vorgefertigtes DefaultListModel. In diesem werden die in die Liste eingefügten Werte gespeichert. Die Liste speichert ihre Daten also nicht selbst, sondern übergibt diese Aufgabe an ihr Model. Verknüpfende Komponenten sind zum Beispiel die Listener. Diese reagieren auf Änderungen der Komponente und teilen diese anderen Objekten mit, die dann ihrerseits reagieren können. Jeder dieser drei Teile kann, muss aber nicht implementiert sein. Die Vorteile der Java Beans sind die modulare Wiederverwendbarkeit, die auch beim INVISIP Projekt von Vorteil ist. Die Komponenten konnten einzeln getestet werden, bevor sie in das gesamte Framework eingefügt wurden. Somit sind die Komponenten auch sehr leicht austauschbar und die verschiedenen Objekte können getrennt und von verschiedenen Personen programmiert werden. Das Zeichnen der Java Beans auf den Bildschirm wird von der Java Virtual Machine (JVM) übernommen und funktioniert nach dem Hierarchiekonzept. Jede Komponente kann weitere Komponenten enthalten. Wird das Zeichnen von einer Komponente aus gestartet, zeichnet diese zuerst sich selbst und gibt dann den Zeichnenbefehl an alle enthaltenen Komponenten weiter. So zeichnet sich dann jede Komponente selbst bis die unterste Hierarchiestufe erreicht ist. Die Granularity Table und der Scatterplot konnten separat programmiert werden, da sie eigenständige Objekte sind und auch ohne die anderen Komponenten lauffähig wären. Die einzigen Anpassungen müssen bei der Anbindung des Datenmodells gemacht werden, da dieses je nach Bedarf unterschiedlich ist. Wenn jedoch geeignete Interfaces zur Verfügung stehen gibt es keine Probleme. Es müssen dann nur die entsprechenden Methoden implementiert werden und die Kommunikation der Objekte funktioniert.

4.3.4 Datenmodell, XML

Grundlage des INVISIP Projektes ist das Datenmodell, das von einem Studenten entwickelt wurde. Damit in den Visualisierungen die Daten überhaupt dargestellt werden können, muss ein XML-Parser die Datenfiles einlesen und parsen. So einen Parser ist im Datenmodell enthalten. Zum Test wurden alte Testdatenfiles des INSYDER Projektes benutzt. Diese befinden sich in einer Art XML-Format. Es fehlt jedoch ein Header und eine DTD. Diese Testdatensätze enthalten Metadaten über WWW-Dokumente. Das Datenmodell hat spezielle Container für die verschiedenen Daten, die in den Datentags (z.B.<Words>137</Words>) des Testfiles gespeichert sind. Die Werte werden in diesen Containern gespeichert. Der im Datenmodell enthaltene XML-Parser parst also das Datenfile, liest alle notwendigen Werte aus und speichert diese. Die nötigen Vorgänge werden von der Hauptanwendung aus gesteuert.

```
<Insyder_Export>
<Index>101</Index><Selection>false</Selection>
<DocumentRead>false</DocumentRead><Relevance>40</Relevance>
<keywords>
<New>0</New><York>0</York><City>93</City><travel>76</travel><guide>57</guide>
</keywords>
<TFIDF>0</TFIDF>
<Title>ALA Annual Conference In New York City, Travel and Tourism</Title>
<LocalURL>c:\Insyder2\Repository\f_6\p_1692.htm</LocalURL>
<URL>http://nova.umuc.edu/~kelley/webb/ny.html</URL>
<Date>30.4.1996</Date><Language>EnglishUS</Language>
<Size>1</Size><Words>137</Words>
<DocumentType>Text/Images</DocumentType>
<Servertype>Miscellaneous</Servertype>
<Abstract>ALA Annual Conference In New York City, Travel and Tourism NEW YORK, NEW
YORK: [..]The Paperless Guide to New York City About New York City WorldNet's New
York City Page Public Access Networks Corporation's New York City Reference
Page</Abstract>
<SegmentText>NEW YORK, NEW YORK:      In this section of W.E.B.B., local librarians
have assembled the materials already produced to show you the Big</SegmentText>
<RelevanceCurve>47,27,4,25,32,27,27,27,27,27,2,2</RelevanceCurve>
<keyword_segments>
<NewSegments>0,0,0,0,0,0,0,0,0,0,0,0</NewSegments>
<YorkSegments>0,0,0,0,0,0,0,0,0,0,0,0</YorkSegments>
<CitySegments>100,100,4,100,100,100,100,100,100,100,2,2</CitySegments>
<travelSegments>99,2,6,7,2,2,2,2,2,2,2,2</travelSegments>
<guideSegments>2,2,2,2,74,2,2,2,2,2,2,2</guideSegments>
</keyword_segments>
</Insyder_Export>
```

Auszug 4.23: Testdatenfile in XML-ähnlichem Format für einen Datensatz

In Abbildung 4.23 ist das File eines Testdatensatzes dargestellt. Die verschiedenen Werte, wie Selektionsstatus, Relevanz und andere sind jeweils in eigene Tags gepackt. Auch die Relevanz der einzelnen Schlüsselwörter (engl.: keywords) ist im File angegeben. Gegen Ende des Dokumentes sind noch Titel, Text und Textsegmentinformationen der Schlüsselwörter angegeben.

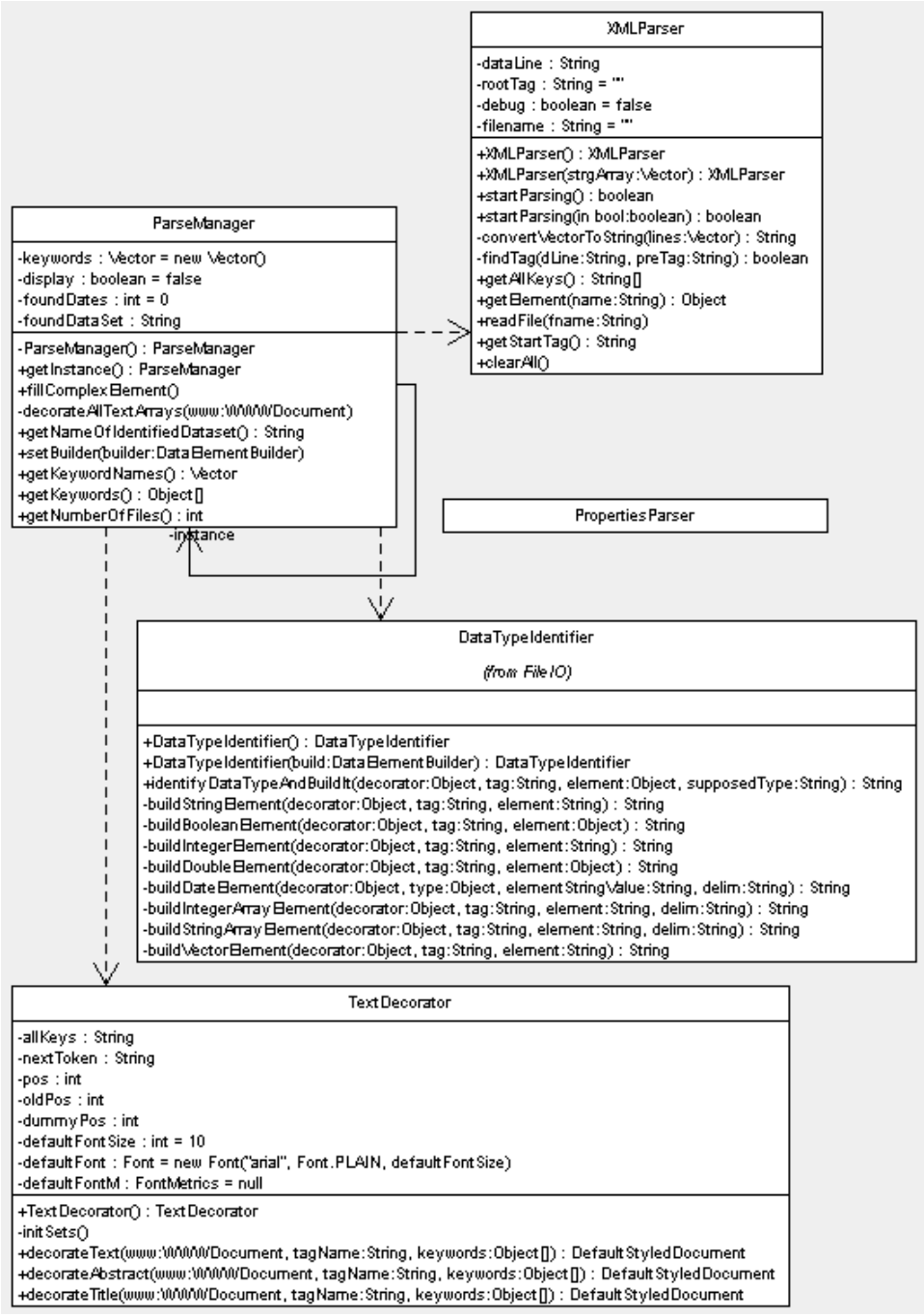

Abb. 4.24: UML der Klassen des Packetes FileIO

Um zu verstehen, wie der Parser die Informationen parst und das Datenmodell diese speichert, ist es notwendig den Aufbau zu verstehen. Abbildung 4.24 zeigt den

Grundaufbau des Paketes, das für das Parsen der Metadatenfiles zuständig ist, in Form eines UML-Diagramms. Der ParseManager wird von der Klasse Mainframe, die im Paket Views liegt, gesteuert. Die Mainframe-Klasse hat alle notwendigen Klassen und initialisiert die gesamte Anwendung. In der Methode prereadXMLData wird die Instanz des ParseManagers geholt und das Einlesen gestartet.

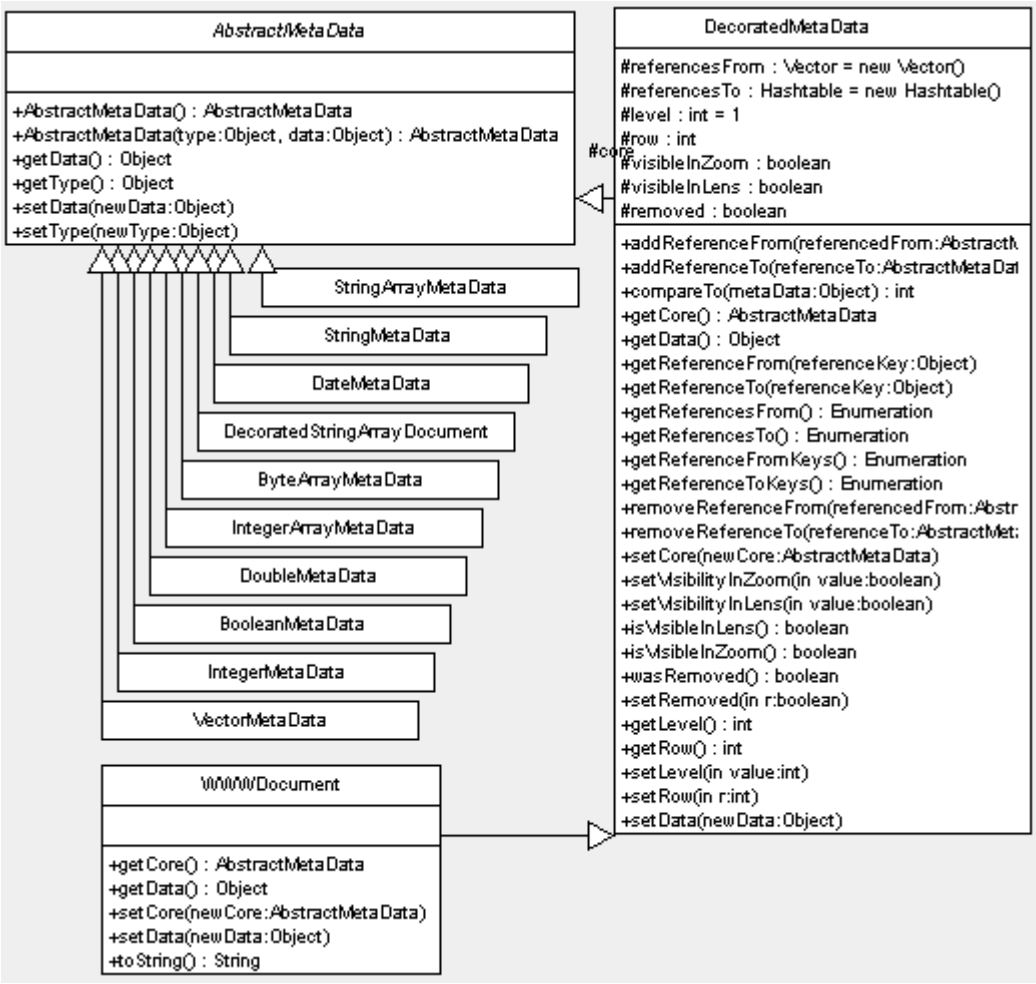

Abb. 4.25: UML der Klassen des Packetes DataModel.DataTypes

Der ParseManager benutzt die Klasse XMLParser, um die eingelesenen XML-Dateien zu parsen. Da in der Granularity Table die Schlüsselwörter farblich markiert dargestellt werden sollen und die dazu notwendigen Operationen sehr rechenintensiv sind, wird der Text der Dokumente schon beim einlesen farblich dekoriert. Dazu wird die Klasse TextDecorator benutzt. Die Klasse DataTypeIdentifier wird benötigt, um die Art der in den Dokumentenfiles enthaltenen Daten herauszufinden. Die Daten können vom Typ String, Integer, Boolean oder auch anderer Art sein. Die meisten Datentypen sind auch Basisdatentypen der Programmiersprache Java und könnten so einfach gespeichert werden. Zur internen Speicherung der XML-Daten im Datenmo-

