Recherche-Bericht

Inhaltsverzeichnis

1 Glossar – die 10 wichtigsten Begriffe

BorderFlow

ist ein Algorithmus zur Clusterung von Graphen, der ausschließlich lokale Informationen zu jedem Knoten nutzt. Zu einem Cluster X werden Knoten zusammengefasst, so dass die Flussstärke innerhalb von X maximal, von X nach außen minimal wird.

Cluster

(engl. Für Bündel, Gruppe, Haufen) eine logische Zusammenfassung von Objekten zu Gruppen bzw. Blöcken nach bestimmten Kriterien (bspw. die Clusterung von Graphen mit dem →BorderFlow-Algorithmus)

Graphansichten(Views)

Es gibt verschiedene Ansichten eines Graphen, die für das Produkt bedeutsam sind: radial hierarchy, spring, simple view, hierarchy circle.

Graph-Browser

ist ein Programm, das dem Anwender erlaubt sich Daten als Graph visualisieren zu lassen. Zur Funktionalität gehören das Auswählen von Ansichten und das Fokussieren auf bestimmte Aspekte.

HDN(Human Disease Network)

ist ein Graph, der menschliche Krankheiten als Knoten und ihre Beziehungen als Kantengewichte enthält.

ICD-9

Internationale statistische Klassifikation der Krankheiten und verwandter Gesundheitsprobleme. Wichtigstes weltweit anerkanntes Diagnoseklassifikationssystem in der Medizin.

Information Visualization Reference Model (IVRM)

ist ein 1999 von Ed H. Chi vorgestelltes Design-Muster. Um Informationen zu visualisieren ist eine Vielzahl von Aufgaben zu bewältigen: Datenoperation, visual mapping, Computergraphik und Interaktion. IVRM bietet eine sinnvolle Trennung dieser verschiedenen Aspekte in mehreren Schichten mit den jeweiligen Referenzen zwischen ihnen.

Komorbidität

Als eine Komorbidität werden in der Medizin ein oder mehrere zusätzlich zu einer Grunderkrankung (Indexerkrankung) vorliegende, diagnostisch abgrenzbare Krankheits- oder Störungsbilder bezeichnet (Doppel- oder Mehrfachdiagnose). Komorbiditäten können, müssen aber nicht –im Sinne einer Folgeerkrankung – ursächlich mit der Grunderkrankung zusammenhängen.

Komorbiditätsmaß

Es gibt verschiedene Verfahren die Stärke der →Komorbidität zu bestimmen. Zwei wichtige Beispiele sind Relative Risk (RR) und die Peason-Correlation (Φ)

Prefuse

ist ein auf Java basierendes Toolkit, das die Erstellung von interaktiven Applikationen zur Informationsvisualisierung ermöglicht. Es bietet eine Vielzahl von Funktionen zur Datenmodellierung, Visualisierung und Interaktion. Weiterhin bietet Prefuse neben vielerlei Techniken zur Codierung des Layout und der Visualisierung optimierte Datenstrukturen für Tabellen, Graphen und Bäume, eine Datenbankanbindung, dynamische Anfragen und integrierte Suchfunktionen.

2 Konzepte

2.1 Phenotypic-Disease-Networks (PDN)

In unserem Projekt geht es um die Visualisierung von *Human-Disease-Networks* (*HDN*), genau genommen um sogenannte *Phenotypic-Disease-Networks* (*PDN*).

Ein HDN ist ein Netzwerk von Störungen und Krankheiten und deren Korrelationen untereinander beim Menschen. Dieses Netzwerk wird in Form eines Graphen dargestellt. Die Knoten repräsentieren die Krankheiten und die Kanten die Verbindungen dazwischen. Es existieren unterschiedliche Ansätze, verschiedene Krankheiten miteinander in Verbindung zu bringen, um die Zusammenhänge zwischen ihnen analysieren zu können. Man erhofft sich damit Rückschlüsse auf mögliche Ursachen und um Prognosen für den weiteren Krankheitsverlauf – die Risiken und die Anfälligkeit für Folgeerkrankungen und Mortalität – treffen zu können.

Die drei wesentlichen Untersuchungsmethoden sind die *genetische*, die *metabolische* (stoffwechselbedingt) und die *phänotypische* (Erscheinungstypen).

In einem PDN werden Krankheiten aufgrund des gemeinsamen Auftretens (*Kookkurrenz*) bei einem Patienten miteinander in Verbindung gebracht. Die phänotypische Analyse bzw. Statistik beruht darauf, dass man das gesamte Erscheinungsbild eines Individuums mit allen seinen (Krankheits-) Merkmalen zu einem bestimmten Zeitpunkt seiner Entwicklung betrachtet.

