Abstract

Das Internet ist ein buntes Netz von ungeordneten, jedoch lose miteinander verbundenen Informationen. Die Datenquellen besitzen dabei oft völlig unterschiedliche Strukturen. Auf semi- und unstrukturierte Daten wie Web-Dokumente sind keine typischen Datenbankanweisungen ausführbar. Durch Anreicherung mit Strukturen, wie es in aktuellen XML- und RDF-Formaten der Fall ist, können Informationen gefunden, entschlüsselt und miteinander in Verbindung gebracht werden, was Information Retrieval genannt wird.

Die unterschiedliche Strukturierung von Datenquellen erschwert also die schnelle Informationsabfrage bei herkömmlichen Abfragetechnologien. Es ist bisher nicht möglich, eine optimale Ausnutzung sämtlicher Datenbasen bei vertretbarem Aufwand zu realisieren.

Fast jedes Umfeld nutzt eigene syntaktische Strukturen und eine eigene Wortwahl. Keine Datensammlung selbst stellt „Weltwissen“ dar, sondern stets eine aus bestimmten Zielen getriebene und auf subjektiven Eindrücken beruhende Sicht zu einem Thema.

Mit herkömmlichen Datenbank-Abfragesprachen ist eine Wissensgewinnung aus dem Netz nicht optimal umsetzbar, da man möglichst von jeder verwendeten Datenquelle Kenntnisse über deren strukturellen und sprachlichen Aufbau besitzen muss. Das schränkt die Anzahl der verwendbaren Datenquellen für den Nutzer ein.

Aus diesem Problem heraus entwickelte sich die Gemeinschaft „Semantic Web“. Sie versucht durch intelligentes Extrahieren und Kombinieren verschiedenster Datenquellen Wissen neu zu generieren, um so zu konkreten und umfassenden Abfrageergebnissen zu gelangen. Der Anwender erhält statt einer Liste mit möglichen Verweisen eine konkrete Aussage auf die eigentliche Frage.

In dieser Arbeit wird die Aufbereitung verschiedener Daten aus unterschiedlichen Quellen zu teilweise gleichen Veranstaltungen am Beispiel eines Veranstaltungsvermittlers gezeigt, welcher auf der Grundlage der Web-Anfragesprache „Xcerpt“ arbeitet und die gewonnenen Daten sinnvoll und anwenderfreundlich zusammensetzt.

Inhaltsverzeichnis

1 Einleitung

„Ich bin jung und kenne meine Stadt. Es gibt viele Ecken, wo nur laut geschrien wird und andere, wo wirklich 'was los ist. Ich habe mittlerweile viele Adressen gefunden, bei denen ich mich zu unterschiedlichsten Events informieren kann.

Das Abgrasen der individuellen Quellen ist aber mit sehr viel Aufwand verbunden, den ich mir sparen will. “

In einer durchschnittlichen deutschen Großstadt finden jedes Wochenende eine unüberschaubare Anzahl öffentlicher Musikveranstaltungen statt, welche von unterschiedlich interessiertem Publikum besucht werden.

Aus der Sicht eines Veranstaltungsvermittlers stellt das obige Zitat eine typische und doch individuelle Meinung dar. Für jede Anfrage möchte er die richtige Anlaufstelle sein und die passenden Vorschläge parat haben. Dafür ist es wichtig, auf jedes persönliche Interesse einzugehen und selbst auf eine breite Palette an Informationsquellen in der jeweiligen Richtung zugreifen zu können. Dabei soll der Einzelne jedoch selbst auch in die Lage versetzt werden, Quellen zu wählen und andere ausschließen zu können. Somit wird eine individuelle Interessenausrichtung garantiert.

Weniger geübte Informationslieferanten gestalten ihre Informationen teils schwer verständlich und uneindeutig. Der gute Veranstaltungsvermittler versteht es, diese Aussagen in die Sprache seiner Kunden zu übersetzen. Das bedeutet oft im Detail, die Wortaneinanderreihung in eine dem jeweiligen Sprachgebrauch übliche Reihenfolge zu bringen, irrelevante Bestandteile zu entfernen und die Wortwahl den üblichen Normen anzupassen.

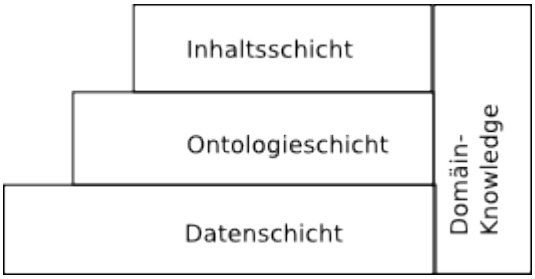
Diese angepassten Aussagen werden mit den Informationen anderer Quellen verglichen, gegenseitig ergänzt und eventuell mehrfach auftretende Sachverhalte zu einer Antwort zusammengefasst.

1.1 Grundlage

Genauer betrachtet gleicht der Vermittler die Aussagen verschiedener Quellen zunächst in **strukturell**er Weise an, indem er die Sprache der Quellen in eine Repräsentationssprache umwandelt. In einem zweiten Verarbeitungsschritt werden die Wortwahl und Begriffsbedeutungen **konzeptionell** vereinheitlicht. Nun vergleicht er das gewonnene Wissen der voneinander unabhängigen Quellen **inhaltlich** miteinander. Dabei werden fehlende Informationen ggf. mit eigenem Wissen ergänzt und mehrfach genannte Events als „identisch“ gekennzeichnet.

In der Informationsverarbeitung wird dieses Vorgehen als das Schichtenmodell für die Datenintegration angewendet (*[WiBaLpzg02]).* Es besteht aus

- einer **Schema-Integration**, welche die Struktur des Quellmaterials analysiert und auf eine gemeinsame Syntax anpasst
- einer **Ontologie-Integration,** durch die abweichende Begriffs-Termini und Relations-Bezeichnungen ausgeglichen werden,
- sowie einer **Record/Entity-Integration**, in der die nun angepassten Daten inhaltlich abgestimmt werden, um bspw. Duplikate zu entfernen



Das dies allgemein nötig ist, begründet sich in der Tatsache, dass aufgrund der Autonomie der jeweiligen Einzelinstitution in jeglicher Art von Informationsbereitstellung diese ihre Datenbasen nach subjektiven Punkten aufbaut. Der Versuch, dem durch Standardisierungen wie der „Dulbin Core Initiative“ [dublinCore06] entgegenzuwirken, kann die Abweichung lediglich minimieren, jedoch nicht verhindern.

Eine mögliche Lösung stellen neben förderativen Datenbanksystemen, welche ..... , Vermittler oder „Mediatoren“ dar [Mil99], [DesignPattern]. Sie bieten eine integrierte Sicht auf heterogene Datenbestände, indem sie diese abrufen, auf eine geeignete Auswahl an Informationen reduzieren, Unterschiede in der Datenkonsistenz bezgl. Struktur, Repräsentation und Semantik ausgleichen und so auf die gewünschte Form aufbereitet an eine Anwendung ausliefern.

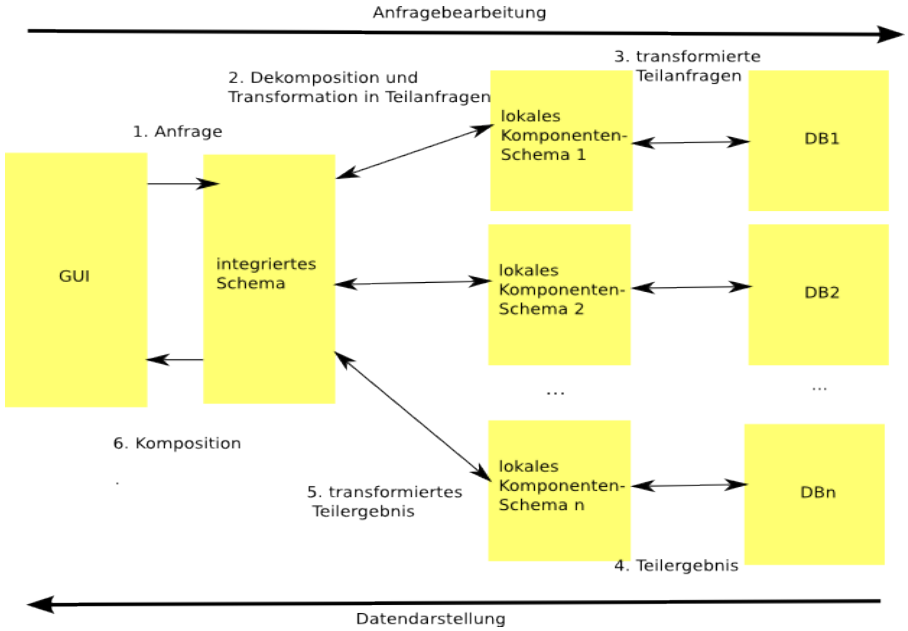


Illustration 1: Abbildung Schema-Architektur

1.2 Aufgabenstellung

Das „Institut für Programmier- und Modelliersprachen“ der Technischen Universität München entwickelt als Teil der Arbeitsgruppe IV „Query“ des Europäischen Forschungsprojektes „Rewerse“ eine Abfragesprache, durch welche der Zugriff sowie die Vereinigung von Teilergebnissen verschiedenster Ressourcen „mit Reasoning-Capabilities“ ermöglicht wird.

In dieser Arbeit sollen allgemeine Lösungsansätze zur Konsolidierung von Ressourceninstanzen zusammengetragen und dargestellt werden.

Dabei wird auf die 3 Integrationsschichten der Daten- , Ontologie- und Inhaltsschicht, welche zusammen das Domänenwissen darstellen, eingegangen. Das jeweilige Vorgehen wird strukturiert aufgezeigt. Darin werden typische technische Umsetzungen beleuchtet.

Die daraus gewonnenen Erkenntnisse werden genutzt, um die Web-Anfragesprache „Xcerpt“ mit Fähigkeiten zur Datenharmonisierung auszustatten und auf dieser Grundlage einen Mediator mit der Fähigkeit zur Harmonisierung entwerfen und prototypisch zu realisieren.

1.3 Vorgehensweise

In den bisherigen Betrachtungen wurde der Leser anhand eines praktischen Beispieles in das Thema eingeführt und mit grundlegendem Verständnis ausgestattet. Die Aufgabenstellung dieser Arbeit wurde erläutert und der weitere Ablauf folgend vorgestellt.

Im nächsten Kapitel wird die Integration abweichender Strukturen, wie sie in Illustration 1 aufgezeigt werden, näher beleuchtet. Es soll die Mächtigkeit sowie der Befehlsumfang der Abfragesprache „Xcerpt“ vorgestellt werden, welche die Grundlage der praktischen Umsetzung der hier vorgestellten Konzepte darstellt. Weiterhin werden Methoden und Programmsammlungen zur automatischen Erkennung der für Xcerpt benötigten Muster besprochen und Beispiele aus dem universitären Umfeld, welche mit diesen Technologien arbeiten, evaluiert.

Der Inhalt des 3. Kapitels wird das Vorgehen bei der ontologischen Anpassung von Datenquellen vorstellen. Der Begriff „Ontologien“ wird eingeführt, zur Auswertung benötigte Formate werden erläutert und Methoden zur kooperativen Nutzung unterschiedlicher Ontologie-Systeme vorgestellt und mit praktischen Programmumsetzungen untermauert.

Kapitel 4, welches den Schwerpunkt dieser Arbeit darstellt, führt den Leser in den stufenweisen Ablauf klassischer Daten-Integration ein. Datensätze, welche auf keinen Fall identisch sind, werden durch Grobtechniken sofort von einer näheren Untersuchung ausgeschlossen. Gewichtungsmethoden werden vorgestellt, welche die Wichtigkeit bestimmter Vergleichsduplikate festlegen.

Statische Vergleichsmethoden für Zeichenketten finden einen Ähnlichkeitsgrad zweier Wörter, so das typographische Abweichungen übergangen werden können. Durch Klassifikationsalgorithmen werden die Ergebnisse der davor angewendeten Techniken ins Verhältnis gebracht und identische von nicht-identischen Datenpaaren unterschieden.

Im 5. Kapitel werden schließlich die Komponenten von Xcerpt herausgearbeitet und die Möglichkeit der Umsetzung jeder Integrationsstufe innerhalb dieser evaluiert. Als Ergebnis wird der in der Einleitung erwähnte Veranstaltungsvermittler technisch umgesetzt.

Das abschließende Kapitel beinhaltet eine Zusammenfassung sowie einen Ausblick, in wieweit eine weiterführende Untersuchung dieses Wissensbereiches durchgeführt werden kann.

2 Schemaintegration

In unserem Zeitalter übermäßiger Werbung und Datenflut ist es wichtig, die Nadel im Heuhaufen bei der Trennung von Spreu und Weizen nicht zu übersehen.

In der folgenden Phase der Datenaufbereitung werden von unterschiedlichen Domänen, welche für die Suche ausgewählt wurden, relevante Informationen erkannt und auf ein weiterverarbeitbare Struktur gebracht.

Das heißt im Einzelnen,

- Schemas erkennen
- mit Muster manuell übersetzen und
- dadurch Instanzen extrahieren.

Bei der Transformation kann bereits domänenabhängige Syntax angepasst werden, beispielsweise das Datumsformat. Im integrierten Schema sollte auf eine standardisierte Darstellung wie DublinCore zurückgegriffen werden.

Durch diesen Schritt wird garantiert, dass nur relevante Informationen erfasst und weiterverarbeitet werden, während redundante und nicht informationstragende Bestandteile vernachlässigt werden.

Im Folgenden wird nach einer kurzen Definition die Mächtigkeit der Web-Abfragesprache „Xcerpt“ näher vorgestellt. Um die dabei genutzten Pattern automatisch erstellen zu können, folgt im Anschluß eine Ausarbeitung möglicher Struktur-Matching-Methoden. Das darin vermittelte Wissen findet anschließend durch eine Übersicht aktueller auf einer individuellen oder kombinatorischen Basis arbeitenden Strukturanalyse-ApplicationenAnwendung.

2.1 Definition

Wenn S das Schema einer Datensammlung A ist, welches aus verschiedenen Entitätsklassen besteht, besitzt S folgenden Eigenschaften:

- Jede Entitätsklasse c aus A wird mit Hilfe von S durch ein Tupel von Attributen $(attr_1, \ldots, attr_n)$ charakterisiert.
- Jedes Attribut $attr$ besitzt einen Bezeichner l und hat entweder einen einfachen Datentyp, oder referenziert eine Menge aus A (Fremdschlüssel)
- Jede Entitätsklasse (kurz Klasse) enthält eine Menge von Einträgen a_i . Diese Einträge setzen sich aus Attributen zusammen.
- Jede Klasse c aus A ist eine Menge von Einträgen $a_{1,} \ldots, a_n$. Dabei enthält jeder Eintrag a_i für jedes Attribut $attr$ (das für die Klasse durch das Schema festgelegt wird) eine Menge von Attributwerten $a_i = (attr_{1,} \ldots attr_n)$.

Ziel dieses Abschnittes ist die Angleichung zweier Schemata S_1 und S_2 zu einem Gesamtschema S_0 , so daß für jedes Attribut $attr_{1n}$ in S_1 im Idealfall ein equivalentes Attribut $attr_{2n}$ in S_2 gefunden wird und diese in S_0 als $attr_{0n}$ repräsentiert werden.

Strukturelle Divergenzen werden dabei in 2 Formen auftreten:

- als unterschiedliche Modellierungskonzepte,
- als Meta-Konflikte, wenn eine Eigenschaft $attr_1$ in S_1 als Wert und in einer anderen S_2 als Information auf Schema-Ebene abgebildet wird.

2.2 „Xcerpt“ - eine Sprache zur Webabfrage

Die Web-Abfrage hat sich zu einer komfortablen und modernen Art entwickelt, Informationen aus dem Netz zu gewinnen. Durch XSLT und XQuery für XML-Formate als auch SPARQL für RDF stellt das W3C erste Ansätze automatisierter Informationsgewinnung zur Verfügung.