dell werden aber vom DataElementBuilder sogenannte WWWDocuments erzeugt. Diese bestehen aus der Klasse DecoratedMetaData, die wiederum die Daten über die abstrakte Klasse AbstractMetaData in konkreten Klassen speichert. Die Klasse WWWDocument selbst, kennt also die konkreten Datentypen, die in ihr gespeichert sind, nicht. Diese Entkopplung über die abstrakte Kopplung ist eigentlich nicht sinnvoll, da so die über den DataManager anfragenden Klassen immer überprüfen müssen, von welchem Typ die gelieferten Daten sind. Um die WWWDocuments (Abb. 4.25) aufzubauen, benutzt der ParseManager die Klasse DataElementBuilder. Diese gibt wiederum weitere Aufgaben an die Klasse DecoratedDataManager weiter. Das eigentliche Kernstück des Datenmodells ist jedoch die Klasse DataGlobalModel. Auch sie benötigt die Klasse DataElementBuilder, um die Daten verändern zu können. Außerdem implementiert sie das Interface Model. Damit sind für die verschiedenen Visualisierungen, die alle dieses Datenmodell benutzen, nur die notwendigen Methoden sichtbar. Die Kommunikation der Views und des Datenmodells funktioniert über das Model-View-Controller (MVC) Konzept [25]. Es erlaubt die Trennung des Datenmodells von seiner Visualisierung. Durch dieses Konzept ist es möglich verschiedene Visualisierungen mit einem Datenmodell arbeiten zu lassen. Genau dies ist hier der Fall. Der Scatterplot und die Granularity Table sind beides Visualisierungen, die auf dem gleichen Datenmodell arbeiten. Zum INVISIP Framework sollen später einmal noch weitere Visualisierungen hinzukommen. Sie müssen sich bei der Instanz des Datenmodells anmelden und werden bei Änderungen automatisch vom Datenmodell informiert. So kann eine eventuell notwendige Aktualisierung ihrer Darstellung ausgeführt werden. Die Klasse des Datenmodells ist außerdem nach dem Singleton Pattern [25] aufgebaut. Dies gewährleistet, dass nur eine Instanz des Datenmodells existiert und alle Visualisierungen mit dieser Instanz arbeiten.

Die im Datenmodell gespeicherten Datensätze müssen für die Komponenten verfügbar sein, die mit den Daten arbeiten sollen. Die Daten werden über die Klasse DataManager zugänglich gemacht. Als erstes muss über den DataManager der DataElementContainer geholt werden. Aus diesem kann auf die entsprechend gewünschten Datensätze zugegriffen werden.

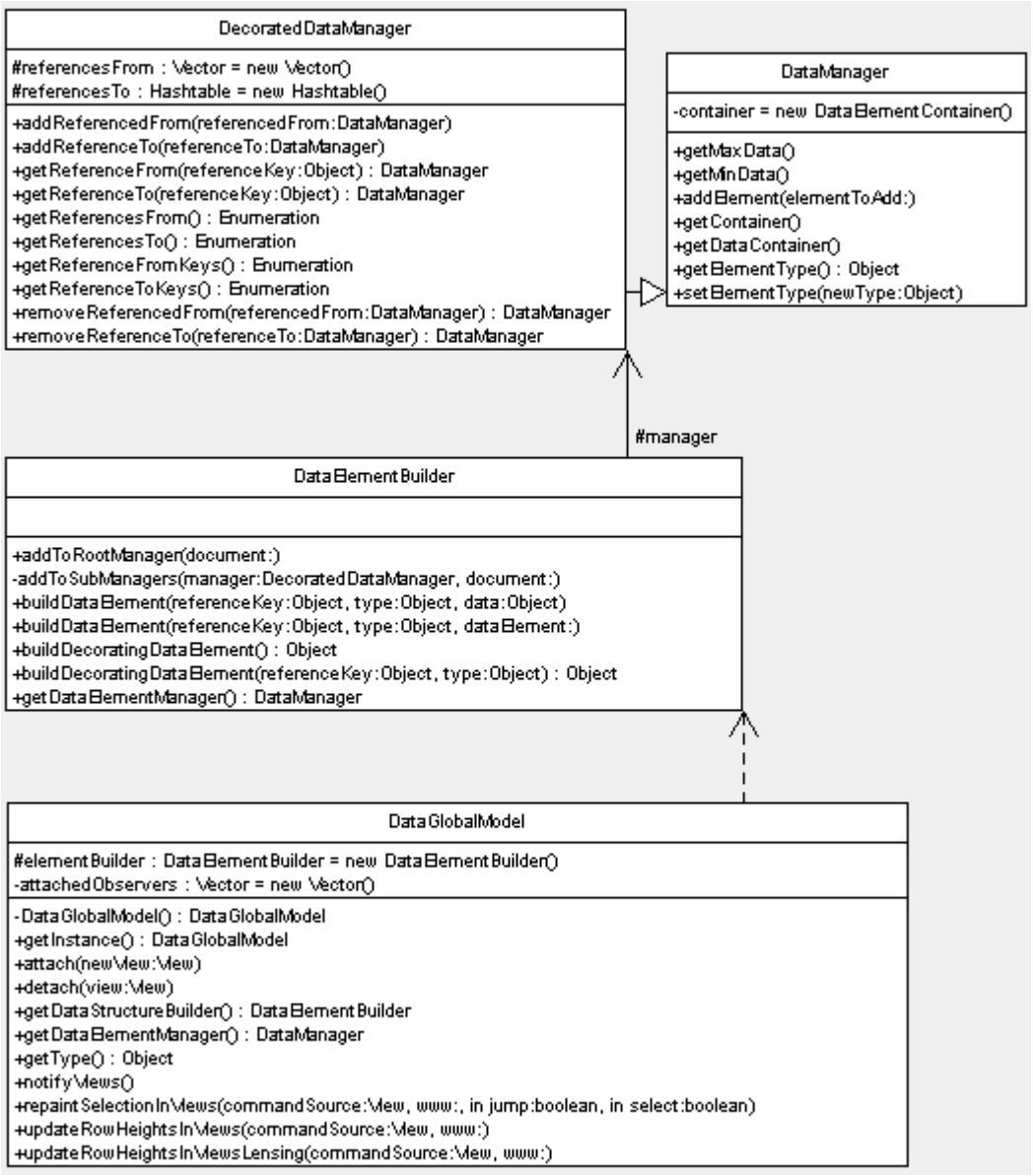

Abb. 4.26: UML-Auszug der Klassen des Packetes DataModel

Abbildung 4.26 zeigt einen Auszug aus dem Packet DataModel. Einige Klassen, wie zum Beispiel das Interface Model, sind herausgenommen, um die Übersicht zu behalten. Der Ablauf bei Start des Programms beginnt mit der Initialisierung der notwendigen Komponenten und der Visualisierungen. Vor dem Parsen der XML-Dateien wird eine neue Instanz der Klasse ParseManager erstellt. Sie bekommt eine Referenz des DataElementBuilder der Klasse DataGlobalModel. Die Hauptaufgaben beim Parsen werden von der Methode fillComplexElement(...) durchgeführt. Als erstes wird ein DataTypeIdentifier erstellt. Über diese Klasse können die Datentypen identifiziert werden. Die Klasse DataElementBuilder konstruiert in der Methode buildDecoratingDataElement(...) für jeden Datensatz ein neues WWWDocument. Danach liest der XMLParser ein Datenfile ein und speichert alle XML Werte in einer

Hashtable. Über den DataElementBuilder und die Hashtable werden die Datenobjekte dann zusammengebaut und im Datenmodell gespeichert. Die Speicherung erfolgt nach vollständiger Umwandlung der Werte des XML-Files mit der Methode addToRootManager(...). Diese Methode fügt das Datenobjekt dem Vektor des DataElementContainers hinzu. Damit befindet sich das Objekt an seinem Speicherplatz und Komponenten, die an das Datenmodell angekoppelt sind, können über den DataManager auf die Objekte zugreifen.

4.3.5 Visualisierung des Datenmodells

Das Datenmodell speichert die in den XML-Files enthalten Metadaten intern ab und stellt sie über verschiedene Schnittstellen zur Verfügung. So kann der Zugriff entweder über den DataManager oder die verschiedenen Submanager, die konstruiert werden können, ablaufen.

Die verschiedenen Visualisierungen müssen sich beim Start des Programms am Datenmodell anmelden. Hierzu stellt das Interface Model die Methode attach(...) zur Verfügung. Eine Visualisierung muss außerdem die Möglichkeit haben, sich wieder vom Datenmodell abzukoppeln. Dies ist zum Beispiel notwendig, wenn die Visualisierung beendet wird. Die Abmeldung der Visualisierung vom Datenmodell wird mit der detach(...)-Methode des Model-Interfaces erledigt. Bei Änderungen am Datenmodell werden die angemeldeten Visualisierungen vom Datenmodell informiert. Die Visualisierungen können dann entweder die neuen Daten vom Datenmodell abholen oder sie bekommen die Änderungen gleich vom Datenmodell übermittelt.

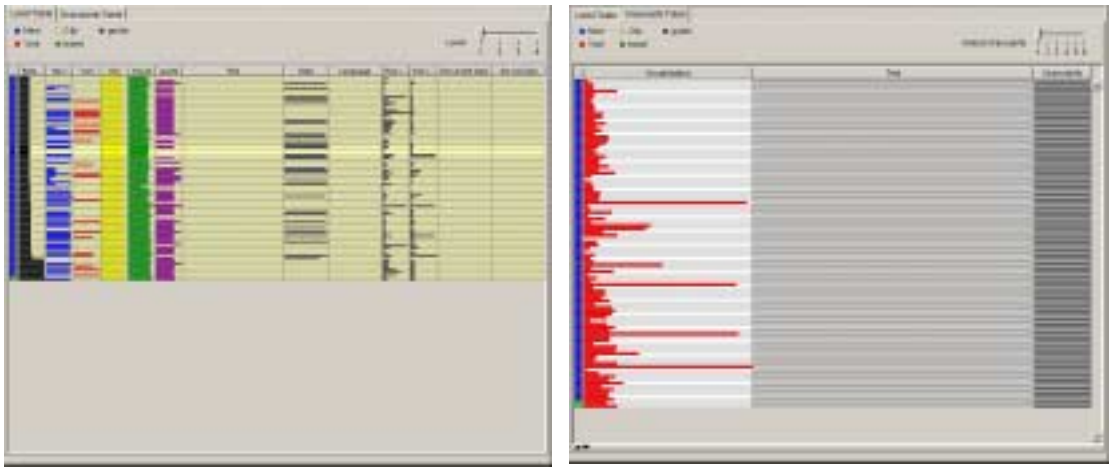

Abb. 4.27: Level Table (links) und Granularity Table (rechts)

Zum momentanen Zeitpunkt sind drei Visualisierungen an das Datenmodell angekoppelt. Die Level Table, die Granularity Table und der Scatterplot (Abb. 4.27 und 4.28).

Es ist möglich beliebig viele Visualisierungen an das Datenmodell anzukoppeln. Dies wird vor allem interessant, wenn das System ein verteiltes System ist. Dann kann das Datenmodell auf einem einzigen, leistungsfähigen Rechner arbeiten, während die Visualisierungen auf den Client-Rechnern laufen. Die Aktualisierung der Visualisierungen finden dann über ein Netzwerk statt.

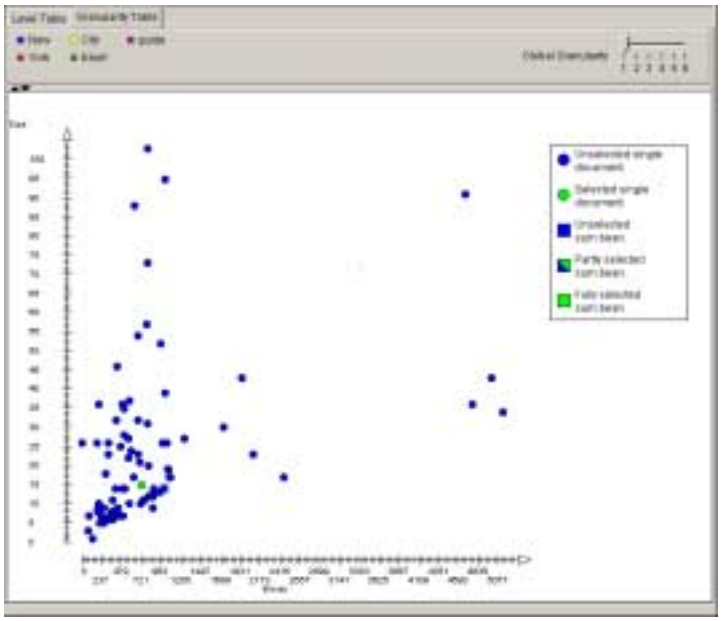

Abb. 4.28: Scatterplot Visualisierung

Andere Möglichkeiten der Datenvisualisierung werden zur Zeit am Lehrstuhl der Uni-versität Konstanz entwickelt. Es sind Erweiterungen in Form eines 3D Scatterplots und einer Circle-Segment-View in Arbeit.