Treten zwei Krankheiten gemeinsam bei einem Patienten auf, so sagt man sie sind komorbid (im Sinne einer Doppeldiagnose, bzw. Mehrfachdiagnose). Diese Komorbiditäten bilden die Grundlage unseres PDNs.

2.2 Graphen/Cluster

Bei Graphen handelt es sich um mathematische Objekte, die aus Knoten bestehen, welche durch Kanten verbunden sind. Von gewichteten Graphen spricht man, wenn den Kanten Werte zugewiesen wurden.

Wenn man in diesem Zusammenhang von einem Cluster spricht, meint man eine Gruppe von Knoten, die bestimmte 'Nähe' zueinander aufweisen, welche sie von anderen Knoten separiert. Es gibt verschiedenste Ansätze diese Nähe zu definieren. Ein einfaches Beispiel ist, die Kantengewichte als Abstand zwischen den Knoten aufzufassen und die Nähe somit als räumliche Nähe zu verstehen.

In dem während des Projekts verwendeten Borderflow Algorithmus werden die Kantengewichte als Fluss zwischen den Knoten aufgefasst. Ein Knoten ist einem Cluster nah, wenn der Fluss in diesem Cluster größer ist als zu allen anderen Knoten. Ein Maß dafür, wie gut der Knoten in den Cluster passt, ist die Silhouette.

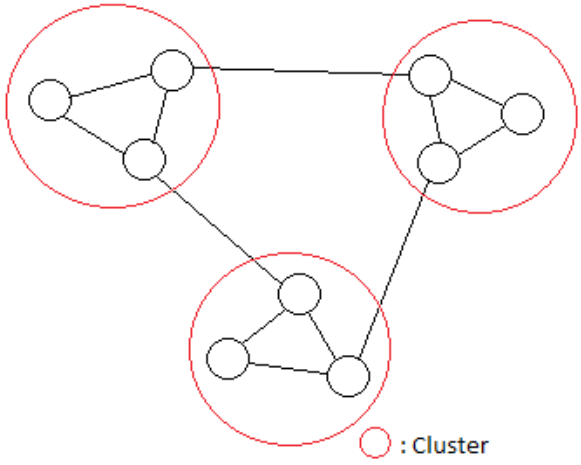

Abbildung 1: Beispiel für einen geclusterten Graphen

2.3 information visualization reference model

Das IRVM ist ein Schichtmodel zur graphischen Darstellung von Daten. Quelldaten können dabei Graphen, Tabellen oder beliebige andere Datenstrukturen sein.

Im ersten Schritt, der 'Data Transformation', werden die Quelldaten eingelesen und optional beliebigen Modifikationen unterzogen. Die Resultate dieser Operation werden in internen Repräsentation der Daten gespeichert, den 'Data Tables'.

Danach erfolgt das 'Visual Mapping', hierbei werden die Daten durch alle für die visuelle Darstellung nötigen Information, wie z.B. Farben und Formen, ergänzt. Es entsteht die 'Visual Abstraction' der Daten.

Im letzten Schritt wird der Inhalt der visuellen Abstraktionen in eine beliebige Anzahl von interaktiven 'Views' gerendert.

Der Benutzer hat dabei die Möglichkeit in jeden dieser Schritte einzugreifen um z.B. von Graph- auf Tabellenansicht zu wechseln, zu zoomen oder andere Daten zu betrachten

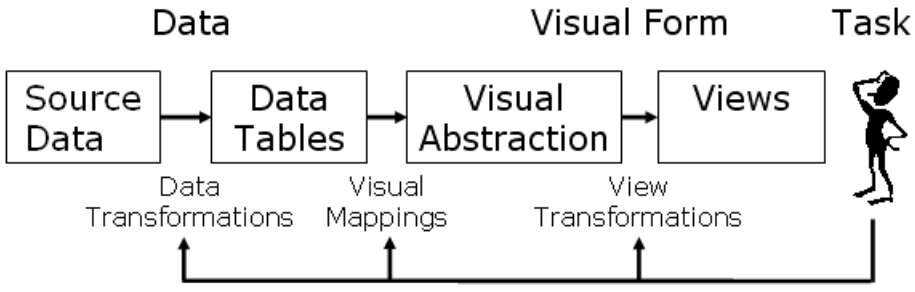

Abbildung 2: ivrm

3 Beschreibung der zu studierenden Applikation

3.1 Prefuse-Architektur

Prefuse basiert auf dem „information visualization reference model“-Konzept. Diese Architektur fordert eine strikte Gliederung der einzelnen Schritte, die erforderlich sind zum Realisieren eines Softwareproduktes, das mit Bibliotheken von Prefuse arbeitet. Zusammenfassend formuliert, umfassen die einzelnen Schritte das Erfassen der Daten, die in einem Graphen visualisiert werden sollen, das Abspeichern in ein geeignetes Format z.B. in Tabellen, das Konfigurieren des Graphen, das bedeutet z.B. Farbe der Knoten, Name der Knoten, Farbe der Kanten etc. sowie die Sicht auf den Graphen wie z.B. Radial-View. Im letzten Schritt wird die Interaktion durch den Anwender und die Darstellung auf dem Desktop durch „Views“ realisiert