Diese konventionellen Abfrage-Sprachen konzentrieren sich jeweils auf eines der vielen verfügbaren Datenformate im Netz. Sollen jedoch Daten verschiedener Quellen mit abweichenden Formaten vereint werden, so benötigt der Nutzer das Wissen verschiedener Abfragesprachen und hat die Abweichungen derer Abfrageparadigmen auszugleichen, sei es durch Skript- oder Metaprogrammierung.

Das „Institut für Programmier- und Modelliersprachen“ der Technischen Universität München entwickelt derzeit die Abfragesprache „Xcerpt“, durch welche der Zugriff sowie die Vereinigung von Teilergebnissen „mit Reasoning-Capabilities“ ermöglicht wird.

Im Gegensatz zur weit fortgeschrittenen Anfragesprache für XML-Daten, XQuery, beruht Xcerpt dabei nicht auf Pfadangaben zur Selektion von Daten. Es wird nicht der Weg vorgegeben, um an Daten zu gelangen, ebenso keine Navigation durch die Struktur/den Graph/den Baum, in welchem die Daten organisiert sind.

Xcerpt nutzt stattdessen die folgenden 3 Grundlagen:

Muster
Xcerpt stellt aktuellen Abfrage-Methoden ein patternbasierendes Konzept gegenüber. Der Benutzer gibt ein exemplarisches, eventuell unvollständiges Beispieldokument an, welches partiell die Struktur des anzufragenden Dokumentes wiedergibt. Xcerpt versucht daraufhin, alle möglichen (Teil-)Bäume der Datenquelle zu identifizieren, die diesem Pattern entsprechen. Das Wissen des Nutzers muss demnach lediglich das Aussehen der Daten umfassen, die er gerne befragen möchte. Dieser SQL und XML-QL ähnelnte Ansatz ist deklarativer Natur, da der Nutzer ausdrückt, „was“ er sucht, nicht „wie“ die gesuchten Daten erhalten werden.

Regeln
Auf einem Ansatz funktionaler Programmierung basierend nutzt Xcerpt, wie auch eine Vielzahl sog. Expertensysteme, Regeln zur Konstruktion von Ergebnissen.Wenn Teile des Rumpfes einer Regel nicht erfüllt sind, werden auch keine Ergebnisse konstruiert. Ein Xcerpt-Programm ist eine Menge von Regeln.

strikte Trennung zwischen Anfrage und Konstruktion:
Eine Regel besteht aus einer Bedingung und einer Ausgabe. Diese Bedingungen stellen die Anfragen dar: Sie befragen Daten und beschränken mögliche Ergebnisse.
Aus Ihnen folgt später die Zusammensetzung dieser Anfragen zu Ergebnissen und Konstruktion neuer Daten im Konstruktionsteil. Dort ist jedoch keine Einschränkung auf die Anfragen mehr vornehmbar.

Der musterbasierte Ansatz ermöglicht es, auch mit unbekannten, unvollständigen und heterogenen Daten umzugehen. Der regelbasierende, deklarative Programmierung ermöglicht es dem Anwender, nicht imperativ auszudrücken, wie er an die gewünschten Informationen kommt, sondern sich so weit wie möglich auf die Spezifikation der gewünschten Informationen zu beschränken. Eine Verschachtelung einzelner Regeln ist mit Hilfe der strikten Anfrage-Konstruktion-Trennung bildbar.

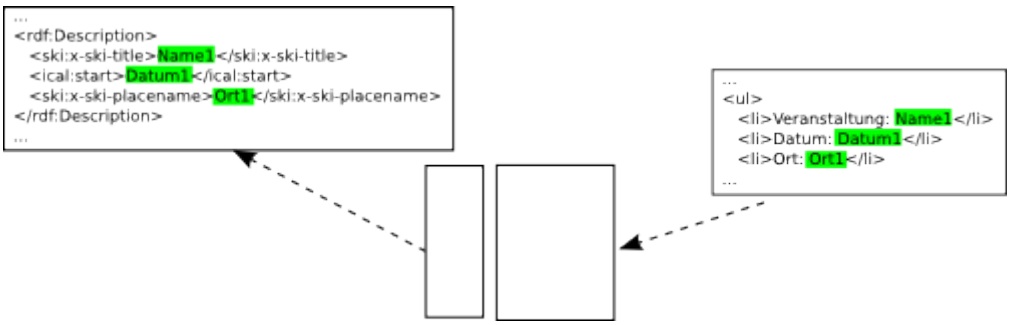
Im folgenden Abschnitt werden Regeln und deren Terme vorgestellt und ihre Abarbeitung-Eigenschaften erläutert.

2.2.1 Regeln

Ein Xcerpt-Programm besteht aus einer Menge von Regeln, welche aus einem Körper und einem Kopf bestehen.

Der Körper einer Regel ist eine Abfrage (Query), welche mit einem Daten-Term (XML-Daten) verglichen wird. Er wird im Folgenden auch als Abfrageterm bezeichnet.

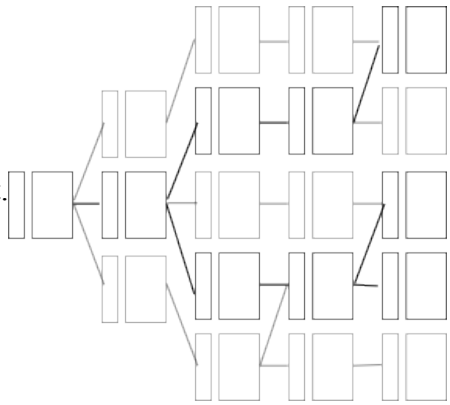
Der Kopf nutzt die Ergebnisse des Matchings und bildet damit einen neuen Daten-Term. Die ermittelten Daten sind entweder im Abfrageterm spezifiziert oder erstellt durch weitere Regeln des Programms. im Folgenden soll er auch als Konstruktionsterm benannt werden.



2.2.1.1 Verkettung

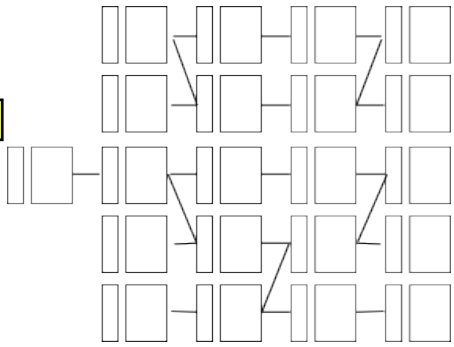
Mit Chaining oder Verkettung bezeichnet man die Strategie zur Verknüpfung von Regeln eines funktionalen Programmes zur Ableitung logischer Folgerungen. Sie arbeiten transitiv, so daß das Ergebnis einer Regel als Grundlage einer weiteren Regel dienen kann.

Rückwärtsverkettung



Backward-Chainig, auch als abfragegetriebenes Vorgehen bekannt, nutzt einfach zu verstehende, deklarative Vereinfachungssregeln. Es beginnt mit dem Anfrageteil eines Zieles und wählt davon ausgehend solche Regeln aus, welche für die Auflösung des Abfrageterms benötigt werden. Der aktuelle Abfrageterm wird durch den Zielterm der gewählten Regel ersetzt. Wenn ein erwartetes Ergebnis klein gegenüber der Anzahl der möglichen Regeln des gesamten Programmes ist, spart ein abfragegetriebenes Vorgehen demnach Resourcen. Theoretisch ist das gesamte Web als Resource denkbar, da Suchräume durch frühzeitige Inkonsistenzprüfung reduziert und komplexe Berechnungen so lang wie möglich vermieden werden können.

Vorwärtsverkettung



Dem gegenüber steht das Forward-Chaining, welches ein datengetriebenes Vorgehen darstellt. Beginnend mit der initialen Datenbasis werden hier alle Regeln iterative gegen die aktuelle Menge von Datentermen ausgewertet, bis eine Sättigung erreicht ist. Es besitzt also keine Regeldatenteile, welche nicht bereits abgeleitet wurden. Dies wird als „Fixpunkt“ bezeichnet, welcher nicht unendlich sein darf (bspw. durch einen Zyklus), da die Iteration in diesem Fall niemals terminieren würde. Ebenfalls werden durch den fehlenden zielgetriebenen Ansatzes Großteile der bearbeiteten Daten meist für die Abfrage nicht genutzt und sind uninteressant. Verwendung findet Forward Chaining in regelbasierte Abfragesprachen für traditionelle Datenbanksysteme, beispielsweise zur Materialisierung von Sichten und Sichten-Pflege.

In auf das Web bezogenen Anwendungen können beide Techniken interessant sein. Durch Forward-

Chaining können Abfrageergebnisse materialisiert werden, beispielsweise zur Generierung statischer Web-Seiten aus einem informationshaltenden XML-Dokument und einem Abfrageprogramm, welches layout-informationen hinzufügt.

Backward-Chaining ist nützlich, falls die genutzte Resource nicht lokal sondern im Web selbst zu finden ist. Es bildet die Grundlage von Xcerpt.

2.2.1.2 Kopf

Es existieren demnach 3 Arten von Regeln für Xcerpt-Programme, welche sich im Kopf-Bereich unterscheiden: Regeln für die Erstellung von Zwischenergebnissen, die als Konstruktionsregeln bei folgenden Abfragen als Quelle genutzt werden, sowie die abschliessende Regel (Goal), welche das finale Ergebnis des Programms produziert.

Da Xcerpt auf Rückwärtsverkettung basiert, tritt hauptsächlich die abfragegetrieben arbeitende *CONSTRUCT*-Regel auf. Regeln, welche datengetrieben zu Programmstart aufgelöst werden, werden durch *MATERIALISE* eingeleitet.

GOAL	CONSTRUCT	MATERIALISE
ghead	chead	mhead
FROM	FROM	FROM
gbody	cbody	mbody
END	END	END

ghead , **rhead** und **mhead** sind dabei die erwähnten Konstruktionsterme.

Ein *GOAL*-Kopf kann im Gegensatz zum *CONSTRUCT*-Kopf Informationen beinhalten, wohin der durch den Konstruktionsterm produzierte Datenterm ausgegeben werden sollte. Standardmäßig wird er im Xcerpt Datenterm-Dateiformat zum Standard-Ausgang gesendet.

2.2.1.3 Körper

gbody, rbody sowie **mbody** sind Abfrage-Terme. Sie stellen Muster dar, welche mit entsprechenden Datentermen, oftmals als XML-Daten repräsentiert, verglichen werden. Passende Resultate werden in einer (evtl. leeren) Menge von Bindungen für die Variablen der Abfrageterme repräsentiert. Eine Bindung assoziiert Datenterme mit den Variablen des Abfrage-Terms.

In der Logikprogrammierung wird dabei ein ähnliches Konzept, genannt Unifikation, genutzt. Während dort allerdings beide in die Unifikation einbezogenen Teile Terme beinhalten könnten, ist in Xcerpt eine der beiden Terme immer ein Datenterm (korrespondierend zu einem Grundterm in der Logik-Programmierung). Das Ergebnis einer Regel ist ein neuer Daten-Term, erhalten durch Anwendung aller oder einiger Bindungen zum Kopf der Regel.

Dieser neue Datenterm könnte durch den Körper einer anderen Regel abgefragt werden. Da der von der Regelanwendung resultierende, neue Term das selbe Format besitzt wie ein aus einer anderen Quelle entnommener Datenterms, kann man ihn in einer weiteren Regel weiterverarbeiten. Ein Regelkörper muss demnach nicht von einer Datei ausgelesen werden, sondern kann stattdessen Xerpt-Daten identifizieren, welche durch eine Konstrukt-Regel erstellt wurde. Dieses Rule-Chaining erhöht die Übersichtlichkeit im Code und damit die Usability für menschliche Nutzer.

Das Ergebnis einer *GOAL* - Zielregel wird entweder in eine Datei geschrieben oder am Bildschirm angezeigt, kann jedoch nicht für weitere Verkettungen genutzt werden.

2.2.2 Terme

Eingabedaten können an Xcerpt entweder als XML- oder dem kürzeren Xcerpt-Format (folgend als

XcerptV2 bezeichnet) übergeben werden. Sie werden als Datenterme repräsentiert, welche aus

- Basis-Konstanten (bspw. Strings und Integer-Werte),
- Bezeichner (Labels) und
- 2 Arten von Term-Spezifikationen (eckige und geschweifte Klammern)

aufgebaut sind.

Im Vergleich zu XML wird ein Tag-Namen durch einen Bezeichner mit anschließend folgenden Term-Spezifikationen dargestellt. Die Parenthesen beinhalten die dazugehörigen Subterme.

```
<html>                                  html {
<head>                                  head{
<title>Test</title>                     title [ „Test“ ]
</head>                                 }
<body bgcolor=“white“>                  body {
<h1>Dies ist ein Test</h1>              attributes { bgcolor [„white“] }
<p>Und es folgt Begleitung</p>          h1 [„Dies ist ein Test“],
</body>                                 p [„Und es folgt Begleitung]
</html>                                 }
                                        }
```

Geschweifte Klammern („{„ und „}“) spezifizieren, daß die direkten Unterterme in der Reihenfolge innerhalb der Sequenz ungeordnet auftreten. Durch eckige Klammern („[„ und „]“) wird hingegen eine Ordnung der direkten Subterme angezeigt. XML-Tags besitzen von Grund auf stets eine feste Reihenfolge. So folgt im obigen Beispiel der Paragraph „p“ nach der Überschrift „h1“.

2.2.2.1 Attribute

Attribute eines XML-Tags stellen eine weitere, beschreibende Instanz dar, sind jedoch als ungeordnet spezifiziert. Sie werden durch das Konstrukt $attributes\{l_1[d_1],\ldots,l_n[d_n]\}$ dargestellt, wobei $attributes$ ein fester Bezeichner, und l_i sowie d_i ($1 \le i \le n$) die Namen von Attributen und deren respektiven Werten sind.

2.2.2.2 Datenterme

Jede Basiskonstante stellt einen Datenterm dar. Falls l ein Bezeichner und $d_1 \ldots d_n (n>0)$ jeweils Datenterme sind, so ist $l[d_1,\ldots,d_n]$ ein Datenterm mit definierter Reihenfolge der direkten Subterme und $l\{d_1,\ldots,d_n\}$ ein Datenterme mit undefinierter Reihenfolge dieser.

Der Subterm eines Datenterms t ist zunächst der Term t selbst. Jeder Subterm von $t_i (1 \le i \le n)$ ist ebenfalls ein Subterm von $l\{t_1,\ldots,t_n\}$ und $l[t_1,\ldots,t_n]$, wobei l ein Bezeichner ist.

Dabei sind $t_1,..,t_n$ direkte Subterm eines Datenterms $l\{t_1,\ldots,t_n\}$ und $l[t_1,\ldots,t_n]$. Subterme von $t_1,\ldots,t_n$, mit Ausnahme von $t_1,\ldots,t_n$ selbst sind indirekte Subterme von $l\{t_1,\ldots,t_n\}$ und $l[t_1,\ldots,t_n]$.

Aus dieser Definition wird ersichtlich, dass ein Unterschied zu XML besteht, bei dessen Baumstruktur jeder Inhalt mit Ausnahme der Attribute geordnet ist. Als Schlußfolgerung korrespondieren nicht alle Xcerpt-Datenterme mit XML-Daten.

2.2.2.3 Abfrage-Terme

Abfrageterme stellen die Muster zu passenden Datentermen dar. Jeder Basisterm ist gleichzeitig ein Abfrage-Term. Jeder Abfrageterm, von der Definition her ebenso ein Datenterm. Er enthält jedoch zur Musteridentifizierung die zusätzlichen Konstrukte der Variablen, Unvollständigkeiten, Eingaberesourcen sowie Abfrage-Konjunktionen.

Variablen

Allgemeine Abfrageterme beinhalten Variablen. Sie binden bei einer erfolgreichen Übereinstimmung eines Abfrageterms bestimmte Terme des Datenterme.