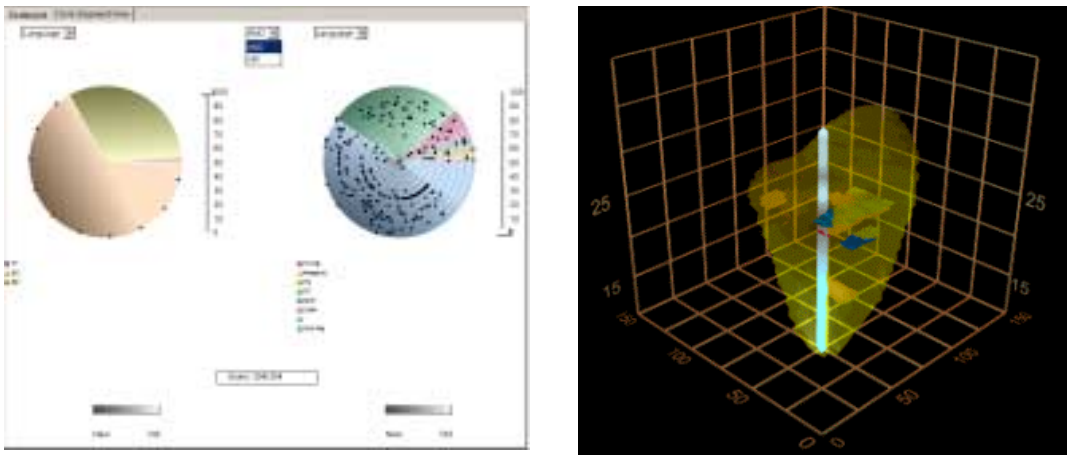

Abb. 4.29: Circle-Segment-View (Screenshot) und beispielhafter 3D Scatterplot entnommen aus [I8]

5 Implementation von Scatterplot und Magic Lens

Es würde den Rahmen dieser Arbeit sprengen, wenn der gesamte Quellcode an dieser Stelle erklärt würde. Stattdessen werden Vorgehensweisen und die grundlegenden Programmierkonzepte und Ideen, die bei diesem Projekt eingesetzt wurden, erklärt.

5.1 Beginn des Projektes und erste Implementation

Die Implementationsarbeit am Scatterplot begann mit einem ersten Projekt. In diesem wurde versucht die entwickelten Ideen straight-forward zu programmieren. Das Grundkonzept der Programmierung eines Scatterplots ist eigentlich relativ einfach. Es gibt eine x-Achse und eine y-Achse, die orthogonal ausgerichtet sind. Beide Dimensionen sind mit bestimmten Werten belegt. So können Punkte, entsprechend ihrer Werte, in den Scatterplot eingetragen bzw. eingezeichnet werden. Problematisch wird es allerdings schon bei der Aufteilung der Achsenwerte auf die zur Verfügung stehende Anzahl an Bildschirmpixeln. Wenn der Benutzer eine Achsenlänge von 1000 benutzen will, kann dieser Wert noch sehr gut auf 500 Bildschirmpunkte umgerechnet werden. Es fallen dann immer zwei Punkte des Benutzerwertes auf einen Punkt des Bildschirms. Sehr problematisch wird es allerdings bei ungeraden Zahlen, die nicht mehr ohne Rest berechnet werden können.

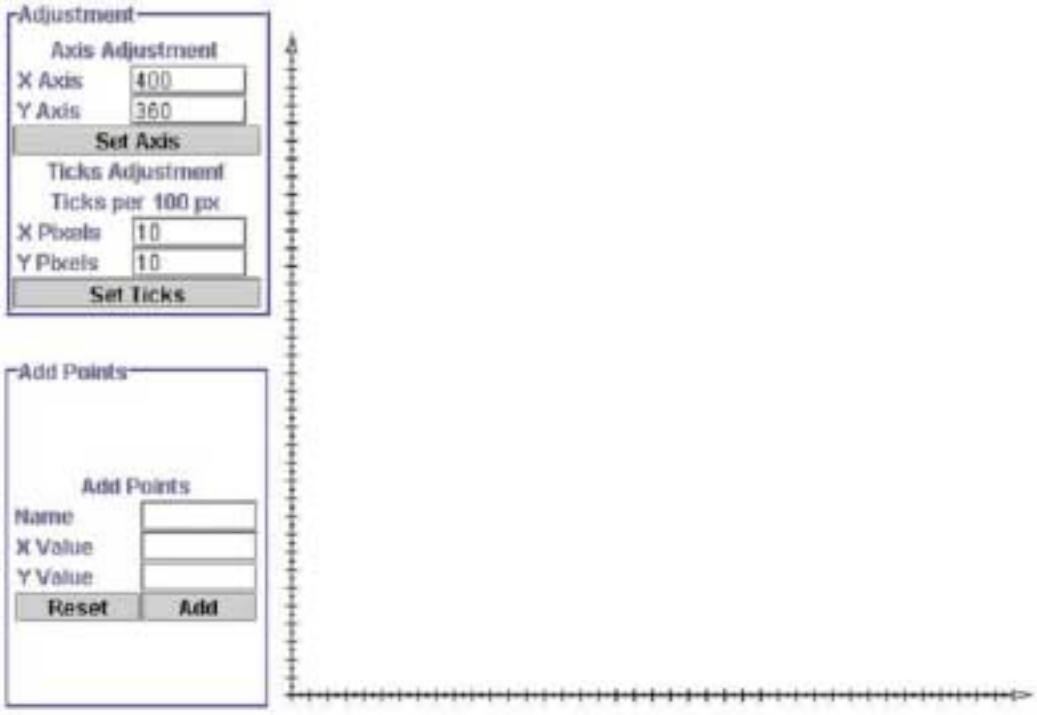

Abb. 5.1: erster Implementationsversuch eines Scatterplots

Wie bringt man beispielsweise eine Achsenlänge mit 700 Punkten auf 500 Bildschirmpixel? Diese Problematik entsteht, weil ein Bildschirmpunkt die kleinste, darzustellende Einheit auf dem Bildschirm ist und nicht mehr geteilt werden kann. Durch auftretende Probleme, die immer komplexer wurden, hat sich die straightforward Implementation als ungenügend herausgestellt. Mit der Zeit wurde das Programm sehr komplex und es konnte kaum mehr nachvollzogen werden, wo welche Operationen durchgeführt werden. Zu viele Funktionen haben Änderungen in zu vielen Bereichen des Programms notwendig gemacht. Dadurch kam es zu Unübersichtlichkeiten, die eine logische und zügige Programmierung unmöglich machten. Zu diesem Zeitpunkt wurde das Projekt abgebrochen.

5.2 Neuimplementation des Scatterplots

Eine vollständige, organisierte und strukturierte Neuimplementation wurde begonnen. Im ersten Schritt wurden Ideen und Konzepte zum logischen Aufbau des neuen Scatterplot entwickelt, um nicht die Fehler der Erstimplementation zu begehen. Die Neuentwicklung erfolgte nach dem MVC-Konzept. Die wichtigen Aufgaben und Operationen wurden jeweils auf die zuständigen Komponenten verteilt. So konnten auftretende Probleme direkt geortet und behoben werden. Bei der ersten Implementation wurden Klassenvariablen von allen möglichen anderen Klassen geändert und es konnte kaum nachvollzogen werden an welcher Stelle im Quellcode die Fehler waren. Der Programmaufbau unterstützt die Kommunikation der Komponenten mit Interfaces und abstrakten Klassen. Mit dem Java-Konzept der Events ist es sehr gut möglich, die Kommunikation zu managen und den Überblick zu bewahren.

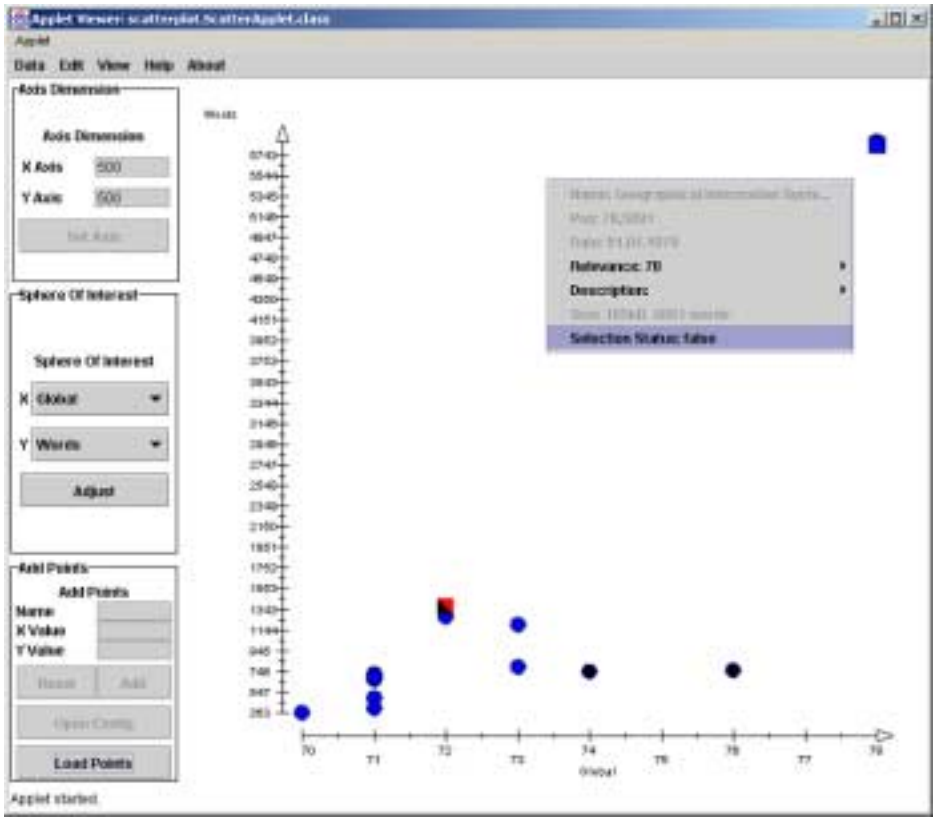

Abb. 5.2: OO-Implementation des Scatterplot als Java-Applet

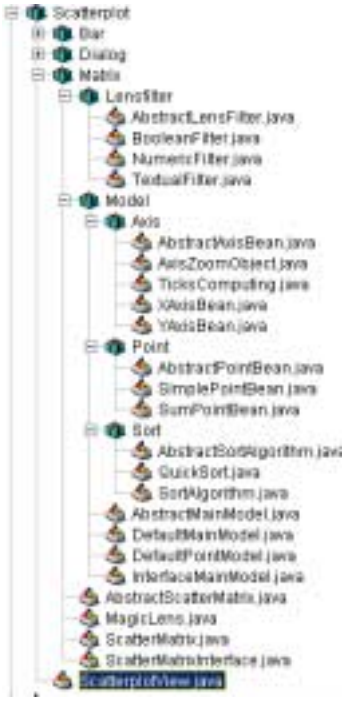

Abb. 5.3: Pakete d. Scatterplots (Screenshot JBuilder 6)

Die Abbildung 5.2 zeigt die fast komplett entwickelte Scatterplot-Komponente. Allerdings war das Programm zu diesem Zeitpunkt noch ein Java-Applet. Bei der Ein-

bindung in das INVISIP Framework wurden noch viele Verbesserungen und Änderungen gemacht. Das Applet musste in eine Applikation umgewandelt werden.
Bei der ersten Implementation waren alle Klassen in einem einzigen Paket. In der Neuimplementation wurden die verschiedenen Klassen in sinnvolle Pakete unterteilt, um eine bessere Übersicht zu erhalten. Die verschiedenen Komponenten des Programms registrieren sich gegenseitig über bestimmte Interfaces und können so zusammenarbeiten. Die Aufteilung der Pakete wird in Abbildung 5.3 gezeigt. Das Paket Scatterplot umfasst alle Klassen des Scatterplot. Im Paket Model sind die Klassen für das Datenmodel des Scatterplot enthalten. Zusätzlich existieren hier die Pakete für die Achsen, die Punkte und den notwendigen Sortieralgorithmus. Die abstrakten Klassen enthalten Quellcode, der sonst in jeder abgeleiteten Klasse nochmals stehen müsste. So wird über das Konzept der Vererbung die Codeverdopplung vermieden. Die Klassen der Pakete Axis und Point, sowie die Model-Klassen sind die Hauptkomponenten des Scatterplot. Die Java Swing Komponenten übernehmen alle Aufgaben, die ihre eigenen Werte und Eigenschaften betreffen. Wenn es notwendig wird von anderen Klassen aus Komponenten zu verändern, geschieht dies nicht direkt. Es werden sogenannte Events erzeugt, die an die zu verändernde Komponente geschickt werden. Diese reagiert dann entsprechend auf das Ereignis. Die Klassen SimplePointBean und SumPointBean sind zuständig für die Darstellung der Punkte im Scatterplot. Die Klasse SimplePointBean repräsentiert einen einfachen Punkt und SumPointBean einen Multipoint. Beide Klassen sind Java Beans, das heißt auch, dass sie von der Klasse Component abgeleitet sind. Damit übernehmen sie alle Grundeigenschaften einer Java Bean.

5.2.1 Komponentenhierarchie im Scatterplot

Die Komponenten zeichnen sich mit der Methode paintComponent(), die durch die Eigenschaften der Java Beans gegeben ist, selbst auf die Matrix des Scatterplot. Auch die beiden Achsen sind als eigenständige Komponenten realisiert und zeichnen sich selbst. Den Grund dafür lieferten die Achsenanpassungen und die vielen notwendigen Repaints. Immer wenn die Größe des Splitpanes oder des Hauptfensters der Anwendung durch den Benutzer verändert wird, muss ein Repaint durchgeführt werden. Außerdem auch jedes Mal wenn der Benutzer einen Zoom im Scatterplot macht oder die Achsenbelegungen ändert. Die Komponentenhierarchie, die beim Zeichnen auf den Bildschirm eingehalten wird, ist in Abbildung 4.15 zu sehen. Ganz

unten ist die ScatterplotView als größte Komponente eingezeichnet. Sie enthält alle anderen Komponenten.