„Schritt für Schritt“

Im nachfolgenden Abschnitt werden die einzelnen Schichten näher erläutert und die dazugehörigen wichtigsten Klassen mit deren Funktionen genannt.

Source Data:

Die erste Schicht im Prefuse Architekturmodell beinhaltet die Rohdaten auf dem die Darstellung des Graphen basiert. Diese Rohdaten können in einer Datenbank, im Textformat oder in einem XML-Format vorliegen, in dem die Informationen über die Knoten und Kanten des Graphen explizit angegeben sind. Für unser Projekt steht uns eine Textdatei zur Verfügung, die alle Krankheiten mit ihren Korrelationen zu anderen Krankheiten, deren Kantengewichte (RR bzw. ϕ) und die relative Häufigkeit des Auftretens der Krankheit – im Bezug auf die Gesamtheit aller Patienten – beinhaltet.

Data Tables:

Prefuse bietet für die Speicherung der Daten verschiedene Klassen an. Es bestehen die Optionen Daten in Tabellen zu speichern, wobei jede Zeile ein Objekt der Klasse Tupel darstellt. Die Klassen Graph und Tree benötigen zur Speicherung der Daten die Klassen Node und Edge. Beide Speicherstrukturen verwenden dennoch intern zum Abspeichern von den Knoten und Kanten Tabellen. Um die Quelldaten in solchen Formaten speichern zu können ist eine Transformation dieser notwendig. Solch eine Datentransformation kann einen unterschiedlichen Arbeitsaufwand beanspruchen. Dies ist abhängig vom Format der Daten. So lassen sich Daten, die in einem XML-Format vorliegen auf trivialem Weg durch die Klasse GraphMLReader auslesen und in einem Graph speichern. Für Textdateien ist ein „TextReader“ erforderlich, den uns Prefuse mit der Klasse DelimitedTextTableReader zur Verfügung stellt. Dieser Reader hat die Möglichkeit jede Zeichenkette aus jeder Zeile auszulesen, wobei die Zeichenkette in jeder Zeile z.B. durch Tabs getrennt sein muss. Des Weiteren benötigen wir einen Parser, um jeden Ausdruck, der in Form einer Zeichenkette ist, richtig zu interpretieren, z.B. als Knoten oder Kantengewicht etc.. Die Prefuse-Bibliothek bietet eine Klasse ExpressionParser, die es ermöglicht, solch einen Parser für unser Projekt zu realisieren.

Abstraction Visualization:

Diese Schicht umfasst alle Informationen, wie der Graph grafisch repräsentiert werden soll. Zentrale Klasse der Abstraction Visualization Schicht ist die Visualization Klasse. Durch Initialisieren eines Objektes dieser Klasse und Hinzufügen der Daten in Form von Tabellen-, Graphen- oder Baumstrukturen, erfolgt das Mapping zwischen den Daten und der Visualisierung. Das Mapping ermöglicht es, für jedes Tupel der Daten weitere Informationen, wie z.B. die Farbe, die Größe der Knoten sowie Kantenfarbe durch das Interface VisualItem zu speichern. Das Einstellen solcher Parameter sowie die Konfiguration von verschiedenen Sichten z.B. Fisheye oder RadialView wird mittels der Klasse Action bzw. Layout und deren Unterklassen bewerkstelligt. Ähnliche Aktionen wie z.B. Farben der Komponenten des Graphen können in einer ActionList zusammengefasst werden oder Aktionen, die in bestimmten Sichten Verwendung finden.

Für die konkrete Darstellung der jeweiligen Komponenten sind spezielle Renderer verantwortlich, die z.B. die Form, oder den Namen eines Knotens festlegen können. Für jede Komponente des Graphen, die auf dem Bildschirm dargestellt wird, wird durch eine RendererFactory der dazugehörige Renderer erfragt, um diese Komponente nach den gewünschten Konfigurationen zu zeichnen.