Man unterscheidet zwischen verschiedenen Auftreten von Variablen:

Variablen ohne Restriktionen (*var B*) können zu jedem Subterm gebunden werden und besitzen Platzhalter-Funktion. Sie können auch an Stelle eines Bezeichners im Anfrageterm auftreten. Falls x eine Variable ist, dann ist var x ein Abfrage-Term.

Variablen mit Restriktionen (*var B -> label {}*) sind nur an Subterme des Datenterms bindbar, die mit dem Muster, welches sie beschränkt, passen. Falls X eine Variable ist und G ein Abfrageterm, dann ist $var\, X \rightarrow G$ ein Abfrageterm.

Unvollständigkeit

Da Abfrageterme die Rolle von Mustern für Datenterme übernehmen, muss für eine Abfrage nicht die komplette Struktur des Dokumentes bekannt sein. Es wird lediglich genügend Material benötigt, um eine eineindeutige Beschreibung zu erhalten. Unvollständige Abfrageterme können dabei in der Dimension der Breite, als auch der Tiefe definiert werden. Zusätzlich können optionale Subterme gekennzeichnet werden.

Breite

Unvollständigkeit in der Breite wird durch die teilweisen Termspezifikatoren „*[[*„ und „*]]*“ sowie „*{{*“ und „*}}*“ ausgedrückt. Durch sie ausgezeichnete Terme können optional zusätzliche, nicht aufgelistete Subterme enthalten. Ihnen gegenüber stehen die bereits in Kapitel [Terme] eingeführten totalen Termspezifikationen „*[*“ und „*]*“ bzw. „*{*“ und „*}*“, welche vollständig angegebene Subterm-Mengen definieren.

Eine Klammer nach dem Bezeichner zeigt demnach, daß nur die angegebenen Elemente in der gegebenen Reihenfolge auftreten. Zwei Eckklammern nach dem Bezeichner (teilweise Termspezifikation) definieren die spezifizierte Reihenfolge, jedoch können andere Elemente ebenfalls vorkommen. Geschweifte Klammern können anstelle der Eckklammern genutzt werden, um anzuzeigen, daß die Reihenfolge nicht signifikant ist.

Ein Abfrageterm mit Eckklammern findet einen Datenterm nur, wenn die Bezeichner des Abfrageterms genau zu den Bezeichnern des Datenterm passen, sowie die direkten Subterme des Abfrageterms in korrekter Reihenfolge zu den direkten Subterm des Bezeichners im Datenterm.

Falls der Abfrageterm eine totale Termspezifikation besitzt, muss die Anzahl der direkten Subterme des Datenterms identisch zur Nummer der direkten Subterme des Abfrageterms sein. Falls der Abfrageterm eine teilweisen Termspezifikation besitzt, muss die Anzahl der direkten Subterme der Datenterme größer oder gleich zur Anzahl der direkten Subterme des Abfrageterms sein.

Die zusätzlichen direkte Subterme können unbestimmt innerhalb der direkten Subterme auftreten. Ein Abfrageterm mit geschweiften Klammern findet Datenterme in ähnlicher Weise mit der Ausnahme, dass die Datenterme nicht in eine bestimmte Ordnung passen müssen. falls *W* ein Bezeichner (*l*) oder ein Variablen- Abfrage -Term ($var\, X$) sowie $q_1, \ldots, q_n (n > 0)$ Abfrageterme sind, dann sind $W[q_1, \ldots, q_n]$, $W\{q_1, \ldots, q_n\}$ sowie $W[[q_1, \ldots, q_n]]$ und $W\{\{q_1, \ldots, q_n\}\}$ ebenfalls Abfrageterme.

Tiefe

Unvollständigkeit in der Tiefe wird durch den Bezeichner *desc* ausgedrückt. *desc t* findet somit einen Term, wenn dieser den Bezeichner *t* trägt. er findet allerdings auch den Subterm eines Termes, wenn dieser Subterm den Bezeichner „t“ trägt. Dabei ist es unerheblich, ob es sich dabei um einen direkten oder einen indirekten Subterm handelt. Falls q ein Abfrageterm ist, dann stellt desc q ebenfalls ein Abfrageterm dar.

2.2.2.4 Eingaberesourcen

Eingaberesourcen werden zum Durchsuchen externer Ressourcen genutzt. Dies können derzeit lokale sowie über das HTTP-Protokoll erreichbare Quellen im xml-, xcerpt- und eingeschränkt auch html-Format sein.

Falls u eine URI und t ein Type (bspw. Format) definiert, so sind $resource[u,t]$ und $resource(u,t)$ Resourcenausdrücke.

Die URI u kennzeichnet das zu verwendende Protokoll (file: oder http:) und weitere Protokollspezifische Informationen, wie der zum Zugriff nötige Pfad. Falls das Suffix der URI von Xcerpt erkannt wird, wird t nicht benötigt.

Falls u eine URI ist und eine Resource identifiziert, dann sind $resource[u]$ und $resource(u)$ Resourcenausdrücke.

Falls r ein Ressourcenausdruck und Q eine Abfrage ist, so sind $in[r,Q]$ und $in(r,Q)$ gezielte Abfrageterme.

Falls $r_i (1 \le i \le n)$ Resourcenausdrücke sind, so sind $or[r_1,\ldots,r_n]$ und $or(r_1,\ldots,r_n)$ ebenfalls Resourcenausdrücke.

Ein or -Resourcenausdruck identifiziert verschiedene Resourcen gleichzeitig. Die Abfrage Q wird dann auf alle Datenterme von jeder der identifizierten Resourcen angewendet. Für die Zielabfrage zum Treffer muss Q den Datenterm in mindestens einer der Resourcen finden.

Ein gezielter Abfrageterm assoziiert die Resourcen (n), welche durch den Resourenausdruck r mit einer Abfrage Q beschrieben werden. Eine Resource speichert Datenterme (oder XML-Terme, welche mit den Datentemen korrespondieren). Die Bedeutung eines gezielten Abfrageterms ist, das Q nicht gegen Daten in den Resource(n), welche durch r beschrieben sind, gefunden werden muss.

Abfragen im Körper einer Regel, welche keine Resourcenausdrücke besitzen, werden mit dem durch die Konstruktionsregeln des Xcerpt-Programmes generiert Datenterme verglichen.

2.2.2.5 Abfragen

Eine Abfrage ist eine Verbindung von 0...* (möglicherweise gezielten) Abfragetermen unter Nutzung der logischen Verbindungen „and“, „or“ und „not“. Eine Abfrage ist wie folgt induktiv definiert:

Ein Abfrageterm oder gezielter Abfrageterm ist eine Abfrage.

Falls $Q_1,\ldots,Q_n (n \ge 0)$ Abfragen sind, so sind $or[Q_1,\ldots,Q_n]$, $or(Q_1,\ldots,Q_n)$, $and[Q_1,\ldots,Q_n]$ sowie $and(Q_1,\ldots,Q_n)$ Abfragen.

Falls Q eine Abfrage ist, dann ist $notQ$ ebenfalls eine Abfrage

Dabei verbindet eine „or“ Abfrage ähnlich einer „union“ in relationalen DBMSs die Ergebnismengen von Variablenbindungen von den Ergebnissen Q_1 ,..., Q_n . Eine „and“ Abfrage verbindet, ähnlich einem „join“, die gemeinsamen Variablenbindungen in Q_1 ,..., Q_n .

Die „ not “ Abfrage nutzt Negation als Fehler und ist erfolgreich, falls Q fehlschlägt.

2.2.2.6 Konstrukt-Terme

Konstruktterme sind ähnlich zu Datentermen, enthalten ebenfalls Variablen, jedoch zusätzlich den

all - und den *some* -Ausdruck. Sie werden im Kopf von Ziel- und Konstruktionsregeln genutzt und beinhalten typischerweise Variablen. Die Variablenbindungen, welche im Konstruktionsterm gebunden werden, werden im Körper der Goal- und Konstruktionsregeln (durch Abfragen) aufgelöst.

Ein Konstruktionsterm ist induktiv wie folgt definiert:

Jede Basiskonstante ist ein Konstruktterm

Falls X eine Variable ist, ist *var X* ein Konstrukt-Term. Falls W ein Bezeichner oder eine Variable und $c_1,\ldots,c_n (n \geq 0)$ Konstruktterme sind, dann sind $W[c_1,\ldots,c_n]$ und $W(c_1,\ldots,c_n)$ Konstruktionsterme. Falls c ein Konstruktterm ist, dann ist *all c* ein Konstruktterm. Falls c ein Konstruktterm und k ein natürliche Nummer, dann ist some k c ein Konstruktionsterm. Damit ist jeder Datenterm ein Konstrukt-Term.

Sammelkonstrukte

Falls vor einem Konstrukt-Term c der Quantor *all* auftritt, konstruiert c einen Datenterm per Variablenbindung der Variablen innerhalb der Liste c .

Die möglichen Datenterme, welche durch *some k c* gebildet werden, besitzen ein oberes Limit k und sind nichtdeterministisch gewählt. Es werden also nur die k ersten Terme der Listenkonstruktion ausgegeben. Falls die Variable x an die Werte „eins“, „zwei“ und „drei“ gebunden wird (im Körper eines Konstrukt-Regel oder Zielregel), produziert der Konstrukt-Term

results [*all result* [var x]]

folgendes Ergebnis:

results [

result [„one“],

result [„two“],

result[„three“]

]

während *results*[*some* 2 *result* [*varx*]]

das Ergebnis:

results [

result [„one“],

result [„two“]

]

liefert.

Durch das Konstrukt „*sort by*“ können Ergebnismengen nach in Klammern genannten Variablen auf- bzw. absteigend sortiert werden.

Das Konstrukt „*group by*“ ermöglicht die Gruppierung nach Werten einer Menge von Variablen, wie dies auch aus dem DBMS-Sektor bekannt ist.

2.2.2.7 Zusammenfassung

Zum jetzigen Zeitpunkt werden entsprechende Wrapper von Hand generiert. Im folgenden sollen nun Möglichkeiten vorgestellt werden, wie über Methoden der Strukturanalyse Schemen selbständig erkannt werden können.

2.3 Struktur-Analyse

Aufgrund des subjektiven Architekturansatzes werden kaum auf einem Gebiet exakt passende Strukturen zu finden sein. Die Identifizierung und Nutzung semantischer Korrespondenzen zwischen den Elementen zweier Schemata wird mit „Schema-Alignment“ bezeichnet und gilt es in einer Vielzahl von Anwendungen zu lösen. Im Bereich der Daten-Integrationen sowie XML-Message-Mapping in heterogenen Systemen beispielsweise vorgefunden, kommen Struktur-Analysen bei e-Commerce- oder Data-Warehousing-Applicationen stark zum Einsatz. Oftmals werden dabei diese Mapping-Aufgaben noch manuell (händisch) mit teilweiser graphischer Unterstützung erledigt, was nicht nur zeitaufwendig sondern mit zunehmender Anzahl unterschiedlicher Schemata ansteigend unpraktikabel ist.

2.3.1 Motivation

Unterschiede in Schemata können sowohl auf einer sprachlichen als auch auf der Bedeutungsebene auftreten.

So können sich sprachliche Ausdrücke gegenseitig verneinen, Schnittmengen bilden, Vereinigungen (Oberbegriff), aber auch Disjunktionen darstellen.

Auf Bedeutungsebene kann beispielsweise

- ein Term unterschiedliche Konzepte beschreiben
- ebenso verschiedene Terme beschreiben unterschiedliche Konzepte
- gleiche Terme unterschiedliche Konzepte beschreiben
- verschiedene Medellierungsparadigmen gelten
- verschiedene Modellierungskonventionen gelten
- ein abweichender Granularitätslevel auftreten
- verschiedene Abdeckung oder
- unterschiedliche Standpunkte vertreten werden.

2.3.2 Ansätze

[Noy05] nennt für die automatische Alternative der Strukturvereinigung die Herangehensweisen „Mapping“, „Artikulation“ und „Merging“.

Das Ziel des Mappings ist es, für die Abbildung gegeneinander ein Dokument als Adapter anzulegen. Es beinhaltet Elemente, welche die Symbole des einen Alphabetes mit denen des zweiten Alphabets verknüpfen.

Diese Mapping-Elemente sind hierbei jeweils 5-Tupel $\langle id, e, e', R, n \rangle$ mit

- dem uniquen Identifier des gegebenen Mapping-Elements id
- e und e, den Entitätsklassen (also z.B., XML-Elemente, Klassen)
- R als Relation zwischen e und e (z.B. Gleichheit , Generalisierung , Disjunktion)
- n als Vertrauensmessung in einer mathematischen Struktur (typischerweise zwischen [0,1])

Eine Angleichung A ist damit eine Menge von Mapping-Elementen, welche von zwei Schemata abhängt. Sie besitzt einige Multiplizitäten (1-1, 1-*, etc.) sowie einige andere Eigenschaften.

Da dabei nicht-übereinstimmende Aspekte leicht verloren gehen, kann diese Mapping gemeinsam mit den beiden Originaldokumenten präsentiert werden. Strukturen die in allgemeineren Wegen als durch

binär ausdrückbare Gemeinsamkeiten miteinander in Beziehungen stehen, so zum Beispiel durch Relationen anstatt von Funktionen, wie es bei Ontologien gegeben ist, können gemeinsam mit einem gemappten Grundübereinstimmungsdokument präsentiert werden, Artikulation genannt.

Beim Merging wir auf Grundlage aller beteiligter Quellen eine komplett neue Struktur erstellt, welche die Instanzen beider vereint.

2.3.3 Betrachtungsaspekte

Zur Analyse von Strukturen werden in [VLDBJ2001] folgende Betrachtungsapsekte genannt:

2.3.3.1 Schema- und Instanzen-Matching

Schema-Matching betrachtet das Schema einer Struktur und versucht auf Grundlage der Eigenschaften von Schema-Elementen, wie deren Name, deren Beschreibungstext, deren Beziehungentypen zu anderen Elementen (Is-a oder Sub-/Supkonzept-Eigenschaften) sowie gegebenen Constraints jeweils Übereinstimmungen in beiden Basen zu finden.

Bei Datensammlungen, welche nur in limitierter Weise nützliche Schema-Informationen enthalten, seien dies semistrukturierten Daten in Form von HTML-Seiten, gibt Instanzenwissen wichtige Einblicke in den Inhalt und die Bedeutung von einzelnen Schemaelementen. Im Extremfall ist keinerlei Schema vorgegeben und muss von den Instanzendaten manuell oder automatisch erstellt werden. Dabei kommen oftmals die im folgenden Kapitel [KapNr] ausführlich behandelten Datenharmonisierungstechniken zum Einsatz.

Aber auch bei vorhandenen Schemata kann Instanzensichtung wichtig sein, um abweichende Interpretationen der Schema-Informationen aufzudecken. Falls bei einem solchem Matching beispielsweise zwei Schema-Zuweisungen gleiche Wahrscheinlichkeit besitzten, kann auf das Element mit höherer Übereinstimmung in den Instanzen zurückgegriffen werden.

2.3.3.2 Element- und Strukturmatching

Im Idealfall eines Matchings besitzen alle Elemente eines Schemas mit Ihnen identische Elemente des zweiten Schemas auf der selben Hierarchie-Ebene. Beide unterscheiden sich hierbei lediglich durch abweichende Schema-Bezeichnungen wie „Adresse.ZIP“ und „Adresse.PLZ“. Oftmals wird diese feinste Granularität über die Attribute jedoch nicht erreicht, und Informationseinheiten überlappen einander oder sind in verschiedenen Sub- oder Subkonzepten vertreten.

Beispielsweise kann in einem Schema Postleitzahl und Ort in einem Attribut zusammengefasst sein. Besitzen nun beide Schemata nur eine bestimmte Schnittmenge gemeinsamer Informationen, wird nur eine partielle strukturelle Übereinstimmung gesucht.

Element-Level-Matcher sind auf eine Ebene beschränkt, während Struktur-Matcher darüber- und darunterliegende Konzepte ebenso als zur Struktur gehörig betrachten, um darin gemeinsame Schnittpunkte zu finden.