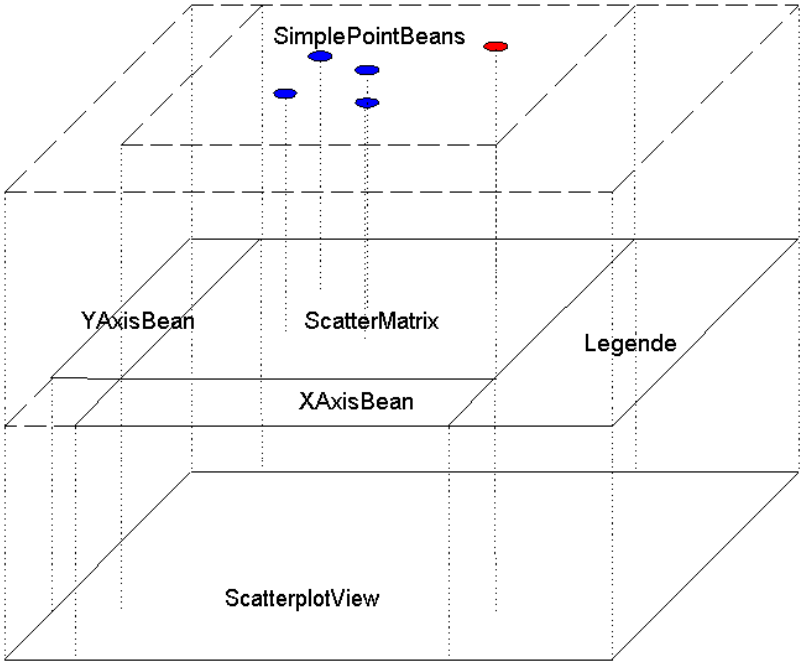

Abb. 5.4: Komponentenhierarchie Scatterplot

Die YAxisBean, die XAxisBean, die Legende und die ScatterMatrix werden bei der Initialisierung an ihren Positionen der ScatterplotView hinzugefügt. Zum Schluss werden die SimplePointBeans (einfache Punkte) und eventuell auch SumPointBeans (Multipoints) der ScatterMatrix hinzugefügt. Der Zeichenvorgang bei einem Repaint folgt genau dieser Hierarchie und wird vom Dispatch-Event-Thread übernommen. Dieser beginnt mit dem Zeichnen bei der ScatterplotView, geht über die Y-AxisBean, die XAxisBean, die Legende und die ScatterMatrix und endet beim Zeichnen der SimplePointBeans.

5.2.2 Aufbau der Struktur des Scatterplots

Um die Kommunikation der Scatterplot-Komponenten zu verstehen, ist die Abbildung 5.5 elementar. Sie zeigt den logischen Aufbau des Scatterplot nach dem Observer-Softwarepattern. Dieses Pattern wird auch MVC Pattern genannt.

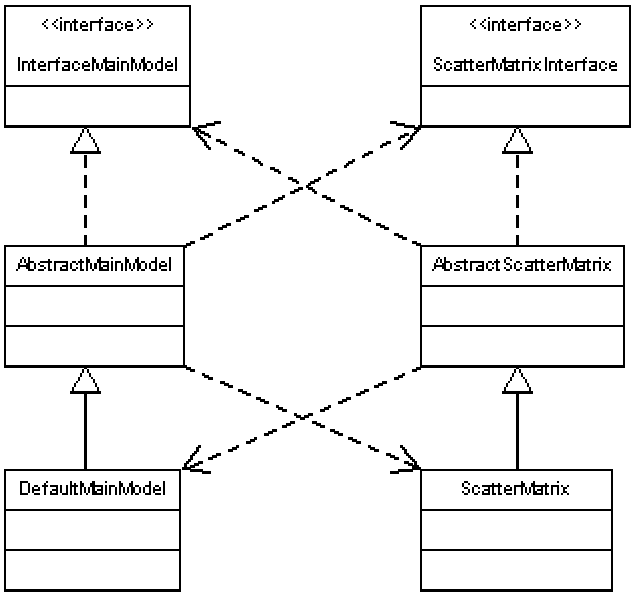

Abb. 5.5: UML des logischen Aufbaus des Scatterplot

Während das Model zur Datenspeicherung auf der linken Seite abgebildet ist, befindet sich die Matrix zur visuellen Darstellung der Daten auf der rechten Seite. Im DefaultMainModel werden alle Punktedaten gespeichert und immer wenn sich etwas an diesen Daten ändert, wird die Operation über das Model ausgeführt. Alle Komponenten, die Änderungen an den Daten vornehmen müssen, haben eine Klassenvariable des InterfaceMainModel. Diese muss einmalig mit einer Instanz des DefaultMainModel belegt werden. Dann können alle Methoden zur Veränderung der Daten über das Model benutzt werden. Jede Visualisierung der Daten muss Änderungen herbeiführen können. Abbildung 5.5 zeigt die Ankopplung der ScatterMatrix, eines Teils der Visualisierung, an das interne Datenmodell des Scatterplot. Die gesamte Visualisierung wird von der Klasse ScatterplotView im Paket Scatterplot übernommen. Sie benutzt die Klassen XaxisBean, YAxisBean und ScatterMatrix, um die Visualisierung des Scatterplot aufzubauen. Wie die Basisstruktur aus Abbildung 5.5 funktioniert, lässt sich am leichtesten durch Beispiele erklären. Die Kommunikation findet in zwei Richtungen statt. Einmal vom DefaultMainModel zur ScatterMatrix, wenn die Visualisierung angepasst werden muss. Die andere Richtung geht von der ScatterMatrix zum DefaultMainModel. Diese Kommunikation findet statt, wenn aufgrund einer Benutzeraktion Änderungen am Datenmodell bzw. an den Daten notwendig werden.

Die Kommunikation vom DefaultMainModel zur ScatterMatrix wird notwendig, wenn Daten im Datenmodell von einer anderen Komponente als der ScatterMatrix

selbst verändert wurden. Dann muss eine Aktualisierung der Datenvisualisierung stattfinden. Dazu stehen dem DefaultMainModel verschiedene Methoden zur Verfügung. Alle diese Methoden beginnen mit fire*Aktionsname*(). Dies ist eine Konvention bei der Implementation dieses Softwarepatterns. Soll also nun ein Repaint der ScatterMatrix vom DefaultMainModel aus ausgelöst werden, wird eine Methode mit dem Namen fireRepaintToMatrix() von der abstrakten Klasse AbstractMainModel zur Verfügung gestellt. Da das DefaultMainModel alle Eigenschaften dieser Klasse erbt, kann ohne weiteres diese Methode aufgerufen werden. Die Methode ist jedoch in der abstrakten Klasse implementiert. Die erleichtert die Unterklasse um den gesamten Quellcode der Kommunikation. Die weitere Kommunikation funktioniert über die Schnittstelle ScatterMatrixInterface. Die abstrakte Klasse AbstractMainModel benutzt eine Klassenvariable des Interfaces, die mit der Klasse ScatterMatrix initialisiert wurde. Das Interface ScatterMatrixInterface stellt die Methoden für Anweisungen an die ScatterMatrix zur Verfügung. Komponenten die Aktionen in der ScatterMatrix auslösen wollen, rufen die entsprechenden Methoden des Interfaces auf. Es wird in unserem Beispiel nun also die Methode repaint() aufgerufen. Da die abstrakte Klasse AbstractScatterMatrix das Interface ScatterMatrixInterface implementiert, müsste diese Klasse auch die im Interface vorgegebenen Methoden im Quelltext haben. Die Klasse ist jedoch abstrakt. Das bedeutet, sie muss nicht alle vorgegebenen Methoden implementieren. Stattdessen wird die Implementation der erbenden Klasse ScatterMatrix überlassen, die ja auch die Aktionen ausführen soll. Die Methode repaint() ist also in der AbstractScatterMatrix nicht zu finden, sondern wird in der ScatterMatrix implementiert. Der dortige Quelltext führt dann einen einfachen Repaint der eigenen Komponente aus. Damit wäre die vom DefaultMainModel aus eingeleitete Aktion erledigt.

Umgekehrt, also von der ScatterMatrix zum DefaultMainModel, funktioniert die Kommunikation auf die gleiche Art und Weise. Es werden nur die entsprechend anderen Komponenten benutzt. Die fire*Aktionsname*()-Methode der AbstractScatterMatrix wird aufgerufen. Diese ruft dann die entsprechende Methode des InterfaceMainModel auf. Dann führt die im DefaultMainModel implementierte Methode, die gewünschte Änderung an den Daten aus.

Im obigen Beispiel wurde gezeigt, wie eine Aktion von einer Komponente in einer anderen ausgelöst werden kann. Dies wurde am Beispiel eines Repaints demonstriert.

Es gibt jedoch auch Aktionen bei denen Daten oder andere Werte an die reagierende Komponente weitergegeben werden müssen. Um auch diese Aktionen ausführen zu können sind im Paket ComObjects die notwendigen Eventobjektklassen. Das Eventobjekt des Scatterplot heißt ChangeEventObject. Das Kommunikationsprinzip mit dem ChangeEventObject funktioniert relativ einfach. Wenn der Benutzer zum Beispiel die Achsenbelegung einer Achse ändert, wird diese zuerst aus der Visualisierung gelöscht. Dann wird eine neue Achse mit den neuen, korrekten Werten im DefaultMainModel konstruiert. Nun muss diese Achse wieder in die Visualisierung eingebaut werden. Um dies zu erreichen wird die Achsenkomponente einfach in ein Change-EventObject gepackt und über den normalen Ablauf an die Visualisierung mitgegeben. Die empfangende Klasse packt die Achsenkomponente wieder aus dem Change-EventObject aus und setzt sie in die Visualisierung ein. Das ChangeEventObject kann alle möglichen Arten von Datenobjekten in sich speichern und stellt auch Methoden zum Auspacken der Objekte zur Verfügung.
Es ist auch möglich die Konzepte von Java noch weiter zu nutzen. So kann dieses Modell noch um das Vetokonzept erweitert werden. Die Empfangskomponente kann hier ein Veto zur Durchführung der geforderten Aktion einlegen. Wenn eine Aktion bei einer Komponente ausgelöst werden soll, wird sie wie oben erklärt weitergereicht. Die empfangende Komponente kann dann jedoch mit einem Ergebnis antworten. Wenn die Aktion ausgeführt worden ist, ist das Ergebnis positiv. Falls die Aktion aus irgendeinem Grund nicht ausgeführt werden konnte, ist die Antwort negativ. Die auslösende Komponente hat dann die Möglichkeit je nach Antwort entsprechend zu reagieren. Dieses Konzept ist im Scatterplot nicht realisiert, da keine Notwendigkeit dafür gesehen wurde.

5.2.3 Die Achsenkomponenten

Ein komplexes Problem bei der Implementation war die Aufteilung der x- und y-Achsenwerte auf die zur Verfügung stehenden Bildschirmpunkte. Um nicht, wie bei der Erstimplementation, jedes Mal aufs neue alle Berechnungen ausführen zu müssen, hat sich eine Problemlösung bewährt, die auf zwei dynamischen Arrays basiert (eines für die x- und das andere für die y-Achse). Außerdem wird die Anpassung und die Berechnung der Achsen direkt in den Achsenklassen vorgenommen. Dies ist möglich, da diese nun eigene Komponenten sind. Das dynamische Array wird in der jeweiligen Achsenklasse (XAxisBean oder YAxisBean) in der Methode matchUser-

ValueToPixels() auf Basis des minimalen und des maximalen Punktwertes erzeugt und mit Werten gefüllt.

Durch den kleinsten und größten Wert kann die Länge der Achse bestimmt werden. Diese Länge entspricht aber noch nicht den passenden Pixelwerten des Bildschirms.

```
 1 private void matchUserValueToPixels()
 2 {
 3  double userLength = getUserMaxXAxisValue()-getUserMinXAxisValue();
 4  double relation = pixelLength/userLength;
 5  xMatchTable = new int[getUserMaxXAxisValue()+1];
 6  if(getUserMinXAxisValue() == 0)
 7  {
 8    for(int i=getUserMinXAxisValue(); i<=getUserMaxXAxisValue(); i++)
 9    {
10      xMatchTable[i] = Math.round((float)(i*relation));
11    }
12  }
13 }
```

Auszug 5.6: gekürzter Quellcodeauszug, Klasse XAxisBean, Methode matchUserValueToPixels()

Zur Umrechnung wird ein Array erzeugt, das für jeden Achsenwert eine Arrayposition bereithält. Wenn die Achse also insgesamt ein Länge von 1000 Werten hat, dann wird ein Array der Länge 1000 erzeugt. Die Länge des Arrays ist performancetechnisch unproblematisch, da immer nur zwei dieser dynamischen Arrays vorhanden sind. Außerdem müssen diese nur einmal, nämlich beim Füllen der Positionen, in Schleifen durchlaufen werden. Es finden also hauptsächlich Zugriffe auf die benötigten Pixelpositionen statt. Diese sind sehr schnell und damit ist die Arraylänge selbst bei 10.000 Werten unproblematisch. Damit jeder der Achsenpunkte einen Pixelwert zum Zeichnen auf dem Bildschirm zugeordnet bekommt, wird das Array mit den Pixelwerten für jeden Achsenpunkt gefüllt. Dazu wird als erstes eine Relation berechnet. Dies geschieht in Zeile vier des Quellcodeauszugs 5.6. Die vorhandene Pixellänge zum Zeichnen der Matrix-Komponente auf dem Bildschirm wird durch die Anzahl der x-Achsenwerte geteilt. Mit dieser Relation kann jetzt für jede Arrayposition der entsprechende Pixelwert berechnet werden. Besteht die x-Achse beispielsweise aus 700 Werten, hat also die Länge 700, dann müssen diese Werte alle auf eine festgelegte Pixellänge von z.B. 500 abgebildet werden. Das erzeugte Array hat für jeden Achsenwert eine Position und damit die Länge 700. Die Relation ergibt also 500/700. Jetzt kann für jede Arrayposition der Pixelwert mit dieser Relation bestimmt werden, indem die Zahl der Arrayposition mit der Relation multipliziert wird. Für den Punkt an der ersten Stelle ergibt sich also die Rechnung 1* (500/700). Durch

die notwendige Rundung, ergibt das genau eins. Für den Achsenwert 700 ergibt die Rechnung den Pixelwert 500. So werden also allen Achsenwerten die entsprechenden Pixelwerte zugeteilt. Durch die Dezimierung der 700 Werte auf 500 Pixel fallen einige Werte zusammen. Es ist also möglich, dass Punkte dann auf den gleichen Positionen gezeichnet werden, obwohl sie eigentlich nicht genau aufeinander liegen. Dieses Problem lässt sich leider nicht umgehen. Durch die Zuordnung von Pixelwerten zu Achsenwerten ist jedoch das Zeichnen der Punkte selbst kein Problem mehr. Jeder Punkt erhält von den Achsen seine genauen Pixelkoordinaten. Dies geschieht im DefaultMainModel in der Methode setPositionsForPoints(...). So können sich die Punkte immer an der richtigen Stelle selbst zeichnen.