Views:

Durch verschiedene Views hat der Anwender die Möglichkeit die Daten aus verschiedenen Perspektiven zu betrachten. Die Klasse Display fungiert dabei wie eine Art Kamera auf ein Objekt der Klasse Visualisierung, dabei stellt das Display alle Komponenten für die aktuelle Sicht dar. Jedes Display unterstützt eine Anzahl von Interaktionen, z.B. Zoomen, Bewegen von Knoten, Bewegen des gesamten Graphen. Für jede Interaktionsmöglichkeit muss der spezifische Controller zum Display hinzugefügt werden.

3.2 Erweiterung von prefuse

Die Funktionalität der Prefuse-Bibliothek kann durch Hinzufügen neuer Klassen erweitert werden, die vorhandene Interfaces implementieren oder eine Spezialisierung der prefuse-Klassen darstellen. Weiterhin können eigene Komponenten über bestimmte Schnittstellen an Prefuse angebunden werden.

Reaktion auf Benutzeraktionen kann durch Java-Pakete awt/swing, insbesondere die Event-**Listener**, wie awt.event.MouseListener (z.B. Reaktion auf Mausklicks), awt.event.KeyListener (Tastatureingaben) realisiert werden.

Eine weitere Erweiterungsmöglichkeit wäre die **BorderFlow**-Bibliothek zur Implementierung des Graphclustering-Algorithmus.

Bei Quelldateien mit Daten, welche eingelesen werden sollen, könnte eventuell ein neuer **Reader/Parser** nötig sein, welcher die Daten entsprechend übersetzt und in die Data-Tables importiert.

Zum **Exportieren** der in Prefuse dargestellten Graphviews könnte eine Funktion zum virtuellen Drucken als Bilddatei bzw. pdf-Datei implementiert werden.

3.3 Demo-Dokumentation

3.3.1 AggregateDemo

Angepasste Schichten:

VISUALISIERUNG
ColorAction: Füllung von konvexen Hüllen, Knoten und Kanten durch Farben
ForceDirectedLayout: Positionierung der Elemente entsprechend der interagierenden Kräften

VIEW
ControlAdapter:
itemEntered: Cursor auf Knoten: Füllung des Knoten grau und Fixierung
Cursor auf konv. Hülle: Umrandung rot
itemPressed: Klicken auf Knoten: Füllung des Knoten grau und Fixierung
Klicken auf Hülle: Fixierung der Hülle, Umrandung rot
itemDragged: Bewegen des Knotens: Knoten bewegt sich, der ganze Graph wird hinter dem Knoten bewegt
Bewegen der Hülle: Hülle bewegt sich ohne Änderung der Form, der Rest des Graphen wird hinter dem Cluster bewegt.
Bewegen des Graphen: Der ganze Graph bewegt sich ohne Formänderung
ZoomControl: bei (rechten) Klicken auf den Hintergrund und gleichzeitigem Bewegen des Cursors

nachoben/unten wird der Graph verkleinert/vergrößert
PanControl: Verschiebung des ganzen Graphen ohne Formänderung bei (linken) Klicken auf den Hintergrund und Bewegen

3.3.2 Congress

Angepasste Schichten:

DATEN
DelimitedTextTableReader: Einlesen der Daten aus txt-Datei in eine Tabelle

VISUALISIERUNG
AxisLayout: Anordnung der Objekte auf der X-Achse (Staat) und Y-Achse (Betrag)
ListQueryBinding: Filterung von Daten nach Jahr (Radio-Box)
SearchQueryBinding: Filterung von Daten nach Name (Such-Box)

VIEW
ControlAdapter:
itemEntered: Füllung des Objektes (rot für Republikaner, blau für Demokraten, grau sonst) sowie Anzeige von Name, Partei-Staat, Jahr, Spendenhöhe bei MouseOver

3.3.3 DataMountain

Angepasste Schichten:

DATEN
DelimitedTextTableReader: Einlesen der Daten aus Dateien in eine Tabelle

VISUALISIERUNG
RandomLayout: die aus einer Datei eingelesenen Objekte werden zufällig angeordnet.
ForceSimulator: Simulation der auf Knoten wirkenden Kräfte (DragForce, NbodyForce, SpringForce)

VIEW
ControlAdapter:
itemEntered: Maus auf Buch-Objekt: Buch-ID wird angezeigt
itemPressed: Mausklick auf Buch-Objekt: Objekt wird fokussiert
itemClicked: Doppelklick: Öffnet die Buch-Webseite auf Amazon.com
itemDragged: Bewegen des Objektes

3.3.4 FisheyeMenu

Angepasste Schicht:

VISUALISIERUNG
ColorAction: Größe und Farbe (grau als Default, rot bei MouseOver) für Zahlen
FisheyeDistortion: Layout für die Verzerrung der Objekte

VIEW
ControlAdapter: Ausgabe des geklickten Wertes auf der Konsole. Die Methode itemClicked wird jedoch mit Methoden aus awt überschrieben.
AnchorUpdateControl: aktualisiert den Verankerungspunkt gemäß der Mausbewegung

3.3.5 GraphView

Angepasste Schichten:

VISUALISIERUNG
ColorAction: Füllung der Knoten (Default blau), bei MouseOver Fixierung (rot) und die Nachbarn von dem fixierten Knoten (orange)
ForceDirectedLayout: Positionierung der Knoten entsprechend der interagierenden Kräfte
ForceSimulator: Simulation der auf Knoten wirkenden Kräfte, wie NBodyForce, DragForce, SpringForce

VIEW
FocusControl: bei einmaligen Klicken wird das gewählte Element fixiert
DragControl: Neuzeichnen bei Objektbewegung
PanControl: Verschiebung des ganzen Graphen ohne Formänderung bei (linken) Klick auf den Hintergrund und Bewegen
ZoomControl: bei (rechten) Klick auf den Hintergrund und gleichzeitigem Bewegen des Cursors nach oben/ unten wird der Graph verkleinert/vergrößert
WheelZoomControl: Zoomen mittels des Mausrades
ZoomToFitControl: Der gesamte Graph wird in der Mitte des Displays bei (rechten) Klicken auf dem Hintergrund dargestellt.
NeighborHighlightControl:Die Nachbarn des Knotens, auf dem gerade die Maus zeigt, werden hervorgehoben (orange).

3.3.6 RadialGraphView

Es wird ein Netzwerk von Freundschaften angezeigt. Die Quelldaten (sozialnetGraph) werden aus einer xml importiert.

Angepasste Schichten:

VISUALISIERUNG
ColorAction: Wird der Mauszeiger über einen Knoten bewegt, so färbt sich seine Beschriftung rot und der Hintergrund grau.

GroupAction: Wird ein Knoten angeklickt, so wird der Baum neu berechnet und der angeklickte Knoten als Wurzel angesehen. Der Hintergrund der Wurzel färbt sich blau.

VIEW
ItemEntererd: zeigt den Titel(Namen) des Knotens, auf dem die Maus zeigt, unten Links an
ItemExited: blendet den angezeigten Namen im unteren Textfeld wieder aus

3.3.7 ScatterPlot

Hier wird ein Streudiagramm erstellt, dass verschiedene Zusammenhänge zwischen Pflanzenmerkmalen aufzeigt. Es können für die x bzw. y Achse verschieden Attribute gewählt werden. Des weiteren lässt sich die Form der Punkte, sowie die Merkmale für x bzw. y Achse in einem Dropdown-Menü verändern.

3.3.8 TreeMap

Die darzustellenden Daten werden aus der Datei chi–ontology als Baum eingelesen. Treemap visualisiert eine hierarchische Struktur von Daten, die durch verschachtelte Rechtecke dargestellt wird. Es werden nur die Blätter des Baums angezeigt.

Angepasste Schichten:

VIEW
ItemEntererd und Itemexited wurden wie bei radialGraphView überschrieben.

VISUALISIERUNG
ColorAction: BorderColorAction sorgt dafür, dass Blätter auf früheren Etagen des Baums einen helleren Rand bekommen als tiefere Blätter.
FillColorAction färbt die Rechtecke entsprechend ihrer Position im Baum heller oder dunkler. Gesuchte Begriffe werden rosa dargestellt.

Layout: LabelLayout bestimmt die Positionen der einzelnen Rechtecke(Blätter)

AbstractShapeRender: NodeRenderer übernimmt die Darstellung der Rechtecke

3.3.9 TreeView

Es wird eine Baumdarstellung der gleichen Daten wie aus TreeMap angezeigt.

Angepasste Schichten:

VISUALISIERUNG
ColorAction: Die Färbung des Hintergrunds der Knoten wird hier geregelt. Je nachdem zu welcher Gruppe (Markiert, gesucht) ein Knoten gehört, wird sein Hintergrund gefärbt.

Action: AutoPanAction zentriert den gewünschten Knoten in der Mitte der Anzeige und passt die Positionen der restlichen Knoten des Baumes an.

3.3.10 ZipDecode

Visualisiert die Verteilung und hierarchische Gliederung der Postleitzahlen in den USA.

Angepasste Schichten:

VISUALISIERUNG
ColorAction: Färbt Regionen, die mit den eingegebenen Ziffern beginnen, weiß ein.