Im Gegensatz zu einem Struktur-Matcher ignoriert ein Elementmatcher bestehende SubKonzepte. Er ist jedoch auch nicht auf das Atomare Level beschränkt, sondern kann auch auf grobkörnigere, höhere nicht-atomare Elemente angewendet werden, jedoch nur in einer Hierarchieebene arbeitend. Er nutzen eine Vielzahl der in der relationalen Datenbank-Bearbeitung genutzen Methoden [siehe Kapitel Datenharmonisierung], und wendet diese auf Elementnamen, Beschreibung oder Datentyp des einzelnen Schema-Elementes an.

2.3.3.3 Matching-Kardinalität

Ein Element eines Schemas S1 kann Ähnlichkeit zu einem oder mehreren Elementen des Schemas S2 besitzen. Ebenso kann umgekehrt ein Element aus S2 mehrere Vertreter aus S1 haben. Es sind demnach beim Struktumatching gewöhnliche Beziehungsrelationen (1:1), mengenorientierte Fälle (1:n bzw. n:1) bis hin zu n:m – Relationen denkbar, jeweils entweder auf eine globale Kardinalität

zwischen verschiedenen Mapping-Elementen oder einer lokalen Kardinalität zu einem individuellen Mapping-Element. Element-Level Matching ist meist auf eine lokale Kardinalität von 1:1, 1:n und n:1 beschränkt. n:m – Ergebnisse erfordern in jeden Fall eine zusätzliche Strukturmatching-Analyse (siehe 2.2.2)

2.3.3.4 Sprache und Constraints

Die linguistische Annäherung wird zunächst in Form der Namensübereinstimmung verwendet. Damit kann ein gleicher Name, ein gleicher Name in kanonischer Sicht, aber auch gleiche Synonyme gemeint sein. Ebenso können zwei Bezeichner einen gemeinsamen Oberbegriff (Hypernym, bspw. in Form einer„is-a“-Beziehung) teilen, oder auf Zeichenebene Ähnlichkeiten besitzen. Domänenabhängig kann vom Nutzer auch eine Ersetzungsliste geliefert werden. Weitere Vorgehen in diesem Bereich werden im Kapitel [Onthologien] und [Zeichenmetriken] vorgestellt.

Da oftmals in Schemata Kommentare und Beschreibungstexte in natürlicher Sprache die gewünschte Semantik der Schemaelemente ausdrücken, können hierin ebenfalls Schlüsslewörter extrahiert werden.

Constraint-basierte Untersuchung beinhaltet den Vergleich zweier Elemente auf gleiche/ähnliche Datentypen, Wertebereiche, Beziehungstypen, Kardinalitäten oder etwaige Optionalität.

Weiterhin kann ihre Eigenschaft als Schlüsselcharakter (unique, Primär-, Fremdschlüssel) genutzt werden. Eine alleinige Nutzung dieser Methode würde zu falschen Übereinstimmungen führen, aber als unterstützendes Element ist die constraintbasierte Untersuchung sehr hilfreich.

2.3.3.5 helfende Systeme

Meist werden zusätzlich zu den oben genannten Faktoren noch externe Quellen wie Wörterbücher und Thesauri-Lexika für Stemming, Synonym-Identifizierungen und somit der Auflösung von Verklemmungen genutzt werden. Ebenfalls kann man bereits bestehende Matchings anderer Strukturen mit der gleichen Domäne verwenden, um eine möglicher Ausrichtung besser erahnen zu können.

2.3.4 Vorgehen

[Studer05] gibt in Bezug auf das „OntologyAlignment“ einen grundsätzlich erkennbaren Ablauf einer Strukturanalysen vor, auf den sich [dis_mehr] genauer bezieht und ihn an diversen Systemen zeigt. Da Ontologien [siehe Kapitel Ontologien] eine spezialisierte Struktur darstellen, ist das Ontology-Alignment eine Unterform der Struktur-Analyse und kann auf alle übrigen XML-Strukturen Anwendung finden.

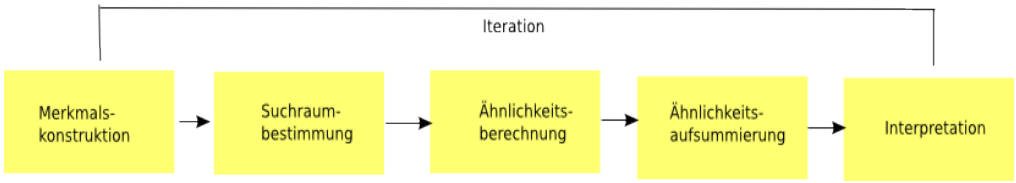


Illustration 2: Reihenfolge Strukturanalyse

Zwei Schemas aufeinander abzubilden bedeutet hierin, dass für jede Entität (Begriff, Relation, Instanz) des ersten Schemas die Entsprechung mit derselben Bedeutung im zweiten Schema gesucht wird. Es wird die unter [Kapitel Ansaetze] definierte 5-Tupel- Funktion align auf einem Vokabular E aller Entitäten $e \in E$, d.h. Begriffe, Relationen und Instanzen und der Menge möglicher Schemata S als partielle Funktion:

$\text{align} : E \times S \times S \to E,$
$\text{mit} \; \forall e \in E_{S_1} \exists f \in E_{S_2}, S_1, S_2 \in S :$
$\text{align}(e, S_1, S_2) = f \vee \text{align}(e, S_1, S_2) = \downarrow$

dargestellt.

1. Merkmalskonstruktion
   In diesem Schritt werden die Merkmale einer Entität ausgewählt, welche sie treffend beschreiben. Dies kann ein lexikalischer Eintrag einer Entität sein, aber auch eine intensionale Struktureigenschaft wie die is-a Beziehung.Desweiteren werden extensionale Beschreibungen genutzt
2. Suchraumbestimmung
   Zur Ableitung von Abbildungen in Strukturen wird ein Suchraum von Kandidaten aufgestellt. Ähnlich einer Blockingmethode stellt diese sinnvolle Kandidaten für einen Vergleich auf und sortiert alle übrigen somit aus.
3. Ähnlichkeitsberechnung
   Hier werden die im Kapitel Datenharmonisierung vorgestellten Methoden, ebenso wie Ähnlichkeitsfunktionen für Einzelobjekte oder Objektmengen genutzt, um die Ähnlichkeit zweier Entitäten zu bestimmen.
4. Ähnlichkeitsaufsummierung
   Im Allgemeinen werden dabei mehrere Ähnlichkeitswerte für ein Entitätenpaar errechnet, beispielsweise auf lexikalische Einträge oder Ähnlichkeit der Relationen zu anderen Entitäten. Es entspricht der in Kapitel Klassifikation näher vorgestellten Klassifikation.
5. Interpretation
   Die Aufsummierten Ähnlichkeitswerte werden nun zur Abbildung der Entitäten aufeinander genutzt. Möglichkeiten dafür sind ein Schwellenwert, eine Mischung aus Struktur und Ähnlichkeitswerten oder Relaxation Labelling, welches Letztere mit weiteren Heuristiken mischt [semanticIntegration_glue]
6. Um die Struktur des Schemas besser zu nutzen, wiederholt ein Algorithmus die beschriebenen Schritte.

Die grundsätzlichen Algorithmen der Stufen 2 bis 5 finden analog bei der Datenharmonisierung Anwendung, welche im Kapitel 3 näher vorgestellt wird. Jeder dieser 6 Schritte besitzt natürlich selbst eigene spezifische Methoden mit jeweils entsprechenden zu setzenden Parametern.

2.3.5 Beispiele

Eine Vielzahl von Applikationen und Frameworks haben dieses Vorgehen bereits fest umgesetzt. Es wird dabei oftmals zwischen Individual und kombinatorischen Systemen unterschieden. Während ersterer sich auf einen Algorithmus konzentriert, verwenden die Vertreter der zweiten Gruppe mehrere Algorithmen und gewichten die teilergebnisse.

Grundsätzlich verwendete Methoden sind

- Heuristisch und Regelbasierte Methoden
- Graphenanalyse
- Maschinen-Lernen-Statistiken
- Probalistische Annäherung
- Reasoning

Es soll nun Auswahl aktueller, aussichtsreicher Vertreter folgen. Diese werden in [dis_meh], [CupidVLDB01] und [ChiunchigliaEtAl'04] näher evaluierter.

2.3.5.1 SemInt („Seamless Modell Integration“)

SemInt als instanzbasiertes System assoziiert Attribute in beiden Schemen mit Treffer-Signaturen. Diese bestehen aus 15 constraint-basierenden und 5 content-basierenden Kriterien, welche von den Instanz-Werten abgeleitet und auf [0,1] normalisiert werden, so daß jedes Kriterium einen Wert in einem 20-dimensionalen Vektorraum einnimmt. Diese werden auf Grundlage ihrer Euklid-Distanz

geclustert und ein neuronales Netz mit Ihnen trainiert, welches auf das zweite Schema angewendet wird. Hierbei handelt es sich um einen hybriden Elementlevel-Matcher, welcher die Schemastruktur nicht beachtet. [CupidVLDB01]. Semint ist einer der ersten Anläufe, keine hartkodierten Kombinationen von individuellen regelbasierten Ähnlichkeiten zu nutzen, sondern statdessen einen auf Maschinenlernbarkeit basierenden Ansatz zu wählen, besitzt jedoch kein namens- bzw. graphenbasiertes Matching.

2.3.5.2 Dike

Dike ermittelt Synonym- und Inklusions-Beziehungen (is-a) eines EntityRelationship-Schemas. Objekte werden als ähnlich markiert, falls die damit verknüpften Attribute ähnlich sind. Als Ergebnis auf zwei ER-SChemata werden die Wahrscheinlichkeitswerte zwischen beiden, gemeinsam mit Synonymen und Inklusionen ausgegeben.

2.3.5.3 Cupid

Cupid ist ein Mehrzweck-Schemamatcher, welcher mit hybriden Verfahren arbeitet. Er funktioniert sowohl element- als auch strukturbasiert, spezialisiert sich jedoch auf Ähnlichkeiten von atomaren Elementen, da dort viele Schemasemantiken aufzufinden sind. Laut [CUPIDVLDB] werden ebenso interne Strukturen erkundet, wobei deren Differenzen nur gering in die Gesamtklassifizierung einfließen. Der Gesamtablauf ist in 3 Stufen gegliedert, wobei im ersten Schritt Elemente (Knoten des Schemas) auf linguistische Eigenschaften verglichen werden, inklusive externen Abgleich von Synonymen. Für den zweiten Schritt, den strukturellen Ablauf, wird das Datamodell in eine Baumstruktur transformiert und Paare daraufhin anhand ihrer Blattknoten verglichen. Ein Ähnlichkeit wird durch den gewichteten Abgleich zwischen linguistischem und strukturellem Ergebnis ermittelt. In der dritten Phase entscheidet der Schwellwert (Threshold) über Übereinstimmung. Dieser Threshold ist domänenabhängig und kann nicht im Voraus angegeben werden.

2.3.5.4 COMA

„Combination of Matching-Algorithms“ nutzt einen direkten, azyklischen Graphen, und arbeitet semiautomatisch mit Hilfe von User-Feedback. COMA befolgt ebenfalls einen Kompositions-Ansatz, welcher eine erweiterbare Bibliothek von verschiedenen Matchern zur Verfügung stellt und eine Vielzahl von Kombination der ermittelten Matching-Ergebnisse unterstützt. Hier finden ebenfalls zuvor errechnete Resultate wiederverwendung. Die Kompositionsstrategien adressieren verschiedene Aspekte des Matching-Prozesses, so z.B. die Aggregation von Matcher-spezifischen Ergebnissen und Match-Kandidaten-Auswahl. Schemata werden auch hier in gerichtete direkte azyklische Graphen transformiert, auf denen alle Algorithmen arbeiten. Jedes Schema ist unique identifiziert durch seinen kompletten Pfad von der Quelle des Schemengraphes bis zum korrepsondierenden Knoten.

COMA erzeugt auf Elementebene Übereinstimmungen von 1:1lokaler und m:n globaler Kardinalität. Der Alignment-Prozess besteht auch hier aus 3 Phasen, die iterativ durchlaufen werden und wobei die erste stets eine User-Feedback-Dialog zum Einstellen verschiedener Parameter darstellt (e.g. Auswahl des entsprechenden Matchers/Annahme, Ablehnung eines entsprechenden Entwurfes.). Daraufhin werden die individuellen Ähnlichkeiten berechnet. Diese basieren meist auf linguistische Informationen, nutzen ebenso im Hintergrund Wörterbücher und strukturelle Elemente as Kinder oder Blätter. In der dritten Phase werden die Matcher wiederum kombiniert, entweder durch ein Maximum, einer Gewichtung einem Durchschnitt oder einer MinimumStrategie. Verschiedene Annäherungen für Schwellwert-Determinationen werden gegeben. Es ist ebenso möglich, das Programm komplett automatisch zu betreiben.

2.3.5.5 CTX S-Match

S-Match ermittelt Annäherungen, um semantische Beziehung zwischen Klassen zweier Klassifikationsschemata abzuleiten. Diese werden dazu von einer Datenbank oder Ontologie

extrahiert. Sie basieren auf Untersuchung der Labels m.H. von „Wordnet“ als Synonym- Thesaurus-Überprüfung. Es sind laut [ChiunchigliaEtAl'04] ebenfalls weitere Element-Matcher vorhanden. Zusätzlich kommt hier ein SAT-Solver zum Einsatz, welcher die Schemata untersucht und dabei vor allem die Taxonomie und geschlußfolgerte Implikationen ermittelt. Er gibt entweder Gleichheit, Untermenge oder Unabhängigkeit zwischen 2 Klassen zurück.

2.4 Zusammenfassung

3 Ontologische Anpassung

Nachdem im vorangegangenen Kapitel Daten aufgespürt und strukturiert wurden, müssen die herausgefilterten Informationen in eine für den Vermittler verständliche und vergleichbare Form gebracht werden.

Dazu müssen unterschiedliche Formulierungen, die Gleiches ausdrücken, erkannt und auf ein Oberformat gebracht werden. Beispielsweise sprachlich und orthographisch voneinander abweichende Begriffe, die Gleiches ausdrücken, müssen vereinheitlicht werden. „Tanzveranstaltung“ als Oberkonzept besitzt u.a. die Synonyme und Unterkonzepte „Disko“, „Bandabend“, „Tanzabend“ oder „Salsanacht“.

Andererseits muss ein Begriff, welcher in unterschiedliche Bedeutung verwendet wird, im jeweiligen Kontext der Quelle auf seinen Sinn untersucht werden. So kann der Begriff „Blume“ als eine Krautpflanze mit auffälligem Blütenstand gemeint sein, jedoch auch der Schaum auf dem Bierglas. Duft und Aroma von Weinen sowie das bei Pferden auftretende weiße Zeichen auf deren Stirn bezeichnet man auch als „Blume“. [Brockhaus03] nennt zusätzlich die weiße Schwanzspitze von Hasen, Wildkaninchen, Fuchs und Wolf als „Blume“, sowie „Friedrich B. Blume“ (1893 – 1975), welcher als Musikforscher arbeitete.

In dieser Phase der Datenintegration wird versucht, zunächst Terme auf ihre Grund- bzw. Stammform zu reduzieren. Dies beinhaltet die Ersetzung von Umlauten, Akzentuierungen und Sonderzeichen. Wortverbünde werden in einzelne Token aufgespaltet, Abkürzungen aufgelöst. Die Bedeutung einzelner Terme wird durch Ontologien erkannt und gegebenenfalls zu Subkonzepten zusammengefasst, von Synonymen und Homonymen bereinigt und somit in den Zusammenhang der gegebenen Wissensbase eingefügt.

3.1 Ontologien

„Dub“ ist ein Unterkonzept des Musikstils „House“ (Quelle [IshkursGuideToElectronicalMusic]),besitzt selbst die Unterkonzepte ... und ..., wird auf elektronischem Weg erzeugt und besitzt ungefähr 80 Beats pro Minute.