Ein nicht triviales Problem ist auch das Zeichnen der Achsenaufteilung, den sogenannten Ticks. Es gibt zwei verschiedene Arten von Ticks. Die Major Ticks teilen die Achse in ihre Hauptabschnitte auf und sind beschriftet. Zusätzlich sind auf der Achse noch die Minor Ticks eingezeichnet. Sie teilen den Bereich zwischen den Major Ticks nochmals auf. Minor Ticks haben eine geringere Länge als die Major Ticks und sind nicht beschriftet. Um eine Achse mittels Ticks in möglichst sinnvolle Abschnitte zu unterteilen muss eine geeignete Anzahl an Major und Minor Ticks berechnet werden. Zur Lösung des Problems konnte an dieser Stelle der einzige Re-Use von Quellcode aus dem INSYDER-Programm erreicht werden. Dort waren die notwendigen Berechnungen in eine eigene Klasse ausgelagert. Diese konnte in den Quellcode des INVISIP-Scatterplot übernommen werden. Der Name der Klasse ist TicksComputing. Zum Re-Use der Klasse mussten jedoch Anpassungen im Quellcode gemacht werden. Dies kann nur geschehen, wenn die Funktion der Klasse verständlich ist. Viele Variablen und Methoden konnten gelöscht werden. Auch einige Abläufe wurden verändert. Nach der Anpassung konnte die Klasse aber übernommen werden, um die Ticks zu berechnen. Durch die Zuteilung von Pixelwerten zu jedem Achsenwert, gibt es auch beim Zeichnen der Ticks keine Probleme.

5.2.4 Datenmodell des Scatterplots

Der Kern des Scatterplot ist sein Datenmodell, das in der Klasse DefaultMainModel realisiert ist. Diese Klasse beinhaltet den größten Teil der Implementation und hat circa 1000 Zeilen Programmcode. Alle Änderungen der gespeicherten Daten funktioniert nur über die zur Verfügung gestellten Methoden dieses Datenmodells. Obwohl die Aufgaben des DefaultMainModel nicht sehr kompliziert sind, sind es doch sehr

viele. Die Daten werden in verschiedenen Vektoren gespeichert, die wieder in eigene Klassen ausgelagert sind. Einfache Punkte werden in einem Pointvektor gespeichert, während für Multipoints ein eigener Vektor existiert. Dann gibt es noch zwei weitere Vektoren, der eine ist für die Speicherung der Zoomhistory zuständig und der andere für die kreierten Magic Lenses. Das DefaultMainModel übernimmt alle notwendigen Aufgaben, die auf den Daten ausgeführt werden müssen. Im DefaultMainModel werden keine komplexeren Algorithmen benutzt. Es werden lediglich relativ einfache Aufgaben ausgeführt. Beispiele sind die Berechnung des minimalen und des maximalen eingelesenen Wertes, die Verteilung der Aufgaben bei einem Zoom und das Setzen der verschiedenen Punkteigenschaften bei Veränderungen durch die Visualisierung. Die einzigen komplexeren Operationen sind die Überprüfung aller Punkte auf MultiPoints und die Sortierung von Arrays und Vektoren über die Quicksort-Klassen. Die Überprüfung auf MultiPoints findet in der Methode checkForSummenBeans() in der Klasse DefaultMainModel statt. Es ist hier notwendig, drei For-Schleifen zu durchlaufen, von denen zwei zusätzlich geschachtelt sind, um das gewünschte Ergebnis zu erzielen. Für die Sortierung wurde der Quicksort in einem Sortier-Framework implementiert. Es können Zahlen, sowie auch Zeichenketten sortiert werden, da intern alle Vergleiche mittels der überladenen Methode compareTo() gemacht werden. Die Funktion des Quicksort soll hier nicht näher erklärt werden. Die Implementation basiert auf der C++ Programmierung in Quelle [27] und wurde in Java umgesetzt.

5.3 Integration in den INVISIP Framework

Die Integration gestaltete sich relativ einfach, da der Scatterplot sehr modular aufgebaut ist. Nach dem Umbau des Applets zur Applikation musste noch die Größe einiger Komponenten angepasst werden. Dann konnte der Scatterplot einfach in das vorgesehene Splitpane der Hauptanwendung eingebaut werden. Durch die Komponentenbasierte Programmierung aller Teile des Programms gab es keine schwerwiegenden Probleme bei der Integration. Da das Splitpane und damit auch der Scatterplot in der Höhe verstellbar sind, musste noch eine Anpassung der Höhe im Scatterplot programmiert werden. Durch die Grundidee, die bei der Anpassung der Achsenkomponenten verfolgt wurde, gestaltete sich diese Programmierung sehr einfach. Der Scatterplot wurde als eigenständige Komponente entwickelt und hat deshalb auch sein

eigenes Datenmodell. Daher musste bei der Anbindung an das Datenmodell des INVISIP-Frameworks das eigene Datenmodell des Scatterplots belassen und mit dem INVISIP Datenmodell verbunden werden. Das Scatterplot-Datenmodell wurde beibehalten, da das INVISIP-Datenmodell die wichtigen Operationen und Funktionen, die in der Visualisierung des Scatterplots benötigt werden, nicht zur Verfügung stellt. Für den Scatterplot spielt der interne Aufbau des INVISIP-Datenmodells also nur eine untergeordnete Rolle, da er selbst auf Basis des MVC-Konzepts aufgebaut ist. Die Datensätze aus dem DataGlobalModel werden nur ausgelesen, um sie im eigenen Scatterplot-Datenmodell zu speichern. Ein Umbau des Datenmodells und dessen Anpassung an die benötigten Funktionen würde mit Sicherheit helfen die Performance des Projektes zu verbessern.

5.4 Implementation des Magic Lens Filters

Die Implementation des Magic Lens Filters gestaltete sich wesentlich einfacher als die des Scatterplots. Zur Realisierung konnte eine einfache Swing-Komponente, das JPanel, verwendet werden. Die Komponente kann, wie jede visuelle Swing-Komponente, durchsichtig gemacht werden. Dazu muss nur ein Eigenschaftswert der Komponente verändert werden. Damit man den Magic Lens Filter jedoch nachher im Scatterplot sieht, erhält die Komponente einen nicht durchsichtigen Rand. Außerdem erfolgt in der Zeichenmethode paintComponent(...) eine Beschriftung der Linse mit den jeweiligen Filtereigenschaften. Der Linsenfilter ist mit der Hilfe von abstrakter Kopplung realisiert (s. Abb. 5.7). Die Linsenklasse erhält die konkrete Filterklasse erst zur Laufzeit des Programms. Sobald ein Magic Lens Filter von einem Benutzer erzeugt wird, wird bei der Konfiguration der entsprechende Filter der erzeugten Linse gesetzt. Durch die Komponentenhierarchie ist auch das Hinzufügen der Komponente zum Scatterplot kein Problem. Die Filterkomponente wird der Klasse ScatterMatrix hinzugefügt, da in dieser auch die Dokumentenpunkte liegen. Es können im Prinzip beliebig viele Magic Lens Filters erzeugt und hinzugefügt werden. Durch die Komponentenhierarchie löst sich auch das Problem der Linsenhierarchie bei mehreren Magic Lens Filtern von selbst. Die Linsen, die zuerst erzeugt wurden, liegen unten in der Hierarchie, die andere weiter oben. Liegt ein Magic Lens Filter über einigen Punkten, kommt die Filterfunktion zum Einsatz.

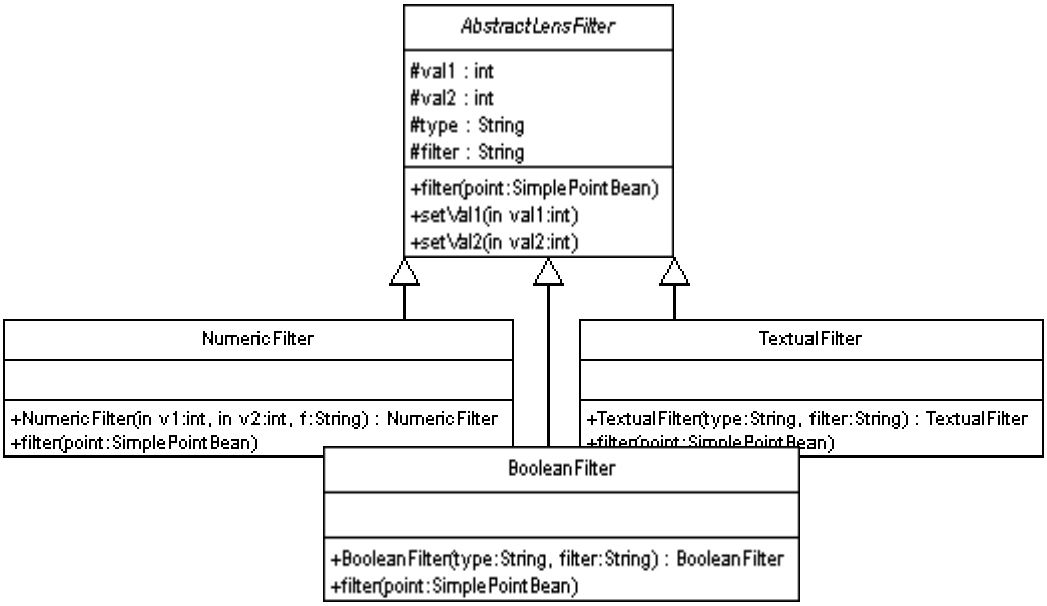

Abb. 5.7: UML der Magic Lens Filter

Es wird überprüft, welche Punkte innerhalb des Magic Lens Filters liegen. Der entsprechende Filter sorgt dann dafür, dass Punkte, deren Werte außerhalb des Filterbereichs liegen, nicht angezeigt werden. Das geschieht durch den Aufruf der Methode filter(...) mit dem zu filternden Punkt.

```
1  public void filter(SimplePointBean point)
2  {
3    o = point.get(filter);
4    if(!new String(""+o.booleanValue).equalsIgnoreCase(type))
5    {
6      point.setVisible(false);
7      point.setVisibleInLens(false);
8    }
9  }
```

Auszug 5.8: Filtermethode des bool'schen Filters

In Zeile drei des Auszugs 5.8 wird zunächst über eine Hilfsklasse QueryObject (Variable o) der benötigte Wert des Punktes geholt. Danach findet der Vergleich des Wertes mit dem Wert des Filters (Variable type) statt. Wird der Punkt durch den Filter herausgenommen, wird die Sichtbarkeitseigenschaft der Punktkomponente auf unsichtbar gesetzt.

Die Veränderung der Daten erfolgt über das Datenmodell, an dem die Linse selbst angekoppelt ist. Auch das Einfügen und Herausnehmen von Linsen, wird vom Datenmodell gemanagt. Liegen mehrere Linsen über den Punkten, filtern die Linsen die Punkte automatisch gemäß ihrer Hierarchie aus.

5.5 Interaktion zwischen Granularity Table und Scatterplot

Da verschiedene Visualisierungen mit dem gleichen Datenmodell und damit auch mit den gleichen Datensätzen arbeiten, musste eine geeignete Kommunikation für die Komponenten gefunden werden. Da auch im Datenmodell des INVISIP Projektes das Observer-Pattern angewendet wurde, gestaltet sich die Implementation jeglicher Interaktion zwischen den Visualisierungen relativ einfach. Der Scatterplot und die Granularity Table sind beide als Visualisierungen am INVISIP-Hauptdatenmodell angemeldet. Dadurch werden auch beide Visualisierungen über notwendige Aktualisierungen informiert. Dies ist durch die Implementierung der vom Datenmodell zur Verfügung gestellten Interfaces möglich. Wird in einer Visualisierung eine Änderung durch den Benutzer gemacht, schickt diese eine Nachricht mit den Änderungen an das Datenmodell. Dieses macht die Änderungen auf den Daten und schickt die Aktualisierungsinformationen an alle angemeldeten Visualisierungen. Diese können nun sehr einfach auf die Änderungen reagieren.

Die einfachste Interaktion zwischen der Granularity Table und dem Scatterplot, entsteht bei der Selektion von Dokumenten. Wird ein Dokument in einer der beiden Visualisierungen selektiert, informiert diese das Datenmodell. Dadurch wird dann auch die andere Komponente aktualisiert. Dieser Interaktionsmechanismus funktioniert in beide Richtungen bei Selektion und Deselektion von Dokumentenpunkten. Zusätzlich zur Selektion, rückt ein Dokument, das im Scatterplot selektiert wird, in der Granularity Table in den Fokus. Die schwarze Linie in Abbildung 5.9 wurde mit einem Bildbearbeitungsprogramm hinzugefügt, um die beiden Visualisierungen des einen Dokuments zu verdeutlichen. Wird der untere Punkt im Scatterplot selektiert, rückt der obere Punkt in der Granularity Table in den Fokus und ist auch selektiert.