Im Folgenden soll kurz gezeigt werden, wie mit Hilfe von Ontologien beispielsweise ein Wissensbereich für elektronische Musik greifbar und entscheidbar abgebildet werden kann. Eine Ontologie als subjektive Sicht auf einen gegebenen Anwendungsbereich, besteht dabei aus Konzepten, wie „Dub“, „House“ oder „BeatsPerMinute“, sowie den dazwischenliegenden Verbindungen.

Ontologien gliedern Informationen nach untereinander verknüpften Konzepten und nutzen dadurch komplexe Zusammenhänge optimal aus. Sie sind aus einer Anzahl von Konzepten („House“), Rollen (Tanzmusik, ...), Eigenschaften („Beats per Minute“) und Individuen (der einzelne Titel) innerhalb eines gegebenen Anwendungsbereich, es einer Anwendungsdomäne, aufgebaut. Dieser Themenbereich, welcher spezielle Erfordernisse definiert, bildet die Grundlage für eine Klassifizierungsordnung durch Ontologien, wodurch diese keine objektiven Gegebenheiten der realen Welt abbilden.

Ontologien besitzen im Gegensatz zu Datenbanken formale Beschreibungen ihrer Daten sowie Regeln über deren Zusammenhang. Sie erlauben es, Rückschlüsse aus den vorhanden Daten zu ziehen [Kramer06], Widersprüche zu erkennen sowie fehlendes Wissen aus dem Vorhandenen durch Inferenz und logisches Schließen zu ergänzen - „Ontology Learning“.

Der Typ der „Lightweight-Ontologie“ besteht aus Konzepten, Taxonomien (Klassifizierung aller

Gegenstände und Eigenschaften), Beziehungen zwischen Konzepten und Eigenschaften, welche diese beschreiben. Der zweite Typ, die sogenannten „Heavyweight-Ontologien“ besitzen zusätzlich Axiome (als gültig anerkannter Grundsatz) und Einschränkungen, wodurch die beabsichtigte Bedeutung einzelner Aussagen innerhalb der Ontologie klarer werden.
Beide Ontologien selbst bestehen aus einer Konzept- und einer Instanzen-Schicht.

3.1.1 Konzeptschichte

Die Konzept-Schicht, auch T-Box genannt, besitzt Konzepte (durch Ellipsen dargestellt) und Relationen (durch Pfeile zwischen den Ellipsen verdeutlicht). Relationen verbinden zwei Konzepte miteinander und schränken diese gleichzeitig ein. Relationen können inverse Beziehungen zueinander besitzen („A erstellt B“ - „B wurde erstellt von A“), was weitere Logik in die Ontologie integriert.

<!-- Ontologiebeispiel einfügen -->

Konzepte besitzen UnterKonzepte, sie können vererbt werden, (ein dicker Pfeil kennzeichnet die Vererbung) die ursprünglichen Relationseigenschaften bleiben dabei erhalten, können jedoch erweitert werden.. Relationen können dabei ebenfalls vererbt werden („A erhält B“==> „A ' produziert B'“). Instanzen einer Ontologie werden mit Hilfe eines einmaligen Ressourcennamens der Instanz dargestellt. Im Semantischen Web wird eine URI zur Kennzeichnung verwendet.

3.1.2 Instanzschicht

Mit einer gegebenen Ontologie wird ein Reasoner genutzt, um Anfragen zu beantworten. Die meisten Anfragen beziehen sich dabei auf die Instanz-Schicht, dem erweiterten Wissen oder auch „A-Box“ der Wissensbasis in Deskriptionslogik (DL-KB), wobei der Schwerpunkt der meisten Abfragen der Instanz-Check oder die Verallgemeinerung darstellen. Also beispielsweise ob ein Individuum ein Mitglied einer bestimmten Klasse ist.

3.2 Formate

3.2.1 Ressource-Description Framework

Das RDF-Modell besteht aus den drei Objekttypen: Ressourcen, Eigenschaftselementen und Objekten. Jeweils eine Ressource, eine Eigenschaft und ein Objekt bilden zusammen ein so genanntes RDF-Tripel. Durch dieses, es ist vergleichbar dem Satzbau aus Subjekt-Prädikat-Objekt, kann eine Aussage über ein bestimmtes Objekt innerhalb einer Domäne getätigt werden, ein Statement. Eine beliebige Anzahl an Tripeln aus Subjekten, Prädikaten und Objekten wird als Sammlung von Statements bezeichnet.

- Ressourcen (Subjekte) stellen alle die Dinge dar, welche durch RDF-Ausdrücke beschrieben werden. Jede Ressource benötigt dabei eine eindeutige Bezeichnung, welche i.A. durch Angabe eine URI gelöst wird.
- Eigenschaften/Prädikate erläutern das Subjekt näher, indem es eine Tätigkeit ausdrückt. Weiterhin stellt es einen Bezug zum Objekt her, verbindet eine Ressource mit einem Objekt.
- Objekte beschreiben den Wert eines Prädikats. Es existieren mehrere Möglichkeiten zur Darstellung von Objekten: Literale als einfachste Art, Ressourcen oder eine leere Ressource.
- Ressourcen verweisen auf weitere Ressourcen, beispielsweise um Redundanzen zu vermeiden
- Leere Knoten interpretieren noch nicht existierende oder namenlose Ressourcen
- Literale enthalten feste Werte.

3.2.2 RDF-Schema (RDF-S)

RDF(S) ist, wie auch RDF, eine W3C-Empfehlung und stellt das Konzept einer Ontologie im RDF-Format dar. Es definiert das Vokabular für eine bestimmte Domäne und repräsentiert die in der Domäne vorkommende Ressourcen, ihre Eigenschaften und Relationen untereinander. Es basiert auf einem mengentheoretischen Klassenmodell und ist für leichtgewichtige Ontologien geeignet.

RDF-S kann Klassen und Eigenschaften darstellen. `Class` legt als Klassenkonzept ein abstraktes Objekt fest und dient in Verbindung mit `rdf:type` zur Erzeugung von Instanzen. Jede Entität in einem RDF-Modell ist eine Instanz der Klasse Resource. `Property` stellt die Basisklasse für Eigenschaften dar und ist eine Unterklasse von `Resource`. `Literal` ist die Klasse für Literalwerte, also Zeichenketten etc.

Als Eigenschaften können `subClassOf` als transitive Eigenschaft zur Festlegung von Vererbungshierarchien von Klassen, `subPropertyOf` als transitive Eigenschaft zur Festlegung von Vererbungshierarchien von Eigenschaften, `domain für` den Anwendungsbereich einer Eigenschaft in Bezug auf eine Klasse sowie range für den Wertebereich einer Eigenschaft vergeben werden.

Da in RDFS die Eigenschaften per se unabhängig von den Klassen definiert werden, muss mit `domain` festgelegt werden, für welche Klassen eine Eigenschaft sinnvoll ist.

3.2.3 Ontology Web Language (OWL)

OWL, ebenfalls eine Spezifikation des W3C, ist eine formale Beschreibungssprache zum Erstellen, Publizieren und Verteilen von Ontologien, welche um die Mengentheorie erweitert wurde. Ziel ist die Beschreibung der unter [KapitelOntologien] vorgestellten Termen und deren Beziehungen einer Domäne, so daß Software-Agenten ihre Bedeutung verarbeiten und verstehen können.

OWL besteht aus dem RDF-Syntax und DAML-OIL, wobei es insgesamt mächtiger als RDF-Schema ist und damit näher an der Prädikatenlogik liegt.

OWL wurde in 3 Unterbereiche geteilt:

OWL Lite stellt eine einfach zu implementierende Untermenge für die Erschaffung einfacher Taxonomien dar.

OWL DL – besitzt die Mächtigkeit einer Beschreibungssprache, ist einer Untermenge der Prädikatenlogik 1.Stufe äquivalent. Besitzt dennoch div. Einschränkungen, um die Abbildbarkeit auf diese Logik zu gewährleisten.

OWL Full – besitzt keinerlei Einschränkungen, dadurch unterscheidbar, aber ermöglicht prädikatenlogische Ausdrücke höheren Grades.

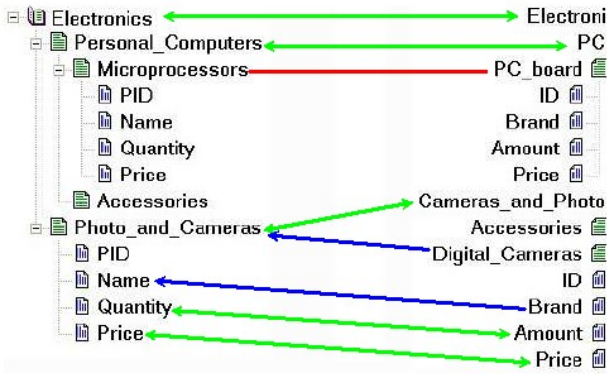
Ein OWL-Dokument besteht ebenfalls aus Klassen, deren Eigenschaften und Instanzen. Für die ontologische Anpassung unter Xcerpt kommt OWL-Lite zum Einsatz.

3.3 Ontology-Alignment

Ontologie bieten also ein Vokubular, welches eine bestimmte Domäne beschreibt, sowie eine Spezifikation der Bedeutungen der genutzen Terme in diesem Vokabular. Sie orientieren sich nicht an objektiven Gegebenheiten, sondern an den Erfordernissen aus der subjektiven Sichtweise ihres Erstellers (Dinge, wie ein Einzelner sie auffasst).

Verweisen dabei Ontologien auf sich untereinander, entsteht ein heterogenes System. Verschiedene Ontologien streifen hierbei gleiche Domänen /Anwendungsgebiete. „Ontology Alignment“ bzw. „Ontology-Matching“ sind ein vielversprechender Ansatz, diese Heterogenität zu beseitigen, indem sie Korrespondenzen zwischen semantisch verwanden Entitäten einer Ontologie finden und

- Ontologien verschmelzen (Ontology-Merging)
- Abfragen (Queries) beantworten
- Daten übersetzen
- und dadurch eine Navigation durch das Semantic Web erlauben



3.3.1 Beispiele

Als Erweiterung der unter [Kapitel Beispiele] vorgestellten Struktur-Matcher sollen hier Applicationen mit Schwerpunkt auf Ontologie-Analyse und -Merging aus dem universitären Umfeld vorgestellt werden. [dis_meh] nennt ONION, SMART, LSD/GLUE, OLA sowie im Zuge der Dissertation umgesetzte FOAM als die verbreitetsten Vertreter.

3.3.1.1 ONION

Das „Ontology compositION system“ ist eine Applikation, um Heterogenitäten zwischen Ontologien zu erkennen. Der Ansatz geht dabei davon aus, daß die komplette Neuerstellung aus 2 vorhandenen Ontologien („Merging“) zu teuer und ineffizient ist. Daher liegt der Fokus auf der Kreation von Artikulationsregeln [vergl.Einleitung] welche daraufhin korrespondierende Konzepte verlinkt. Um ein manuelles Erstellen dieser Regeln zu vermeiden, wird ein semi-automatischer Ansatz genutzt, welcher heuristisch auf verschiedene einfache Relationen wie Bezeichner, Kommentare, Einordnungshierarchien und Attributwerte zurückgreift. Dem Nutzer wird das jeweilige Zwischenergebnis zur manuellen Bestätigung vorgestellt. Die Artikulationsregel-Verlinkung kann angewendet werden, wenn eine Anwendung Informationen von beiden Ontologien anfordert.

3.3.1.2 SMART / Prompt, Anchor-Prompt

Smart ist ebenfalls ein auf Linguistik basierender Ansatz, welcher in [NoyMusen99] vorgestellt, als Plugin for „Protoge“, dem Program für Ontologie-Erstellung der Stanford-University, weit verbreitet ist. Es überprüft, ähnlich „Onion“ Konzeptnamen auf Ähnlichkeit sowie findet übereinstimmende Relationen und Attribute, worauf es 1:1-Übereinstimmungen von Ontologie-Entitäten identifiziert.

Prompt stellt die semi-automatische Annäherung an die Herausforderung des Ontology-Mergings und -Alignments dar. Diese basiert auf dem SMART-Algorithmus. Nachdem über das Label-Matching gefundene Übereinstimmungen dem Nutzer angezeigt wurde, entscheidet dieser, welche Mergings durchgeführt werden sollen. „Prompt“ zeigt hierbei während des Mergins mögliche Inkonsistenzen wie Namenskonflikt, o.ä.. Dem Nutzer bleibt die Entscheidung, wie er manuell damit umgeht.

Anchor-Prompt erweitert diesen Workflow um zusätzliche Ontologie-Strukturuntersuchungen. Es werden erkennbar gleiche Abschnitte beider Ontologien, sog. „Alignment Pairs“ durch Ankerpunkte festgesetzt (meist identifiziert aus zeichenbasierten Vergleich der Entitäten oder direkt durch den Nutzer. Mit diesem Ankerpunkt-Wissen wird die Ontologie mit dem Ergebnis traversiert, das Vorschläge für zusätzliche Übereinstimmungen für Entitäten zwischen den bekannten Ankerpunkten gezeigt werden. Nach der Zuweisung durch den Nutzer folgt die nächste Iteration.

3.3.1.3 LSD/GLUE

Das „Learning Source Description“-System nutzt maschinen-lern-techniken, um eine unbekannte Datenquelle gegen ein vorher bereits determiniertes (erkundetes) globales Schema zu vergleichen. Durch ein vom Nutzer gegebenes Übereinstimmung einer Datenquelle zum globalen Schema, untersucht der Einleitungs-Schritt Instanzen dieser Datenquelle, um den Lernalgorithmus zu trainieren, und entdeckt dabei charakteristische Instanz-Muster und Übereinstimmungsregeln. Falls die Konzeptinstanzen des zweiten Schemas mit dem ersten Klassifizierer übereinstimmen, kann das Konzept als identisch angesehen werden. Das individuelle Matcher-Ergebnis wird daraufhin wiederverwendet, um einen allumfassenden „globalen“ Matcher zu trainieren. Mit Annahme dieses Matchers ist es nun möglich, Übereinstimmungen zwischen globalen Datenquellen und neuen Quellen zu erkennen.

LSD wurde später zu GLUE erweitert . GLUE orientiert sich mehr in Richtung Ontologien. Wie auch für LSD, sucht es die ähnlichsten Konzepte in 2 Ontologien unter Verwendung verschiedener Matcher.

Seine Lernkomponente erkennt Konzept-Klassifizierer (matcher) für Instanzen, basierend auf Instanz-Beschreibungen, bspw. den textuellen Inhalt einer Webseite. oder ihre Bezeichnung. Faktisch ist GLUE eine Multi-Learning-Strategie, weil es eine Vielzahl unterschiedlicher Typen von Informationen gibt, welche Konzeptklassifizierer nutzen können. Sie reichen von Instanz-namen zu Wort-Frequenzen in Dokumenten oder Werte-Formaten. (Value-Formats)

Von diesen gelernten Konzeptklassifizierern leiten sie ab, ob Konzepte in 2 schemen zueinander korrespondieren .

Konzepte und Relationen werden zusätzlich durch Nutzung von „relaxation Labeling“ verglichen . Diese „relaxation labeling“ besitzen den Hintergrund, dass die Bezeichner eines Knotens, oder auch Übereinstimmungen zu einer Eintität typischerweise von den Merkmalen der Knotennachbarschaft im Graphen beeinflußt werden.

<!-- Graphen-Grafik? -->

Eine lokal optimale Übereinstimmung für jede Entität wird unter Nutzung der Ähnlichkeitsresultate von benachbarten Entitätspaaren einer vorhergehenden Runde erkannt. Die individuellen Constraint-ähnlichkeiten werden für die finale Alignment addiert – Wahrscheinlichkeit- die zusätzlichen relaxation-labelling, welche die ontologische Struktur angeht, basiert wiederum auf manuell enkodierten, vordefinierten Regeln. Normalerweise muss man alle möglichen Labeling-Konfigurationen überprüfen, welche die Alignments aller weiteren Entitäten mit einbeschließt. Die Komplexität der hier vorgestellten Konzepte wird durch eine sensible Partitionierung der Ergebnisse performant gehalten. So werden z.B. Bezeichnergrupen mit gleichen Merkmalen gruppiert und damit nur einmal verarbeitet

Der Glue-Maschinenlearning-approach ist angebracht für ein Scenario mit extensive textuellen Instance-Beschreibungen, aber wird nicht auf rein Ontologie-schema behafteten Quellen wirken. Weiterhin können Relationen oder Instanzen nicht direkt in GLUE verbunden werden.