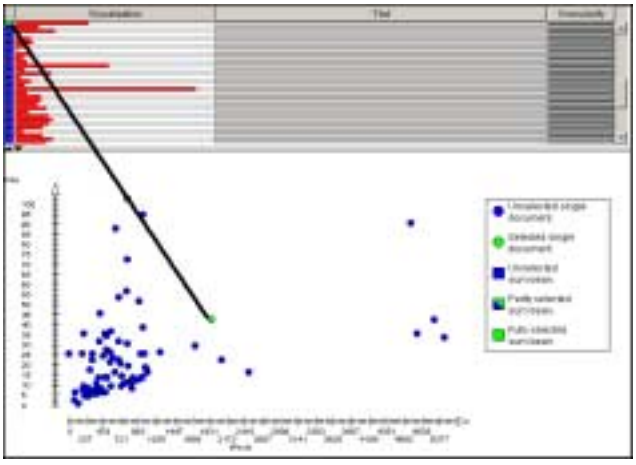

Abb. 5.9: Interaktion per Selektion zwischen Scatterplot und Granularity Table

Die zweite Interaktionmethode funktioniert über das Kontextmenu der Dokumentenpunkte im Scatterplot. Dort ist ein Slider verfügbar mit dem die Granularitätsstufen der Dokumente in der Granularity Table vom Scatterplot aus eingestellt werden können. Im linken Bild der Abbildung 5.10 sieht man, wie der Slider des selektierten grünen Punktes im Scatterplot auf Stufe drei gestellt wurde. Dadurch befindet sich die Tabellenzeile des Dokuments in der Granularity Table auch in Stufe drei.

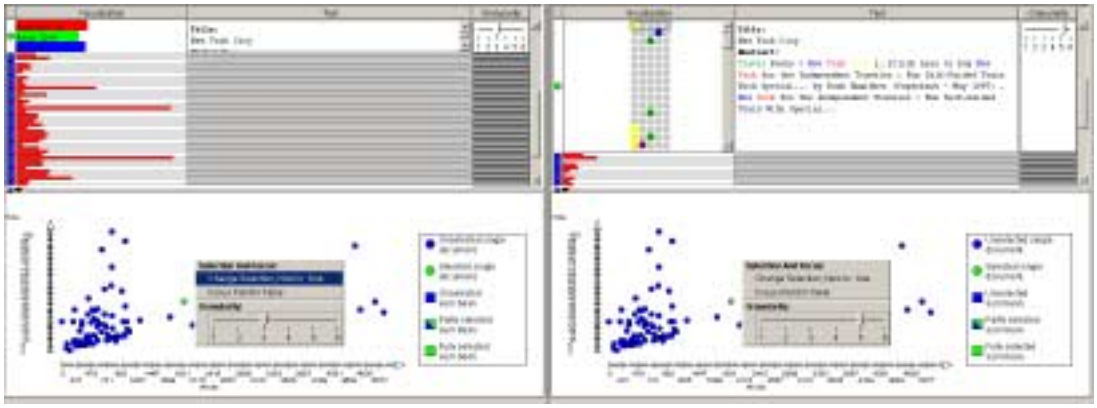

Abb. 5.10: Interaktion per Slider im Kontextmenu

Der dritte Interaktionsmechanismus tritt beim Einsatz eines Magic Lens Filters in Kraft. Alle Dokumente, die durch den Filter nicht mehr sichtbar sind, werden auch in der Granularity Table verändert dargestellt. Eigentlich sollten die Dokumentzeilen ganz aus der Tabelle verschwinden. Dies ist jedoch nicht möglich, da die Dokumente aus Performancegründen nicht aus dem Datenmodell genommen werden können. Würden sie dennoch herausgenommen, müssten sie, sobald der Magic Lens Filter weiterbewegt wird, wieder aus dem Datenmodell geladen und in die Tabelle eingefügt werden. Eine flüssige Bewegung der Linse über den Punkten des Scatterplots wäre damit nur bei einer sehr geringen Anzahl an Punkten möglich. Aus diesem Grund wurde eine Alternative programmiert. Die Höhe der auszublendenden Zeilen wird einfach auf einen Pixel gesetzt. Diese Lösung ist leider auch nicht optimal, da die Zeilen nicht ganz verschwinden. Die Höhe einer Tabellenzeile kann aber in Java aus unbekannten Gründen nicht auf null gesetzt werden.

Die letzte verfügbare Interaktion zwischen der Granularity Table und dem Scatterplot wurde beim Einsatz des Zoomings im Scatterplot programmiert. Auch diese Interaktion geht vom Scatterplot aus und beeinflusst die Granularity Table. Im Prinzip passiert genau das Gleiche wie beim Einsatz eines Magic Lens Filters. Die Do-

kumente, die aus dem visuellen Bereich des Scatterplots fallen, werden nur noch mit einer Zeilenhöhe von einem Pixel dargestellt.

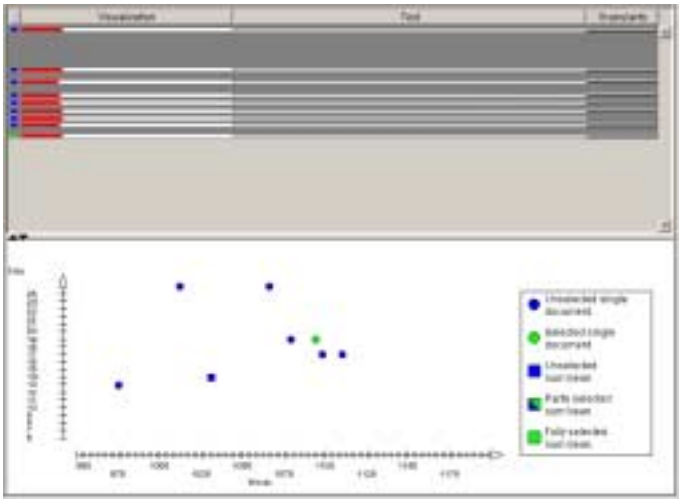

Abb. 5.11: Interaktion per Slider im Kontextmenu

Die grauen Balken in der Tabelle entstehen durch viele untereinander liegende Zeilen mit einer Zeilenhöhe von einem Pixel.

Die Implementation der Kommunikation ist durch die Interfaces sehr einfach. Da alle Visualisierungen das View Interface (Auszug 5.12) implementieren müssen, müssen sie die eingetragenen Methoden enthalten. In diesen Methoden wird die Reaktion der jeweiligen Visualisierung festgelegt.

```
public interface View
{
  public abstract void update(DataManager elementManager);
  public abstract void repaintSelectionInView(WWWDocument www, boo
lean jump, boolean select);
  public abstract void updateRowHeightInView(WWWDocument www);
  public abstract void updateRowHeightInViewLensing(WWWDocument
www);
}
```

Auszug 5.12: Das Interface View

Bei Änderungen im Datenmodell werden dann die implementierten Methoden jeder angemeldeten Visualisierung aufgerufen. Kommen neue Interaktionsfunktionen hinzu, genügt es, eine neue Methode im Interface einzutragen. Über diese wird dann die notwendige Kommunikation sichergestellt. Alle anderen Visualisierungen sind nun gezwungen, die neue Methode auch zur Verfügung zu stellen und zu implementieren. In der Implementation kann jetzt jede Visualisierung eine andere Weise der Reaktion festlegen. Die Projektgruppe INVISIP an der Universität Konstanz entwickelt

momentan noch weitere Ideen zur Interaktion zwischen der Granularity Table und dem Scatterplot. Die Entwicklung der Interaktionen ist jedoch kompliziert, da sehr gut überlegt werden muss, ob Benutzer mit den ausgedachten Konzepten zurechtkommen. Später, wenn noch andere Visualisierungen hinzukommen, entstehen noch viel mehr Möglichkeiten zur Interaktion zwischen den einzelnen Sichten. Im Gegensatz zur einfachen Implementation ist die Theorie dieser Interaktionen sehr kompliziert. Durch die Unterschiedlichkeit der Datendarstellung treten sehr viele Probleme auf. Außerdem gibt es auf diesem Gebiet noch nicht sehr viele Erfahrungen.

6 Ergebnisse

Im folgenden sollen die Erfahrungen und Ergebnisse dieser Arbeit kurz zusammengefasst werden. Durch die intensive Beschäftigung mit dem Projekt und dessen Implementierung sammelte das Programmierteam sehr viele Erfahrungen im Umgang mit der Programmiersprache Java und der Objektorientierung. Bei der Implementation traten sehr viele unterschiedliche Probleme in verschiedenen Gebieten auf. Die Lösung war oft mit dem Nachschlagen in der Java API und der Suche nach ähnlichen aufgetretenen Problemen in Internetforen und im WWW verbunden. Alle beim Projekt aufgetretenen lösbaren Probleme konnten schließlich gelöst werden und es entstand der in dieser Arbeit vorgestellte Scatterplot.

Durch die zuerst unterschätzte Komplexität des Projektes, wurden zu Beginn einige Fehler begangen. Ein Fehler war der Start der Implementation ohne vorherige Planung. Dieser erste Versuch der Programmierung schlug dann auch fehl und das Projekt musste von neuem implementiert werden. Es ist also wichtig, die Komplexität eines Projektes schon vor dessen Beginn einschätzen zu können, um den Umfang der notwendigen Planungs- und Konzeptionsphase festzulegen. Werden größere Projekte ohne Planungsphase begonnen, besteht die Möglichkeit, dass das Projekt irgendwann aus Unübersichtlichkeiten, Unklarheiten, Konzeptionproblemen oder sonstigem frühzeitig abgebrochen werden muss. In der freien Wirtschaft kommt oft erschwerend noch ein enormer Zeitdruck und ein begrenztes Projektbudget hinzu. Komplexen Programmierprojekten sollte eine relativ lange Planungs- und Konzeptionsphase vorausgehen. Die Implementationsphase benötigt dann viel weniger Zeit und viele Probleme sind schon vorher bekannt.

Ein weiterer Fehler war, der Beginn der Implementation des Scatterplots ohne die Konzepte der Programmiersprache Java zu beachten. Die anfänglich prozedurale Programmierung war unübersichtlich und nach einer gewissen Zeit, als das Projekt schon sehr umfangreich war, kaum mehr zu durchschauen. Erst nach Abbruch des ersten Versuchs wurde die Objektorientierung und der mögliche Einsatz von Software-Patterns bedacht. Durch deren Einsatz konnte die Übersichtlichkeit gewahrt bleiben. Außerdem brachten die Konzepte und Patterns noch andere Vorteile, wie die Unabhängigkeit von Software-Komponenten, mit sich. In der Planungsphase eines Projektes sollte sehr gut darüber nachgedacht werden, wie die Struktur und der Aufbau der zu entwickelnden Software aussehen soll. Es muss geplant werden an wel-

chen Stellen des Programms die Software-Patterns benutzt werden können. Auch die Schnittstellen, die eine nachträgliche Anpassung des Programms gewährleisten, müssen vor der Implementation definiert werden.

Wichtige Erkenntnisse dieser Arbeit wurden auch bei der Implementation der Benutzeroberfläche gesammelt. Die Komponentenhierarchie, das Java Beans Konzept und das Zeichnen der Komponenten mussten sorgfältig bedacht werden. Weiterhin ist die Verteilung der Aufgaben auf die verschiedenen Komponenten elementar. Der Quellcode einer Software kann leichter verstanden werden, wenn jede Komponente gekapselt ist und ihre eigenen Aufgaben übernimmt. Werden dagegen Komponenten im Quelltext von sehr vielen Stellen aus verändert, fördert dies die Unübersichtlichkeit des Programms und die Software kann von anderen Programmierern nur mit übermäßigem Zeitaufwand verstanden werden. Problematisch ist auch die Ansicht des Quellcodes, da Programmierer oft verschiedene Arten der Quelltextformatierung bevorzugen. Arbeitet ein Team ohne gewisse Formatierungsstandards zusammen, entstehen zeitliche Verluste durch unnötige Neuformatierungen des Quelltextes.

Eine weitere Erkenntnis wurde durch den immensen Speicherbedarf der Gesamtanwendung INVISIP gewonnen. Die JVM, die Laufzeitumgebung für Javaprogramme, hat standardmäßig nur wenige Megabytes an Speicher zur Verfügung. Beim Einlesen von sehr vielen Testdaten, wie das beim INVISIP Prototyp mit 600 Datensätzen der Fall ist, reicht irgendwann der Speicher der JVM nicht mehr aus. Dann kommt es zum Absturz des gesamten Programms. Dieses Problem kann behoben werden, indem der JVM beim Aufruf mehr Speicher zugewiesen wird. Der JVM werden die Parameter ‚-ms32m -mx200m' in einer Befehlszeile mitgegeben. Der erste Wert 32 steht für den minimalen Speicher und der zweite Wert 200 für den maximalen Speicher. Beide Werte geben den Speicher in Megabyte an und können je nach vorhandener Hardware beliebig verändert werden. Durch diesen großen Bedarf an Speicher leidet allerdings die Performanz des gesamten Programms.