3.3.1.4 OLA

Der OWL Lite Aligner nutzt verschiedene Komponenten der involvierten Ontologien, um Ähnlichkeiten zu erkennen. Die Basis-Ähnlichkeit werden von den Bezeichnern berechnet - Iterativ beeinflußen sich diese basis-ähnlichkeit gegenseitig, bis die Ähnlichkeiten zwischen allen Paaren der beiden Ontologien untereinander gut ausbalanciert sind. In jeder Iteration werden die Ähnlichkeiten unter Hinzunahme der Ähnlichkeiten der Nachbarknoten rekalkuliert, wobei Nachbarschaft heißt, daß eine Beziehung zu ihnen besteht.

OLA ist somit eine Annäherung, welche neben element- ebenfalls strukturinformationen nutzt, Die fließenden Ähnlichkeiten sind in Abhängigkeit zu ihrer Art (subsumption, instantiation,...)

unterschiedlich gewichtet, . Der Nutzer setzt diese Gewichte in Abhängigkeit zu seinen Präferenzen. Das Finden der korrekten Übereinstimmung ist ein Optimisierungsproblem mit maximaler Ähnlichkeiten. Der OLA gibt 1:1-Alignments von Konzepten, Realtionen und Instanzen zurück.

3.3.1.5 FOAM

Das „Framework for Ontology Alignment and Mapping." ist aus der Dissertation Dr. Marc Ehrics entstanden und setzt die in dieser Arbeit erarbeiteten Erkenntnisse um. Es nutzt KAON2 als OWL-DL-interpreter und greift auf WEKA, ein Open-Source-Tools für maschinenlernbare Algorithmen, sowie das bereits mehrfach erwähnte WordNet, ein Online-Synonym-Datenbank für englische Begriffe, zurück. FOAM unterstützt OWL-DL-Formate und ist durch seine flexiblen Klassen für Feature-Selektion sowie einer Vielzahl von Bibliotheken für Ähnlichkeitsmessungen (vergl. Kapitel Datenharmoisierung) sehr gut erweiterbar. Verschiedene Kombiationsstrategien für die einzeln erzielten Ähnlichkeitsmessungen sind möglich, von simpler Durchschnittsberechnung, hin zu maschinenerlernbaren Entscheidungsbäumen bis zu neuronalen Netzen. Eine Eingreifen des Nutzers in den iterativen Prozeß, z.B. zur Korrektur von Zwischenergebnissen ist möglich. Ein Plugin für die Plattform OntoMap ist ebenfalls vorhanden.

3.4 Vorgehen

Die bisher beschriebenen Techniken dienen dazu, Domänenwissen meiteinander zu vereinen. Dadurch entstandene Konzeptsammlungen sind auch im Bereich der ontologischen Anpassung von Begrifflichkeiten und Instanzen nutzbar. Grundsätzlich sollen hier semantische Abweichungen erkannt und durch einheitliche, auf syntaktischer Ebene vergleichbare Zeichenketten ersetzt werden. Dazu kommen die folgenden Methoden zur Anwendung.

3.4.1 Morphzerlegung

Mit Morphzerlegung oder Stemming ([Ferber03]) wird ein Vorgang bezeichnet, bei dem Worte auf ihre grammatikalischen Grundformen oder kleinsten bedeutungstragenden Bausteine reduziert werden. Nach [SprachSynth06] werden Wortformen, welche durch Konjugation, Deklination oder Steigerung enstanden sind (Flexionsformen), auf ihre Grundform zurückgesetzt.

In der Derivatsanalyse werden komplexe Wörter, welche durch Anhängen von Suffixen oder Prefixen an ein lexikalisches Grundwort entstanden sind, zerlegt. Semantische Änderungen wie bei „unfruchbar", welches aus „un" und „fruchtbar" zusammengesetzt wurde, werden mit einem entsprechenden Negations-Flag gekennzeichnet.

Die Struktur von Wortzusammensetzungen unterliegt nun der Kontrolle, so zum einen die Aufspaltung von Grund- und Bestimmungsworten (Bier+Fass -> Bierfass), zum anderen grammatikalisch und semantisch gleichberechtigte Bestandteile, wie bei „Schnee"+"Regen".

Diese Analysen können durch lokale Synthesen asugehend von Grundformenlexika und Generierungsregeln erfolgen. Weiterhin wird im Ontologie-Umfeld oftmals die Online-Analyse über Dienste wie „WordNet" [wordnet06] oder „OpenThesaurus" [openthesaurus06] genutzt.

3.4.2 Korrekturlisten

Unterstützende kommen Korrekturlisten von zu ersetzenden Zeichenketten zum Einsatz, welche bestimmte Wörter, Abkürzungen und Einzelzeichen eliminieren oder durch Synonyme ersetzen.

Zum Beispiel können Variationen von „-straße" durch „-strasse", „-str" oder „Str." dargestellt sein und in diesem Schritt durch eine einheitliche Schreibweise repräsentiert werden.

Dabei ist zu beachten, dass diese Ersetzung kontextabhängig ist, als Eigenname (bsp. „Joseph Strasse") darf dies nicht verändert werden. Jeder Eintrag einer solchen Liste besteht aus einem String (welcher einen oder mehrere Wörter oder ein einfaches Zeichen darstellt) und einem

korrespondierenden Ersetzungsstring. Der Eingabestring (die Source) wird nach jedem Eintrag in dieser Liste durchsucht und falls ein Originalstring gefunden wird, durch die korrespondierende Ersetzung ersetzt.

Korrekturlisten sollten dabei nach Originalstring-Länge sortiert und abgearbeitet werden, sodaß lange Zeichenketten zuerst gefunden und ersetzt werden. (genauste Regel zuerst). Ebenfalls ist auf Leerzeichen vor und nach der jeweiligen Ersetzung zu achten. Sie sind wichtig, da ansonsten auch Zeichenketten innerhalb von Eigennamen verändert werden könnten („bsp“ innerhalb „Abspiel“ nicht durch „abeispieliel“ ersetzen).

Das Endprodukt der Datenreinigung ist ein neuer String, in welchem jedes Auftreffen eines in der Korrekturliste gefundenen Substrings durch einen entsprechenden Ersatzstring ersetzt wurde.

3.4.3 Tagging

Falls nicht bereits mit dem Schema geliefert, ist ein Tagging durchgeführt, wobei der Eingabestring an Leerzeichen in Einzelstrings aufgeteilt in eine Liste Wörter, Nummern und möglichen Trennzeichen wird. Einzelnen Wortbestandteilen werden Bezeichner zugeordnet. Dieser Schritt wird oftmals parallel zu Morphemzerlegung und Korrekturlisten durchgeführt, da das Tagging die Einordnung in verschiedene ontologische Gattungen beinhaltet und damit eine Korrekturlisten-Abarbeitung gestattet.

Aus „Prof. Dr.-Ing. habil. Erwin P. Stoschek“ wird demnach ['Prof.','Dr.-ing.','habil.','Erwin','P.','Stroke']. Über Nachschlagetabellen und Regeln wird jedes Element der Liste mit einem oder mehreren Tags versehen (Bsp.: Vorname, Nachname, Postleitzahl, etc). [Febrl05] geht an dieser Stelle soweit, spezielle Teile durch verallgemeinerungen zu ersetzen, bspw. „Dr“, „Doktor“, „doc“, „Phd“ und „md“ einheitlich durch „dr“. Die getaggten Zeichenketten werden nun in die entsprechend korrekten Ausgabefelder segmentiert (Mit „Strassennamen“ getaggte Zeichenketten in das DB-Feld für „Straße“).

Nach einer ontologischen Analyse können bisweilen vertauschte Wortfelder („Swapping“ [Tailor06] identifiziert und korrigiert werden.

4 Inhaltsintegration

„In here, thi trik is thinkin rite. Thas all u 1/2 2 do. U ½ 2 think rite. U ½ 2 b dairing & koshis, u ½ 2 b ver sensibil & totily mad. Moast ov ol u ½ 2 b cluvir, u ½ 2 b ingenius. U ½ 2 b abil 2 use whatevir is aroun u, & thass whot it reely cums doun; [...] so iss up 2 u reely what yoos u make ov it aftir that; iss ol about injinooty [...]“
[Iain M. Banks „Feersum Endjinn“, 1994]

Bei einer abstrakten Datensammlung, handelt es sich um eine Reihe von Entitätsklassen, die jeweils durch eine bestimmte Menge von im Schema vereinbarten Attributen charakterisiert werden. Diese Attribute können sowohl informationshaltende Zeichenketten (einfaches Attribut) als auch Fremdschlüssel auf andere Datensätze (referenzierte Attribute) darstellen.

Nachdem im Kapitel [KapitelNrSchemaIntegration] bereits ein allgemeingültiges Vorgehen bei der Anpassung der Schemata heterogener Quellen vorgestellt wurde, folgt hier der entsprechende Part auf Instanzebene.

Diese Standardisierung umfasst i.A. die Auflösung von Inkonsistenzen in der Art und Weise, wie Informationen in den Daten repräsentiert oder umgesetzt sind.

Im Gegensatz zu reinen Schema-Integrationen ist die Entitätenanzahl hier um ein Vielfaches höher. Daraus ergibt sich das Bedürfnis, auf iterative Techniken weitesgehend zu verzichten und die Abarbeitung linear zu gestalten.

Sie besteht bei großen Datenbeständen oftmals aus den Stufen:

- Grobauswahl
- Ähnlichkeitsfindung
- Bewertung

Ausgewählte Vertreter der jeweiligen Techniken werden darauf folgend genauer untersucht und gegeneinander gewichtet.

Abschließend soll auf Software verwiesen, welche die vorgestellten Algorithmen implementiert und zur Verfügung stellt.

4.1 Ansatz

Nach der Definition aus [KapitelSchemaIntegrationDefinition] wurden durch die bisher abweichende Schemata $S_{1,\dots},S_n$ auf ein gemeinsames Schema S_0 angepasst. Jede Entitätsklasse der beiden Datensammlungen A und B besitzt somit die selben Attribute $(attr_1,\dots,attr_n)$.

In der „Ontologie-Integration“ wurden Inkonsistenzen durch
- Synonymen,
- Homonymen,
- Abkürzungen und
- Füllwörtern ohne Informationsgehalt

beseitigt, so daß die in den jeweiligen Attributen verwendeten Konzepte sich so weit wie möglich ähneln.

Duplikate entstehen durch die mehrmalige Erfassung eines Meta-Datensatzes innerhalb von verteilten Datenbasen. Die erfassten Daten besitzen aufgrund der autonomen Einzelinstitutionen immer noch Abweichungen, seien dies Eingabefehler, Erfassung in unterschiedlichen Sprachen, Erfassung an unterschiedlichen Orten, usw.

In diese Stufe sollen folgende Abweichungen erkannt und über eine Ähnlichkeit klassifiziert werden:

- typographische Fehler,
- Datenaufnahme-Fehler wie Wortvertauschungen (Name Vorname = Vorname Name)
- falsche Buchstabierung
- Integration multipler Quellen

Die Aufgabe der Harmonisierung besteht in diesem Bereich demnach in der Identifizierung und der Zuordnung von Duplikaten anhand inhaltlicher Informationen aus einer begrenzten Anzahl von Datenfelder, sowie der Verknüpfung von als identisch erkannter Klassen.

Die Grundlage eines entscheidungstheoretischen Ansatz bildet das Modell von Fellegi und Sunter [Fellegi69] . Um einen Überblick zu geben, wird das Modell durch Terme geordneter Paare in einem Produktraum beschrieben.

4.1.1.1 Duplikate

Seien nun die Elemente einer Datensammlungen A und B mit a und b benannt. Es wird angenommen, dass einige Elemente in A und B identisch sind.

<!-- Im Falle einer Duplikatesammlung bestehe dabei B aus nur einer Entitätsklasse mit n Attributen $attr_1, \ldots, attr_n$ -->

Die Menge geordneter Paare

$$A \times B = (a, b) : a \in A, b \in B$$

ist demnach die Vereinigung von 2 disjunkten Mengen von Übereinstimmungen

$$M = \{(a, b) : a = b, a \in A, b \in B\}$$

und Abweichungen

$$U = \{(a, b) : a \neq b, a \in A, b \in B\}$$

Die Datensätze, welche sowohl mit Elementen aus A als auch aus B übereinstimmen, werden durch $\alpha(a)$ und $\beta(b)$ abgebildet. Der mit den Datensätzen verbundene Vergleichsvektor γ ist definiert durch:

$$\gamma[\alpha(a), \beta(b)] = \{\gamma^1[\alpha(a), \beta(b)], \gamma^2[\alpha(a), \beta(b)], \ldots, \gamma^K[\alpha(a), \beta(b)]\} \quad ,$$

wobei jeder der $\gamma^i, i = 1, \ldots, K$ Stützstellen je einen Vergleichsoperator repräsentiert . So könnte γ^1 die Übereinstimmung des Geschlechts zweier Vergleichssätze umfassen, γ^2 wäre bspw. der Vergleich auf Idendität der Nachnamen.

Bei Eindeutigkeit wird die Funktion γ über $A \times B$ mit $\gamma(\alpha, \beta)$, $\gamma(a, b)$ oder γ bezeichnet. Die Menge aller möglichen Realisationen von γ wird mit Γ bezeichnet. Die Auftrittswahrscheinlichkeit von $\gamma(a, b)$ falls $(a, b) \in M$ ist gegeben durch

$$m(\gamma) = P\{\gamma[\alpha(a), \beta(b)] | (a, b) \in M\}$$

$$m(\gamma) = \sum_{(a,b) \in M} P\{\gamma[\alpha(a), \beta(b)]\} \cdot P[(a, b) | M]$$

analog dazu wird die Auftrittswahrscheinlichkeit von γ für $(a, b) \in U$ durch $u(\gamma)$ angegeben.

Wird ein Vektor von Informationen $\gamma(a, b)$ verbunden mit einem Paar (a, b) betrachtet, so soll man die Möglichkeit haben, ein Paar als

A_1	den gleichen Sachverhalt ausdrückend (Link),
A_2	möglicherweise den gleichen Sachverhalt ausdrückend (possible Link) oder
A_3	abweichende Sachverhalte ausdrückend (Nicht-Link)

ausweisen zu können.

4.1.1.2 Klassifikation

Eine Verlinkungsregel L ist dafür eine Abbildung auf dem Vergleichsbereich Γ , auf einer Menge von zufälligen Entscheidungsregeln $D=\{d(\gamma)\}$, wo

$$d(\gamma)=\{P(A_1|\gamma), P(A_2|\gamma), P(A_3|\gamma)\}, \gamma\in\Gamma$$

und

$\sum_{i=1}^{3} P(A_i|\gamma)=1.$, so daß jedes Element einer der 3 Gruppen angehört.

4.1.1.3 Fehler

Dabei können 2 Typen von Fehlern, die mit den Link-Regeln verbunden sind, auftreten:

Ein **Typ I** - Fehler tritt auf, wenn ein unpassender Vergleich fälschlicherweise verlinkt ist. Dieser besitzt die Wahrscheinlichkeit

$$P(A_1|U)=\sum_{\gamma\in\Gamma} u(\gamma)\cdot P(A_1|\gamma)$$

Ein **Typ II** - Fehler tritt auf, wenn ein passender Vergleich fälschlicherweise nicht verlinkt ist. Er besitzt die Wahrscheinlichkeit

$$P(A_1|U)=\sum_{\gamma\in\Gamma} m(\gamma)\cdot P(A_1|\gamma)$$

Die Güte einer Duplikate-Analyse wird gekennzeichnet von der Vollständigkeit („Recall“) und der Präsizion („Precision“) ([Wiki06]).