6.1 Probleme und Verbesserungsmöglichkeiten

Obwohl der Scatterplot eine gut funktionierende Visualisierung ist, können noch einige Verbesserungen durchgeführt werden. Ein nach wie vor offenes Problem ist die Berechnung der Minor und Major Ticks. Der Algorithmus zur Berechnung konnte zwar aus dem alten INSYDER-Scatterplot übernommen werden, er funktioniert

jedoch nicht in jedem Fall optimal. Hier wäre eine bessere Lösung für dieses mathematische Problem von Vorteil. Ein weiterer Punkt ist die Anbindung des Scatterplot an das INVISIP-Datenmodell. Zur korrekten Darstellung im Scatterplot sind meistens Operationen zur Berechnung von Werten notwendig, die vom Datenmodell nicht unterstützt werden. Deshalb besitzt der Scatterplot zusätzlich sein eigenes Datenmodell. Würden vom INVISIP-Datenmodell die notwendigen Methoden zur Verfügung gestellt, könnte auf das Datenmodell des Scatterplot verzichtet werden. Daraus würde ein geringerer Speicherbedarf und die direkte Anbindung der Visualisierung an das INVISIP-Datenmodell resultieren.
Die Funktion des INVISIP-Datenmodells als Subject im Observer-Pattern ist ein sehr gutes Programmierkonzept. Problematisch sind jedoch die vorhandenen Zugriffsarten auf die gespeicherten Daten. Viele eigentlich sinnvolle und notwendige Metadaten werden vom Datenmodell nicht angeboten. Außerdem sind die Methoden zum Zugriff zu komplex gehalten. Ein Beispiel für einen Datenzugriff sieht im Quelltext des Scatterplot wie in Auszug 6.1 aus. Zusätzlich sind für den eigentlichen Datenzugriff noch weitere Operationen notwendig. Eigentlich würde eine einfache Methode genügen, die einen Index des gewünschten Dokuments als Parameter erhält. Ein Datenmodell sollte einfachere Möglichkeiten anbieten, um auf die gewünschten Daten zuzugreifen.

```
WWWDocument www =
(WWWDocument) manager.getContainer().getElementAt(rowIndex);
```

Auszug 6.1: Quellcode für den Zugriff auf das Datenmodell

Das Hauptproblem des Projektes ist die schlechte Performance und der sehr hohe Speicherbedarf. Beides entsteht durch die Visualisierung einer großen Anzahl von Daten. Bei wenigen Dokumenten ist die Performance und der Speicherbedarf unproblematisch. Die gesamte Anwendung funktioniert dann flüssig und die JVM benötigt keinen zusätzlichen Speicher. Erst ab ca. 100 darzustellenden Dokumenten reagiert die Anwendung träge und braucht sehr viel Speicher. Nach einer Verbesserung des Datenmodells müsste der Scatterplot kein zusätzliches eigenes Datenmodell verwenden. Momentan benötigt das Datenmodell bei jedem Zugriff auf einen Datensatz Zeichenkettenvergleiche, obwohl einfache Zahlenvergleiche eigentlich ausreichen würden. Bei einer großen Anzahl an Datenzugriffen müssen hier sicherlich Geschwindigkeitsverluste hingenommen werden. Durch einen Umbau des Datenmo-

dells könnte hier die Performance der Anwendung verbessert werden. Der Scatterplot für sich läuft auch bei einer großen Anzahl von Daten relativ flüssig. Problematischer ist die Tabellenansicht. Dort müssen in jeder Zeile nicht nur die einzelnen Punkte visualisiert werden, sondern auch der Text des Dokuments, eine Visualisierungskomponente und ein Slider. Diese Komponenten benötigen viel Speicher und verlangsamen dadurch die Anwendung. Die Darstellung von farblich markiertem Text ist dabei der Hauptgrund für längere Wartezeiten, die der Benutzer hinnehmen muss.

Aber auch der Scatterplot lässt noch einige Verbesserungsmöglichkeiten offen. Dazu gehört der Magic Lens Filter Dialog. Er muss noch benutzerfreundlicher gestaltet werden. Ein weiteres Problem ist, dass beim Einsatz eines Magic Lens Filters die in der Linse liegenden Punkte nicht mehr vom Benutzer angewählt werden können. Dies liegt daran, dass die Linse eine eigene Komponente ist und über den Punkten liegt. Damit werden die Mausklicks des Benutzers schon von der Linsenkomponente verarbeitet und die Mouse-Listener der Punkte werden nicht aktiv. Dieses Problem könnte zum Beispiel dadurch gelöst werden, dass eine eigene Komponente für die Linse programmiert wird, die dann nicht sich selbst, sondern nur ein Quadrat aus vier Linien in den Scatterplot zeichnet. Dadurch könnten die Mouse-Events von den Punkten empfangen werden und die Mausklicks funktionieren wieder. Es stellt sich dann aber die Frage, wie die Linse bewegt werden soll, da dies auch mittels Mouse-Events realisiert werden soll. Eine Lösung könnte eine kleine Komponente im Quadrat des Magic Lens Filters sein, die angeklickt werden muss, um den Filter zu bewegen. Ein weiteres offenes Problem ist neben der Berechnung der Ticks auch die Beschriftung der Achsen. In seltenen Fällen treten Datenbereiche auf, bei denen sich die Beschriftungen überschneiden.

Ansonsten traten beim Test der Anwendung keine Schwierigkeiten auf. Bei einem ausführlichen Test kann es jedoch sein, dass noch die ein oder anderen kleinen Fehler in der Implementation auftauchen. Dies ist bei umfangreichen Projekten (wie INVISIP), an denen mehrere Programmierer arbeiten, die Regel und es müssen nachträglich noch Änderungen im Quellcode gemacht werden. Da INVISP aber noch in der Entwicklung ist, können noch ohne weiteres Anpassungen vorgenommen werden.

6.2 Vorläufige Evaluation

Ein erster INVISIP Prototyp wurde im Juni 2002 von drei Experten evaluiert. Der Entwicklungsstand dieses Prototypen ist nicht vergleichbar mit dem momentanen Stand des Projektes. Dennoch zeigt die vorläufige Evaluation die schon vermuteten Schwachstellen deutlich auf. Die Experten überprüften den Prototypen auf der Basis einer Checkliste von Xerox aus dem Jahre 1995 [I9]. Es wurden nicht alle Gebiete, auf die sich der Fragebogen bezieht, untersucht, da einige beim INVISIP Projekt keine Anwendung finden konnten. Die Themengebiete, die benutzt wurden sind Sichtbarkeit des Systemzustands (Visibility of System Status), Konsistenz und Standards (Consistency and Standards), Erkennbarkeit statt erneutem Aufruf (Recognition rather than Recall), Flexibilität und minimalistisches Design (Flexibility and Minimalist Design), Ästhetik und minimalistisches Design (Aesthetic and Minimalist Design), sowie zufriedenstellende und respektvolle Interaktion mit dem Benutzer (Pleasurable and Respectful Interaction with the User). Das Gesamtergebnis der identifizierten Usability Probleme zeigt Abbildung 6.2.

Gesamtergebnis der Evaluation				
	Experte A	Experte B	Experte C	gesamt:
Catastrophe	2	8	0	10
Major	4	5	7	16
Minor	23	14	8	26
	29	27	15	

Abb. 6.2: Gesamtergebnis der Evaluation

Insgesamt wurden von Experte A 29 Probleme identifiziert, von Experte B 27 und von Experte C 15. Die sehr niedrige Zahl des Experten C resultiert unter anderem daraus, dass für diesen keine Ergebnisse zu den letzten beiden Gebieten des Fragebogens vorlagen. Zu den letzten drei Gebieten wurden nur wenige Usability Probleme der Kategorie Minor, also der Stufe mit dem niedrigsten Grad, identifiziert. Ein Problem der Kategorie Catastrophe kam im letzten Gebiet vor. Dieses betraf allerdings die Tabellenvisualisierung und nicht den Scatterplot.

Den Scatterplot betreffend wurde herausgefunden, dass Benutzer sich schwer tun diesen überhaupt zu finden. Beim Start der Anwendung wird nur die Tabellenvisualisierung angezeigt und das Splitpane des Scatterplots ist ganz unten ‚versteckt'. Der Benutzer muss den horizontalen Balken des Splitpane nach oben ziehen, um den

Scatterplot anzuzeigen. Eine Lösung für dieses Problem wäre eine Anzeige beider Komponenten schon beim Systemstart. Weiterhin wurde die Darstellung der Dokumente als runde Punkte kritisiert. Die Anmerkung war, sie würden an Radio Buttons erinnern aber eine mehrfache Selektion zulassen. In einem Scatterplot werden aber immer Punkte als visuelle Repräsentation verwendet. Dieses Problem kann hier nicht nachvollzogen werden, zumal in anderen Arbeiten, wie Envision [14] oder dem Filmfinder [15], diese Probleme auch nicht erwähnt wurden. Die letzten Kritikpunkte zum Scatterplot betreffen das Tabbed Pane zur Konfiguration, den Konfigurationsdialog des Magic Lens Filters und die Achsenbeschriftung des Scatterplots. Da der Dialog zum Magic Lens Filter nur zum Test dienen sollte und noch nicht ausgearbeitet ist, entspricht er nicht den Konventionen. Dies wurde in der Evaluation bestätigt. Auch das Tabbed Pane wurde noch nicht nach Usability Gesichtspunkten gestaltet, da daran immer wieder Veränderungen während der Implementation vorzunehmen waren. Nach der Überarbeitung dieser beiden Komponenten könnte eine erneute und dann auch sinnvollere Evaluation stattfinden. In der Evaluation wurde auch festgestellt, dass sich bei einigen Wertebereichen der Achsen des Scatterplots die Beschriftungen überschneiden. Deshalb können dann die Zahlen nicht oder kaum gelesen werden. Der Grund dafür liegt im Algorithmus zur Berechnung der Ticks. Dieser sollte so verbessert werden, dass die Aufteilungen bei allen Wertebereichen korrekt berechnet werden. Trotzdem wurden in der Evaluation einige Verbesserungsvorschläge gemacht, die bei zukünftigen Implementationen helfen werden.

6.3 Ausblick

Das Ende dieser Arbeit bildet ein kurzer Ausblick. Ideen und Möglichkeiten, die in Zukunft noch in das Projekt einfließen könnten, sollen hier erläutert werden.

In der Evaluation [13] des alten INSYDER-Scatterplots wurde festgestellt, dass die Benutzer des Scatterplots eine Anzeige der Gesamtrelevanz der Dokumente vermissen. Dies ist natürlich nur der Fall, wenn die Gesamtrelevanz nicht auf einer der beiden Achsen angezeigt wird. Zwei Ideen zur Lösung des Problems sollen hier kurz erklärt werden. Einmal könnten die Dokumentenpunkte zylinderförmig dargestellt werden. Die Höhe des Zylinder wäre dann von der jeweiligen Dokumentrelevanz abhängig. Das Problem dabei ist, dass die Übersichtlichkeit bei vielen Dokumenten sehr stark abnimmt. Ein weiteres Problem entsteht bei den sogenannten Multipoints.

Diese entstehen, wenn mehrere Dokumente an einer Stelle liegen. Offen ist, wie diese Multipoints zylinderförmig dargestellt werden sollen. Eine weitere Idee zur Visualisierung der Gesamtrelevanz wird auch in Envision [14] verwendet. Die Relevanz ist dort in drei Farbabschnitte unterteilt. Sehr relevante Dokumente werden rot dargestellt, die anderen gelb und grau. Diese Lösung würde die Gesamtrelevanz der Dokumente wenigstens in drei Gruppen visualisieren. Ein Konflikt entsteht aber durch die notwendige Konsistenz mit den anderen Visualisierungen, die es bei Envision nicht gibt. Auch selektierte Dokumente müssten dann auf eine andere Weise dargestellt werden. Eine zufriedenstellende Lösung des Problems wurde bisher leider nicht gefunden.

Eine alternative Darstellung für Dokumentengruppen (Multipoints), wären Multipoints, die in der Lage sind, sich aufzuteilen. Lässt der Benutzer seinen Mauszeiger über einem solchen Multipoint, dann teilt sich dieser in die enthaltenen Dokumente. Dabei bilden diese einen Ring aus Punkten um die vorherige Position des Multipoints. Je nach Anzahl enthaltener Dokumente besteht der Ring aus entsprechend vielen Punkten. Diese Idee wird momentan in den INVISIP-Scatterplot integriert.

Eine weitere Anregung bietet das Projekt Filmfinder [15]. In der rechten Konfigurationsleiste sind sogenannte Alphaslider angebracht. Mit diesen kann der Benutzer die Anzahl der angezeigten Filme bestimmen. Wird ein Alphaslider nach rechts bewegt, werden die Eigenschaften der angezeigten Punkte untersucht. Entspricht ein Punkt nicht mehr den gewünschten Werten, die im Alphaslider eingestellt wurden, wird der Punkt aus der Visualisierung entfernt. Diese Idee könnte auch beim Scatterplot realisiert werden. Für die verschiedenen Eigenschaften der Punkte müssten dann Slider zur Verfügung stehen. Je nach Veränderung dieser könnten dann Punkte aus- bzw. eingeblendet werden.

Eine weitere Idee zum Scatterplot ist der geplante Einbau einer ‚Focus and Context'-Komponente. Dazu sollen an den Rändern des Scatterplots Bereiche liegen, die auch Punkte enthalten können. Wird zum Beispiel ein Zoom durchgeführt, fallen die unrelevanten Punkte nicht mehr komplett aus der Visualisierung. Sie werden dann an den Rändern der Matrix des Scatterplots in speziellen Komponenten visualisiert. Bei einem Zoom-Out rücken dann die Punkte entsprechend wieder in die Hauptansicht zurück. Damit die Randkomponenten auch alle Punkte anzeigen können die aufgenommen werden sollen, müssen die Anzeigebereiche an den Rändern (im Gegensatz

zum Hauptanzeigebereich) gestaucht werden. Durch die Realisierung dieser Idee würden im Scatterplot immer alle Punkte angezeigt werden, auch bei einem Zoom.

Die Datensätze des INVISIP Projektes sollen in Zukunft eventuell auch Daten zu semantisch zusammenhängenden Dokumenten enthalten. Auf der Basis dieser Daten könnten auch im Scatterplot die semantisch nahen Dokumente in besonderer Weise visualisiert werden. Eine einfache Möglichkeit wäre eine Linie zwischen den zusammenhängenden Dokumenten zu zeichnen. Dadurch kann der Benutzer sofort erkennen, welche Dokumente miteinander verbunden sind. Er bekommt so einen Überblick über Dokumentengruppen, die mit herkömmlichen Anzeigemitteln des Scatterplots nicht visualisiert werden können.

Eine letzte Verbesserungsmöglichkeit betrifft nicht nur den Scatterplot, sondern das gesamte INVISIP-Projekt. Die vorhandenen Performance- und Speicherprobleme entstehen auch durch die mächtigen Swing-Komponenten, die in allen Visualisierungen benutzt werden. Die Firma IBM hat leichtere Javakomponenten entwickelt, die weniger Speicher benötigen. Diese Komponenten heißen SWT(Standard Widget Toolkit)-Komponenten. Sie sind weniger umfangreich als die Swingklassen, unterstützen aber die wichtigsten Eigenschaften. Ein Umbau der Swing-Komponenten der INVISIP-Visualisierungen wäre zwar ein enormer Arbeitsaufwand, würde jedoch mit Sicherheit einiges an Speicherplatz einsparen. Dieser Umbau sollte aber nur stattfinden, wenn das Speicherproblem nicht auf eine andere Weise gelöst werden kann.

Bis zum Ende der Entwicklungszeit von INVISIP werden noch einige Arbeiten an diesem System ausgeführt werden. Programmierarbeiten zu den oben erläuterten Themen werden momentan in Angriff genommen. Weitere Visualisierungen, wie ein 3D-Scatterplot und eine Circle-Segment-View werden entwickelt und in das System integriert. Interessant werden die Ergebnisse der Endevaluation des Projektes sein. Dort wird sich herausstellen, mit welchen Visualisierungen die Benutzer am liebsten arbeiten und mit welchen sie die gestellten Aufgaben am schnellsten lösen.

7 Quellen

7.1 Schriftliche Quellen

1 Salton, Gerard; McGill, Michael: Information Retrieval : **Grundlegendes für Informationswissenschaftler**; - Hamburg [u.a.] : McGraw Hill, 1987. - (McGraw-Hill Texte)

2 Laurini, Robert; Thompson, Derek: **Fundamentals of Spatial Information Systems**; ACADEMIC PRESS, 1999

3 Eibl, Maximilian (2000): **Visualisierung im Dokument Retrieval: Theoretische und praktische Zusammenführung von Softwareergonomie und Graphik Design**; Dissertation. Universität Koblenz-Landau.

4 Richard Lamers: **Studien Review 2002 Internet & E-Business**; Erscheinungsjahr: 2002; Band 1

5 Card, Mackinlay, Shneidermann 1999, **Readings in Information Visualization**

6 Memmel, Thomas: **Implementation einer tabellenbasierten Visualisierung für geo-räumliche Metadaten**; Bachelorarbeit Universität Konstanz, 2002

7 James P. Callan, W. Bruce Croft, Stephen M. Harding: **The INQUERY Retrieval System**

8 Balzert Helmut: **Lehrbuch der Software-Technik**; Heidelberg; Berlin: Spektrum Akad. Verl., 2. Aufl.; 2000

9 Müller Frank, Klein Peter, Reiterer Harald, Eibl Maximilian: **The SuperTable + Scatterplot Visualization for Metadata Retrieval**

10 Mann, T.H., Reiterer H.: **A Combined Visualization Approach for WWW-Search Results**; In: Proceedings of the IEEE Visualization 1999 (Vis ‘99), San Francisco

11 Reiterer, Harald; Mußler, Gabriela; Mann, Thomas; Handschuh, Siegfried: **INSYDER – An Information Assistant for Business Intelligence**, Proceedings of the 23 Annual International ACM SIGIR 2000 Conference on Research and Development in Information retrieval, ACM press, 2000

12 Reiterer, Harald; Mußler, Gabriela; Mann, Thomas: **Visual Information retrieval for the WWW**, in: Smith M.J. et al. (eds.), Proc. HCI International 2001, New Orleans, Lawrence Erlbaum, 2001

13 Mann, M. Thomas: Thesis – **Visualization of Search Results from the World Wide Web**; 2001

14 Nowell, Lucy T.; France, Robert K.; Hix, Deborah et al.: **Visualizing Search Results: Some Alternatives to Query-Document Similarity**. In: Frei, Hans-Peter; Harman, Donna K.; Schäuble, Peter et al. (Eds.): SIGIR 1996: Proceedings

of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval.

15 Ahlberg, Christopher; Shneiderman, Ben: **Visual Information Seeking: Tight Coupling of Dynamic Query Filters with Starfield Displays**. In: Adelson, B.; Dumais, S.; Olson, J. S. (Eds.): CHI 1994: Conference Proceedings Human Factors in Computing Systems.

16 Bartelme, Norbert: **Geoinformatik, Modelle Strukturen Funktionen**; Springer Verlag Berlin-Heidelberg 1995

17 Fairbain, D., Andrienko, G., Andrienko, N., Buziek, G., and Dykes, J.: **Representation and its relationship with cartographic visualization: a research agenda**; Cartography and Geographic Information Science, Special Issue "Research Challenges in Geovisualization"

18 Shneiderman, Ben: **Designing the User Interface, Strategies for Effective Human-Computer Interaction**; Addison-Wesley 1998

19 Christopher Ahlberg, Erik Wistrand: IVEE: **An Information Visualization & Exploration Environment**; published in Proceedings of IEEE Viz'95

20 Fishkin, K. and Stone, M.C.: **Enhanced Dynamic Queries via Movable Filters**. 1995. CHI'95 Conference Proceedings, 415-420. New York, NY: ACM Press. ML

21 Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton, and Tony D. DeRose: **Toolglass and Magic Lenses: The See-Through Interface**; Proceedings of 1993 ACM SIGGRAPH Conference, 73-80 ML

22 **The Movable Filter as a User Interface Tool**. In: Human Factors in Computing Systems: Proceedings of the CHI '94 Conference. New York: ACM, 1994. ML

23 Eric A. Bier, Maureen C. Stone, Ken Fishkin, William Buxton, and Thomas Baudel: **A Taxonomy of See-Through Tools**; Human Factors in Computer Systems, CHI '94 Conference Proceedings, ACM Press, 1994, 358-364. ML

24 Streit, U. (1995): **Statistical Analysis of Spatial Data in Geographic Informations Systems**; In: Bock, H.H. und W. Polnsek (Hrsg.): Data Analysis and information systems. Statistical and conceptual approaches. Vol. 7; Springer, Heidelberg

25 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns**; Addison Wesley Publishing Company (15. Januar 1995)

26 Pree, Wolfgang: **Komponentenbasierte Softwareentwicklung mit Frameworks**; Verlag: dpunkt.verlag; 1. Auflage 1997

27 Sedgewick, Robert: **Algorithmen in C++**; Addison-Wesley, 1. Auflage 1992

7.2 Internetquellen

Alle Internetquellen waren im August 2002 online erreichbar.

I1 InGeo IC, eine Suchmaschine für Geodaten; http://www.ingeoic.de

I2 Das INVISIP Projekt; http://www.invisip.de

I3 CommonGIS; http://borneo.gmd.de/and/CommonGISApplet/eu-pl.html

I4 MolSurf, VRML Projekt der Universität Erlangen zur Visualisierung chemischer Moleküle, http://www2.chemie.uni-erlangen.de/services/molsurf/

I5 Envision von Virginia Tech, http://www.dlib.vt.edu/projects/Envision/index.html

I6 Beispiel einer Implementation einer Magic Lens vom Xerox Parc Research Center http://www2.parc.com/istl/projects/MagicLenses/LensDemo.html

I7 DCMI Frequently Asked Questions. Dublin Core Metadata Initiative, 2001. URL http://dublincore.org/resources/faq/

I8 3D surfaces in Epidemiology; http://www.nottingham.ac.uk/psychiatry/3d.html

I9 Heuristic Evaluation, A System Checklist, von Deniese Pierotti, Xerox http://www.stcsig.org/usability/topics/articles/he-checklist.html

7.3 Abkürzungen

Abkürzung	Bedeutung
DBMS	Datenbank-Management-System
GPS	Global Positioning System
IRS	Information Retrieval System
Java API	Application Programming Interface
JVM	Java Virtual Machine
KI	Künstliche Intelligenz
MVC	Model View Controller
OO	Object Orientation
UML	Unified Modeling Language
URL	Uniform Ressource Locator
SSH	Secure Shell
WIMP	Windows Icons Menus Pointers
XML	Extended Markup Language

7.4 Bilderverzeichnis

Nummer	Quelle
1.1	Card, Mackinlay, Shneidermann 1999, **Readings in Information Visualization**, Seite 4
1.2	Card, Mackinlay, Shneidermann 1999, **Readings in Information Visualization**, Seite 4
2.1	Laurini, Robert; Thompson, Derek: **Fundamentals of Spatial Information Systems**; ACADEMIC PRESS, 1999
2.2	Vgl.: Stürmer, Günter: **Oracle7 Die verteilte semantische Datenbank**, dbms publishing 1993; Vgl.: Scholl, Marc: Vorlesungsunterlagen zu Datenbanken, 2001
2.3 2.4	Screenshot einer installierten Version des Forschungsprototypen INSYDER
2.5	Screenshots einer installierten Version des Forschungsprototypen INSYDER
2.6	http://www.invisip.de/invisip/index.html ; Bildname: szenario.gif; online im August 2002
2.7	Gundelsweiler, Fredrik; Memmel, Thomas; INVISIP Prototyp Screenshot
3.1	Gundelsweiler, Fredrik; Screenshot Windows 2000 SSH- / Telnet- Remotekonsole zu einem Linux Server
3.2	CommonGIS, Screenshot des Applets unter http://borneo.gmd.de/and/CommonGISApplet/eu-pl.html; online im August 2002
3.3	MolSurf, Screenshot der Webanwendung unter http://www2.chemie.uni-erlangen.de/services/molsurf/; online im August 2002
3.4	MolSurf, Screenshot der Webanwendung unter http://www2.chemie.uni-erlangen.de/services/molsurf/; online im August 2002
3.5	Card, Mackinlay, Shneidermann 1999, **Readings in Information Visualization**, Seite 17
3.6	Nowell, Lucy T.; France, Robert K.; Hix, Deborah et al.: **Visualizing Search Results: Some Alternatives to Query-Document Similarity**. In: Frei, Hans-Peter; Harman, Donna K.; Schäuble, Peter et al. (Eds.): SIGIR 1996: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Conference: Zürich, Switzerland, August 18 -22 1996. New York (ACM Press) 1996.
3.7	Ahlberg, Christopher; Shneiderman, Ben: **Visual InformationSeeking: Tight Coupling of Dynamic Query Filters with Starfield Displays**. In:

	Adelson, B.; Dumais, S.; Olson, J. S. (Eds.): CHI 1994: Conference Proceedings Human Factors in Computing Systems. Conference: Boston, MA, April 24-28 1994. New York (ACM Press) 1994. p. 313-317.
3.8	http://www2.parc.com/istl/projects/MagicLenses/; online August 2002
3.9	**The Movable Filter as a User Interface Tool**. In: Human Factors in Computing Systems: Proceedings of the CHI '94 Conference. New York: ACM, 1994. ML
3.10	Screenshot des Java-Applets unter: http://www2.parc.com/istl/projects/MagicLenses/LensDemo.html; online August 2002
3.11	Gundelsweiler, Fredrik; vgl. **The Movable Filter as a User Interface Tool**. In: Human Factors in Computing Systems: Proceedings of the CHI '94 Conference. New York: ACM, 1994. ML
4.1 - 4.15	Gundelsweiler, Fredrik; Memmel, Thomas; INVISIP Prototyp, Screenshots
4.16	Gundelsweiler, Fredrik; UML-Diagramm
4.17 4.18	Gundelsweiler, Fredrik; UML-Diagramm; vgl.: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns**; Addison Wesley Publishing Company (15. Januar 1995)
4.19	Gundelsweiler, Fredrik
4.20 4.21	Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns**; Addison Wesley Publishing Company (15. Januar 1995)
4.22	Gundelsweiler, Fredrik; Screenshot des Borland JBuilder 6
4.23	XML-Datensatz des INSYDER Projektes
4.24 4.25 4.26	Gundelsweiler, Fredrik; UML Diagramm auf Basis des INVISIP Quellcodes
4.27 4.28	Gundelsweiler, Fredrik; Memmel, Thomas; INVISIP Prototyp, Screenshots
4.29	Links: Screenshot der Circle-Segment-View des INVISIP Prototyp; rechts: http://www.nottingham.ac.uk/psychiatry/3d.html; online im August 2002

5.1	Gundelsweiler, Fredrik; Memmel, Thomas; Screenshot Erstimplementation Scatterplot
5.2	Gundelsweiler, Fredrik; Memmel, Thomas; Screenshot überarbeitete Implementation Scatterplot
5.3	Gundelsweiler, Fredrik; Screenshot Borland JBuilder 6
5.4	Gundelsweiler, Fredrik; Komponentenhierarchie im Scatterplot
5.5	Gundelsweiler, Fredrik; UML Diagramm auf Basis des INVISIP Quellcodes
5.6	Gundelsweiler, Fredrik; Quellcode INVISIP Scatterplot
5.7	Gundelsweiler, Fredrik; UML Diagramm auf Basis des INVISIP Quellcodes
5.8	Gundelsweiler, Fredrik; Quellcode INVISIP Scatterplot
5.9 - 5.11	Gundelsweiler, Fredrik; Memmel, Thomas; INVISIP Prototyp, Screenshots
5.12	Quellcode INVISIP Framework
6.1	Quellcode INVISIP Framework
6.2	Evaluationsergebnisse,

7.5 Zitate

Nummer	Quelle
Z1	Diann Rush-Feja, Zitat: Was sind Metadaten
Z2	Tim Berners-Lee, Zitat: Was sind Metadaten
Z3	Gartner Group von 1989