Vollständigkeit

Der Recall beschreibt die Anzahl der als Duplikate erkannten Paare zu den in Wirklichkeit existierenden Paaren.

Zur Definition dieser Größen sei eine ideale Referenz-Duplikate-Zuordnung R sowie eine weitere Duplikate-Zuordnung A gegeben, so daß

$R(A,R)=\frac{|R\cap A|}{|R|}$ gilt.

Abgeleitet kann gesagt werden,daß ein hoher Recall auf viele gefundene Duplikate hinweist, ohne anzugeben, wieviele davon korrekt ermittelt wurden. Ist der Recall also sehr hoch, kann von vielen flasch-korrekten ausgegangen werden.

Präzision

Die Präzision beschreibt daraufhin, wieviele der gefundenen Duplikate auch wirklich welche sind. Es wird damit das Verhältnis von korrekt Richtigen und Inkorrekt-Richtigen bestimmt.

$$P(A,R)=\frac{|R\cap A|}{|A|}$$

Die Präzision, invers zur Fehlerrate der Erkennung, gibt die als korrekt erkannten Richtigen eines Duplikatevergleichs an. Sind alle Duplikate korrekt erkannt, umfaßt die Präzision demnach den Wert 1.0, ohne dabei auszusagen, wie viele korrekte Duplikate insgesamt gefunden wurden.

Daher wird oftmals das Verhältnis aus beiden Werten gebildet, und nach [vanReijsenbergen75] mit f-measure bezeichnet.

f-measure

Bei gegebener Referenz-Duplikatezuordnung, einer beliebigen Zuordnung A sowie der Präzision und der Vollständigkeit dieser Werte ergibt sich das f-Maß mit

$$F(A,R)=\frac{(b^2+1)\cdot P(A,R)\cdot R(A,R)}{b^2\cdot P(A,R)+R(A,R)}$$

wobei $b=1$ der Standard-Gewichtungsfaktor ist: $F_1(A,R)=\frac{2\cdot P(a,R)\cdot R(A,R)}{P(A,R)+R(A,R)}$

Das F-Maß stellt das harmonisierte Verhältnis von Präzision und Vollständigkeit dar und soll als Haupt-Meßwert genutzt werden. Um die Ausgewogenheit dennoch erkennen zu können, werden bei etwaigen Graphen oftmals dennoch Precision und Recall gegeneinander abgedruckt.

4.1.1.4 Verlinkung

Es ist zu beobachten, daß falls γ einen Vergleich von K Attributen *attr* repräsentiert, mindestens 2^K Möglichkeiten der Form $m(\gamma)$ existieren. Falls γ die Übereinstimmung von K Attributen repräsentiert, wäre zu erwarten, dass dies öfter für Duplikatetreffer M als für Fehlschläge U zutrifft. Das Verhältnis R wären dann sehr groß.

Alternativ, falls γ aus Fehlschlägen besteht, wäre das Verhältnis R sehr klein.

Falls der Zähler positiv und der Nenner 0 ist, wird eigenmächtig eine sehr große Zahl als Verhältnis zugewiesen. Die Fellegi-Sunter Verlinkungsregel L_0 nimmt dann folgende Form an:

falls $R>Obergrenze$, dann bezeichne (a,b) als Link.

falls $Untergrenze\le R\le Obergrenze$, dann bezeichne (a,b) als möglichen Link.

falls $R<Untergrenze$, dann bezeichne (a,b) als Nicht-Link.

Die Übergänge „Untergrenze“ und „Obergrenze“ sind determiniert durch die gewünschten Fehlerraten-Grenzen.

4.1.2 Abarbeitung

Die Stufe der Ähnlichkeitsfindung ist als wohl die Aufwendigste anzusehen, da in ihr die γ -Vergleichsoperationen durchgeführt werden, welche letzlich eine Einteilung in A_1 , A_2 und A_3 erlauben.
Abschließend werden in der Bewertungs-Stufe die Ergebnisse angewendet, Übereinstimmungen transitiv geschlossen und ggf. über diverse Lernalgorithmen die Gewichtungsparameter für zukünftige Harmonisierungsaufgaben verbessert.

Quellen: [Tailor]

4.2 Suchraumbestimmung

Aus Gründen der Effizienz und Skalierbarkeit soll ein Vergleich aller Tupel in jeder Relation verhindert werden. Analog zu Schritt 2 aus [„Strukturanalyse“] wird daher für jeden Datensatz der Suchraum zur Ableitung möglicher Abbildungen begrenzt.

Da in der Menge $A\times B$ im Allgemeinen $u(\gamma)\gg m(\gamma)$ gilt, werden für einen Datensatz die Vergleichsdatensätze entfernt, welche auf keinen Fall zutreffen. Eine Vielzahl von Methoden haben als Aufgabe die Rechenkosten der Namensfindung gering zu halten, indem eine erste Grobauswahl aus dem Gesamtdatenbestand isoliert wird. Sets von geordneten Paaren Q , auf denen der Fellegi-Sunter Verlinkungsalgorithmus Anwendung findet, sind daher mit „Blockierkriterien“ bzw. Sortierschlüssel belegt.

Man nehme als Beispiel ein Tupel-Set, in welchem eine Telefonnummer vorhanden ist. Diese

Telefonnummer kann in Verbindung mit einigen wenigen Zeichen des Namens zur Bestimmung von Verlinkungen nutzen. Bei anderen Paaren mögen zusätzlich die Informationen zu Straßennamen und Stadtnamen zugrunde gelegt werden.

4.2.1 Standard-Blocking

Der traditionelle Blockier-Algorithmus gruppiert Datensätze, wenn diese den selben Inhalt in einer Blockiervariable besitzen, also bspw. den identischen Eintrag für Stadt und Nachname in einem Datensatz. Blockierkriterien müssen dabei gut gewählt werden, da bei zu allgemeinen Auswahlschlüsseln die Menge näher zu untersuchender Datensätze zu hoch ist (korrekte Negative), bei einer zu engen jedoch passende Datensätze ausgesperrt bleiben könnten (falsche Positive).

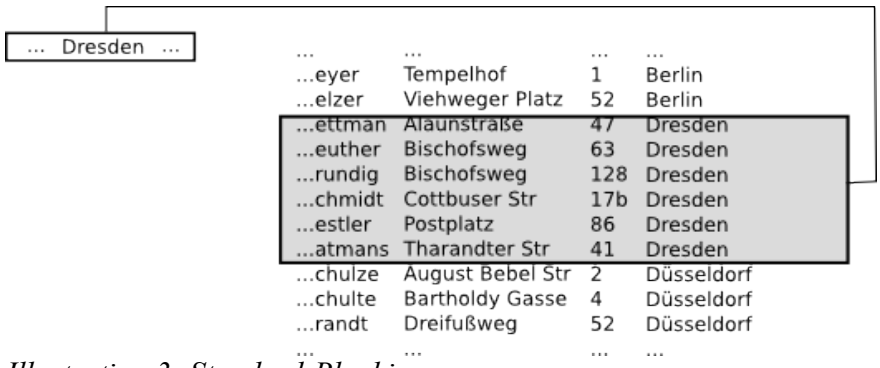


Illustration 3: Standard-Blockierung

Ebenfalls muss die Fehler-Charakteristik des gewählten Schlüssels bekannt und ein Fehlerauftritt minimal sein. Durch die Verwendung multipler Blockierschlüssel kann der Fehler in einem der Schlüssel abgemildert werden. Die Berechnungskomplexität ist pro gewählten Schlüssel,

$O(h(n)+n^2/b)$, wobei $h(n)=n\log n$ bei sortierter Blocking-Menge, $h(n)=n$ bei blocking durch hashing. Ergebnisse sind allerdings nur schwer abzuschätzen.

4.2.2 Sorted-Neighbourhood / sliding window

Eine Weiterentwicklung stellt die Sorted-Neighbourhood- oder Fenster- Strategie („windowing strategy) nach [HS95] dar. Sie sortiert einen Datenbestand nach dem Schlüssel („Key“)-Attribut mit dem größten Unterschied und vergleicht alle Records innerhalb eines fliessenden Fensters („sliding Window“) in der aussortierten Ordnung . Die Effektivität dieses Ansatzes hängt stark von der Qualität des gewählten Schlüssels ab.

„Poor chosen keys will result in a poor quality merge“

Die „Sorted-Neighbourhood“-Methode kann in 3 Phasen zusammengefasst werden:

1. **Erstellen der Schlüssel**: Errechne einen Schlüssel für jeden Datensatz in der Liste durch extrahieren relevanter Felder oder Feldbestandteilt
2. **Sortierung der Daten**: Sortiere die Datensätze in der Datenliste unter Nutzung des Schlüssels aus Schritt 1
3. **Vereine**: Bewege ein Fenster mit konstanter Größe durch die sequentielle Liste der Datensätze und limitiere den Vergleich auf passende Werte auf solche innerhalb des Fensters. Falls die Fenstergröße w Datensätze umfasst, so wird jeder neuer Datensatz mit den vorhergehenden w-1 Sätzen verglichen, um „passende Datensätze“ zu finden. Der erste Datensatz im Fenster slided dabei aus dem Fensterbereich.

Illustration 4: Funktionsweise Sliding-Window

Bei sehr großen Datenmengen empfiehlt sich ein vorheriges Clustern des Datenbestandes. Dabei wird der Datenbestand in Sequenzen durchsucht, wobei ein n-attributiger Schlüssel (Bsp. Nachname+1.Buchstabe des Vornamens) extrahiert und in den n-dimensionalen Clusterraum abgebildet wird.

Da auch dieses gewählte Attribut einen Fehler besitzen kann, ist ein einzelnes Schlüsselattribut zu wenig. Um demnach Fehler im Hauptschlüssel auszuschließen, schlägt [HS95] folgende Methoden vor:

1. Durch die Vergrößerung des sliding-windows werden mehr Datensätze mit eingeschlossen.
2. ein mehrmaliges Ausführen mit unterschiedlichen Schlüsseln und relativ kleinem Fenster gibt, nach einzelnem Abgleich der Ergebniss-Gruppen und transitivem Schließen (nur bei geringer Fenstergröße empfehlenswert) die besseren Ergebnisse.

Als Grenzwerte sind hier die Schlüsselanzahl j sowie die Fenstergröße w anzugeben. Damit errechnet sich für n Datensätze eine Komplexität von $O(\sum_{i=0}^{j}(n\log n+wn))$, wobei $O(n\log n)$ für die Sortierung benötigt wird.

k-nearest - Methode

4.2.3 Fuzzy Blocking / Bigramm-Indexing

Diese Methode basiert auf Bigramme (in [SprachSynth06] auch Diphone oder Halblaute genannt). Die Grundidee dabei besteht darin, dass nach der Indexerstellung die Werte der Blockiervariablen in eine Liste von Bigramme (Diphone, Halblaute) konvertiert werden, welche alphabetisch sortiert und von Duplikaten bereinigt werden. Nun werden Sub-listen erstellt, für welche ein benutzerdefinierte Untergrenze (eine Zahl zw. 0.0 und 1.0) für alle möglichen Kombinationen gegeben wird.

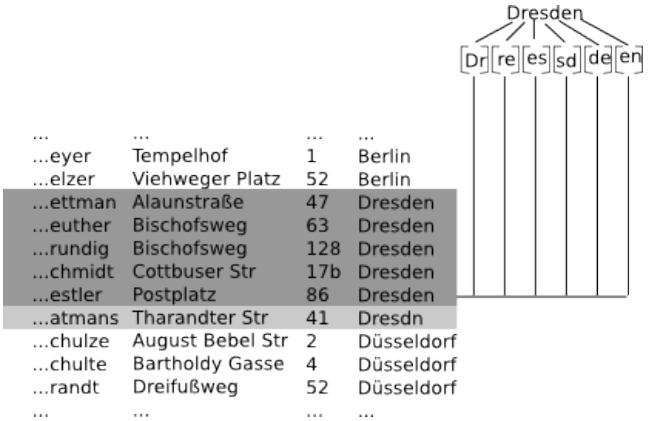


Illustration 5: Bigramm-Blockierung

Die resultierende Bigram-Subliste wird in einen invertierten Index eingefügt, z.B. Datensatz-

Nummern in dem Block werden eingefügt in das Lexikon für jede Bigram-Subliste.

Beispiel: Der String 'peter' würde die Bigram-Liste ['pe','et','te','er'] mit vier Elementen ergeben. Es würde sortiert zu ['er','et','pe','te'], und bei einer Untergrenze von 0.8 ergebe dies 4 * 0.8 = 3.2, gerundet 3, was bedeutet, dass alle Kombinationen der Länge 3 berrechnet werden. Für das gegebene Beispiel

['et','pe','te']

['er','pe','te']

['er','et','te']

['er','et','pe']

Damit wird die korrespondierende Datensatz-Nummer mit den Schlüsseln 'etpete', 'erpete', 'erette' und 'eretpe' in die invertierten Index-Blöcke eingefügt.

Je geringer die Untergrenze, desto kürzer die Sub-Liste, aber umso mehr Sub-Listen gibt es pro Feldwert, was zu mehr (kleineren) Blöcken im invertierten Index führt.

Als Grenzwerte kann hier die Anzahl der übereinstimmenden Bigramme T_1 sowie die allgemeine Bigramm-Länge T_2 identifiziert werden. Daraus ergibt sich, wie beim Standard-Blocking, die Komplexität von $O(\frac{n^2}{b})$ [tailor02], wobei b die Anzahl der zu vergleichenden Blöcke darstellt.

4.2.4 Canopy-Methode

Alternativ dazu liefert die in [Nigam2000] vorgestellte „Canopies Methode“ eine schnelle und generelle Zeichenketten-Ähnlichkeit-Metrik.

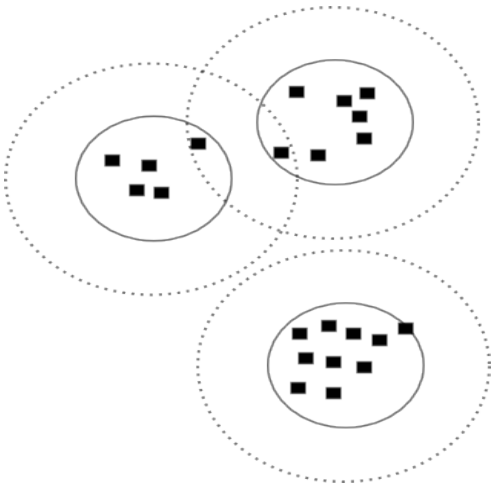


Illustration 6: Canopy-Blockierung

Diese besteht aus 2 Schritten, wobei der 1. Schritt einen Blockieralgorithmus in der Form darstellt, daß er über eine billige Distanzmessung die Entitäten in n>= 1 sich überlappende, sog. Canopy (engl. „Abdeckhaube“) einteilt. Dazu wird aus der Liste der Datensätze ein beliebiger Eintrag genommen und mit allen anderen Einträgen der Liste verglichen (bspw. nach der Anzahl identischer Attribute). Alle Datensätze innerhalb der Distanz T_1 werden dabei in ein Canopy gegeben. Alle Datensätze innerhalb der Distanz T_2 werden von der Liste gelöscht. Dies wird wiederholt, bis die Liste leer ist.

Dadurch werden Nachteile des Grobblockings bzw. Sliding-Windows ausgeräumt, wie ein mögliches Paar außerhalb der Fenstergröße.

Die „Term-frequency-inverse data frequency“ (TF-IDF) Cosinus Ähnlichkeit ist eine billige Distanzmessungen, um sich überlappende Datensatz-Cluster zu erstellen. Es basiert auf der

Auftrittshäufigkeit einzelner Terme und wird im Kapitel [TDIDF] näher behandelt.

Im 2. Schritt werde daraufhin mit einer teureren Distanzmessung nur Punkt-Paare verglichen, welche sich im selben Canopy befinden.

Bei diesem Ansatz können 3 Parameter gesetzt werden: während Stufe 1 der Loose-Grenzwert (T_1) und der Tight-Grenzwert (T_2) . Bei Verwendung des GAC (siehe unten), 3. Parameter der Stopp-Parameter, bei dessen erreichen keine weitere Iteration ausgeführt wird.

Die Berechnungskomplexität kann bei Nutzung der invertierten Liste für Schritt 2 wie folgt dargestellt werden

w .... Anzahl der Vergleichsdatensätze

V .... die möglichen Werte der Attribute, |V| Anzahl der Canopies

n .... anzahl aller Entitäten/Datensätze

$$O(n\frac{w^2}{|V|})$$

Loose- und Tight sollten so gewählt werden, daß n gering und |V| möglichst groß ist, um den Umfang der Berechnung zu verringern. bei zu großen Werten können typographische Fehler nicht mehr erkannt werden.

weitere Verwendung von Canopy: GreedyAgglomerativeClustering und SVM

4.2.5 Zusammenfassung

Vergleichsmessungen, durchgeführt in [Kdd03]. Die Ergebniss-Anzahl bei Standard-Blocking können großen Schwankungen unterliegen und nehmen mit zunehmenden Blockierschlüsseln rapide ab. Kleinere Blöcke bedeuten dabei zwar weniger Vergleiche, schliessen jedoch ebenso mehr korrekte Paare aus.

Sorted-Neighbourhood-Methoden verhindern diese Extreme in der Performance des Standard Blockings und erlauben ein vorhersehbares Verhalten durch Erhöhung der Fenstergröße w. Durch größere Fenster erhöht sich die Paar-Vollständigkeit in den Ergebnissen, doch das Reduktionsverhältnis sinkt.

Mit gut gewählten Parametern sind jedoch sowohl Bigramm- als auch Canopies-Clustering signifikant performanter. Bei Ersterem wird eine Bigram-Vergleichsanzahl von 3 - 4 empfohlen, der Schwellwert für Canopy (siehe Kapitel 3.2.3.3) sollte um 1.5 liegen.

	Komplexität	korrekte Negative	falsche Positive
Standard-Blocking	O (n)	++	++
Sliding Window	O(nlog n + wn).	+	+
Bigramme	$O(\frac{n^2}{b})$	-	+
Canopy	$O(n\frac{w^2}{\|V\|})$	-	+

4.3 frequenzbasierte Gewichtung

Die Auftrittsfrequenz eines Terms in einer gegebenen Datensammlung gibt einen Hinweis auf die Bedeutung dieses Terms für dieselbige. In diesem Abschitt sollen Methoden vorgestellt werden, mit denen einzelne Terme in einer Datensammlung/eines Attributes oder einer Anfrage gewichtet, also Gewichtsvektoren für Datensammlungen und Anfragen bestimmt werden können. Sie geben Auskunft darüber, in wieweit diese Ähnlichkeit Auswirkungen für das Gesamtergebnis besitzt.

Falls für eine Duplikatefindung keine Trainingsdaten in der Form „Duplikate“ / „kein Duplikat“ vorliegen, so kann die Namensübereinstimmung über bedingte Wahrscheinlichkeiten mit Hilfe frequenzbasierten Schätzung ermittelt werden.

Man sieht hierfür eine Datensammlung als ein festes Vokabular von Termen. Die Auftrittshäufigkeit einzelner Terme gibt diesen dabei eine Gewichtung welche für die Inhaltsbeschreibung nutzbar ist. Die Reihenfolge der Terme im Datensatz wird jedoch vernachlässigt.

Es sei demnach $D=\{a_1, \ldots, a_n\}$ eine Menge von Datensätzen einer Entitätsklasse sowie

$c=\{attr_1, \ldots, attr_n\}$ eine Menge von Attributen $attr_j: D \to R$ auf diesen Datensätzen.

Für jeden Datensatz $a_i \in D$ sei zu jedem Attribut $attr_k \in c$ ein Gewicht $w_{i,k} \in R$ gegeben, was manuell oder über die im Kapitel [TF-IDF] vorgestellte Methode errechnet werden kann.

Diese Gewichte des Datensatzes a_i lassen sich zu einem Vektor $w_i=(w_{i,1}, \ldots, w_{i,n}) \in R^n$ zusammenfassen. Dieser Vektor beschreibt das Dokment im Vektorraummodell: Er ist seine Repräsentation und wird Dokumentenvektor genannt.

Anfragen „Queries“ werden durch einen Vektor $q \in R^n$ repräsentiert und bei der Anfrage durch eine Menge gewichteter Terme dargestellt. Eine Ähnlichkeitsfunktion $s: R^n \times R^n \to R$ definiere für je zwei Vektoren $x, y \in R^n$ einen reellen Ähnlichkeitswert $s(x, y)$.

Beim Beispiel TF-IDF ist diese Ähnlichkeitsfunktion beispielsweise das Skalarprodukt zwischen ihren Vektorraum-Darstellungen x und y, dem Cosinus des Winkels zwischen beiden:

$$s(x, y)=\frac{\{x \cdot y\}}{\|x\|\|y\|}=\frac{\sum_{i=1}^{d} x_i y_i}{\|x\|\|y\|}$$

Der Ähnlichkeitswert gibt an, wie viele Terme sowohl im Anfrage-Term, als auch im Datensatz vorkommen. Nach diesem Ähnlichkeitswert können die Datensätze in eine Rangfolge gebracht werden. Dabei ist zu erwarten, daß Datensätze, welche viele Terme aus der Anfrage enthalten, einen oberen Rang besitzen und Datensätze mit wenigen Anfragetermen am unteren Ende zu finden sind.

Über eine damit verknüpfte Gewichtungsfunktion ist es möglich, die auftretenden Attribute nochmals feiner abzustimmen, so z.B. über Stoppwort-Listen oder Gewichtung einzelner Attribute.

	Der	Ferrari	F430	Spider	ist	grün
Stoppwort	x	-	-	-	x	-
Gewichtung	0,0	0,7	1,0	0,8	0,0	0,4

Illustration 7: Beispiel Stoppwort-Liste, Termgewichtung

Praktisch bedeutet dies, daß beispielsweise eine Übereinstimmung zweier seltener Bezeichnungen eine höhere Priorität geniessen wird, als eine Übereinstimmung zweier allgemein gängiger Bezeichnungen.

4.3.1 Jaccard

Für 2 Wortmengen S und T is die Jaccard-Distanz, als Tokenbasierte Distanz definiert als $(|S \cap T|)/(|S \cup T|)$

4.3.2 Term-Frequenz

In einer überschaubaren Term-Menge ist eine manuelle Gewichtung der Terme der Datensatz-Vektoren von humanen Indexierern denkbar, z.B. in Form einer Stoppwort-Liste.

Dabei werden bestimmte Terme wegen ihres allgemeinen Auftretens und fehlenden semantischen Inhalts ignoriert bzw. bestimmte Wortteile über manuell erstellte Auswahllisten mit einer Gewichtung belegt.

Per Hand entstehen dadurch neben hohen Arbeits-, Zeit- und Kostenaufwand ebenfalls Inkonsistenzen durch die subjektive Meinung und Tagesform der Personen. Daher sind halb-automatisch bzw. vollautomatisch Methoden zu nutzen.

Termgewichtungen sind unterscheidbar in globale/kontextunabhängige und lokale/kontextabhängige Faktoren. Ein lokales Kriterium für eine Gewichtung wäre dabei beispielsweise die Häufigkeit des Terms innerhalb eines Datensatzes.

Unter den globalen oder kontextabhängigen Gewichtungsfaktoren tritt die Häufigkeit eines Terms in der Sprache bzw. in einer Datensatzsammlung häufig auf. Zipf [Ferber05] beschreibt eine Relation

$h(w) \sim \frac{c}{r(w)}$ mit

C - Textkorpus,

$W(C)$ die Menge der Wörter in C und

$h(w)$ deren Häufigkeit im Textkorpus, sowie

$r(w)$ bezeichne den Rangplatz von w in $W(C)$,

unter der Voraussetzung der Sortierung der Wörter nach abfallender Häufigkeit.

Daraus folgt, daß eine kleine Anzahl von häufigen Wortern einen großen Anteil der Datensammlung abdeckt und die großen Anzahl der seltenen Wörter nur einen kleinen Anteil des Textes ausmacht.

Häufige sind Wörter in jedem Text zu erwarten, während sehr seltene Wörter nicht in allen relevanten Texten vorkommen. Daher sind Terme mittlerer Häufigkeit optimal, welche zwar häufig genug sind, um genügend relevante Inhalte abzudecken, aber auch signifikant genug, um nichtrelevante Texte auszuschließen.

Anstelle der Häufigkeit von Termen kann hierbei die Dokumenten-/ oder Datensatzhäufigkeit (document frequency) verwendet werden, die Anzahl der Datensätze/Dokumente, in denen ein Term auftritt. Bei einer zufälligen Verteilung eines Wortes in einem Korpus von Datensätzen werden durch den Übergang von der Häufigkeit zur Dokumentenhäufigkeit die Häufigkeitsunterschiede besonders für häufige Terme verringert: Bei der Bestimmung der Dokumentennhäufigkeit spielt es keine Rolle, ob ein Term oft in einem Dokument vorkommt oder nur einmal.

4.3.3 Inverse Dokument-Frequenz

Während eine Stoppwortliste beim boolschen Retrieval also eine harte Häufigkeitsschranke für den Ausschluss setzt, läßt sich der Einfluss der Häufigkeit mit der Möglichkeit, Terme zu gewichten, differenzierter modellieren. Meistens wird dazu eine Form der inversen (oder auch invertierten) Dokumenthäufigkeit (inverted document frequency – idf) verwendet:

$$w_{i,j} = idf(j) = \frac{1}{d(j)} \quad ,$$

wobei wiederum $D=(a_{1,\dots},a_n)$ die Menge der Dokumente und $T=(t_{1,\dots},t_n)$ die der Terme und $a(j)$ die Anzahl der Dokumente, in denen Term t_j vorkommt. In der Praxis wird oft die modifizierte Form $w_{i,j}=\ln(\frac{m}{d(j)})$ oder $w_{i,j}=\ln(\frac{m-d(j)}{d(j)})$ verwendet, wobei der natürliche Logarithmus ln hierbei große Werte dämpft, also in diesen Formeln die Gewichte seltener Terme wieder abschwächt.

4.3.4 TF-IDF

Auch bei kontextabhängigen Einflussfaktoren wird vor allem die Häufigkeit eines Terms in einem Dokument zur Berechnung von Termgewichten herangezogen (TF). Dabei wird im Allgemeinen davon ausgegangen, dass Terme, die häufig in einem Dokument auftreten, für die inhaltliche Beschreibung wichtiger sind als solche, die nur selten auftreten. Im einfachsten Fall kann die Häufigkeit eines Terms in einem Dokument direkt in der Form $w_{i,j}=h(i,j)$ bei $h(i,j)$ als Bezeichnung für die Häufigkeit von Term t_j im Datensatz a_j. Andere Formeln beschränken die Gewichte auf ein Intervall und dämpfen den Einfluss sehr häufiger Terme, wie z.B. die Formel

$$w_{i,j}=\frac{h(i,j)}{a+h(i,j)}$$

Außerdem kann man die Häufigkeit eines Terms zu der des häufigsten im Text in Relation setzen und damit die unterschiedlichen Gesamttext-längen ausschalten.

$w_{i,j}=K+(1-K)\frac{h(i,j)}{max_{l\in\{1,\dots,n\}}h(i,l)}$ wobei K den Parameter bezeichnet, mit dem bestimmt werden kann, wie groß der Einfluss der Gewichtung sein soll. K=0 verwendet nur die Häufigkeitsgewichtung, bei K=1 spielt sie keine Rolle und es wird für alle Terme das konstante Gewicht 1 vergeben.

Häufig werden lokale und globale Gewichtungen zu Formeln vom Typ $w_{i,j}=\frac{h(i,j)}{d(j)}$ verknüpft, indem die Termhäufigkeit (TF) mit der invertierten Dokumentenhäufigkeit (IDF) multipliziert wird. Gewichtsformeln von diesem Typ werden auch als TF-IDF-Gewichtung bezeichnet (term frequenz – inverted document frequency).

Hauptquellen: [Ferber03], [Inte03]

4.4 statische Zeichenketten-Vergleiche

Der Annäherungs-Vergleich ist ein wichtiges Feature für die erfolgreiche Gewichtsermittlung wenn Zeichenketten von Namen und Adressen verglichen werden sollen. Anstelle einer einfachen Ja-Nein-Gewichtung ganzer Terme oder x Anfangsbuchstaben, erlauben Annäherungsvergleiche partielle Übereinstimmungen falls Strings nicht identisch aber sehr ähnlich sind, was durch typographische, geographisch-bedingte oder andere Fehler auftreten kann.

Eine Vielzahl von Annäherungsalgorithmen sind entwickelt worden, sowohl im statischen DuplikateFinden als auch in den Informatik und natürlichen Sprachverarbeitungen entwickelnden Communities. Abhängig von der Fehlerursache können diese unterteilt werden in

- Verfahren zur phonetischen Distanz
  Sie wandeln die Attributinhalte in Lautsprache um und vergleichen sie auf dieser Basis (Bsp.: Soundex)
- Editierdistanz – Verfahren
  Diese messen den Abstand der Buchstaben in zum Vergleich herangezogenen Wörtern (Bsp. Jaro, Jaro-Winkler, Levenshtein)
- typewrite Distanz – Verfahren
  welche die Dimension der Eingabegeräte betrachten und beispielsweise Tastenabstände benachbarter Tasten betrachten und damit die Wortähnlichkeit ermitteln. (Bsp. Needleman-Wunsch, Smith-Waterman)
- Swapping - Distanz
  Sie betrachten die Möglichkeit, daß bei der Eingabe Terme an das falsche Attribut gekoppelt wurden (Wortvertauschung)
- kombinierende Verfahren (Bsp. Kompression)

Sie unterscheiden sich hauptsächlich in Abarbeitungsgeschwindigkeit und Genauigkeit Ihrer Ergebnisse.

Es ist mitunter daher empfehlenswert, hier ebenfalls nochmals Vorselektionen durch einfacherere Methoden durchzuführen, wir beispielsweise eine BagDistanz-Annäherung vor der Berechnung einer Levenshtein-Distanz.

4.4.1 Hamming-Distanz

Die Hamming-Distanz wird hauptsächlich für nummerische Werte wie Geburtsdaten oder Postleitzahlen genutzt. Sie zählt die Anzahl von Fehlstellen zwischen 2 Zahlen. So liegt die Hammingdistanz zwischen den Postleitzahlen „01097“ und „09017“ bei 2, da 2 Fehlstellen existieren.

Die Berechnungskomplexität ist somit linear zur Zeichenanzahl des kürzeren Terms.

4.4.2 phonetische Suche

Erkenntnis aus der Phonetik und Phonologie werden genutzt, um die klangliche Repräsentation eines Wortes zu ermitteln. Um diese phonet. Repräsentation zu finden, werden Wörter in „Phoneme“ zerlegt. Je nach gewünschter Unschärfe der Sucher werden ähnliche Laute zusammengefasst und damit in die klangliche Dimension abgebildet. Dort können sie ebenfalls durch im Anschluß erläuterte Methoden equivalent untersucht werden (bswp. Lautersetzungen, -auslassungen, Levenshtein-Analyse). Es ist zu beachten, daß die Art der Abbildung jeweils sprachenabhängig ist.

4.4.2.1 Soundex

Soundex ([Soundex00]) ist ein phonetischer Algorithmus zur Indizierung von Wörtern und Phrasen in englischer Sprache, welcher ebenso für den deutschen Sprachraum gute Ergebnisse liefert. Er kodiert gleichklingende Wörter zu identischen Zeichenfolgen, welche aus dem Anfangsbuchstaben sowie 3 Ziffern, die aus dem darauffolgenden Konsonanten des Wortes generiert werden. Hierbei besitzen

ähnliche Laute den gleichen Ziffern-Code.

1	B, F, P, V
2	C, G, J, K, Q, S, X, Z
3	D, T
4	L
5	M, N
6	R

Table 1: Soundex-Zuordnung

Hierbei werden doppelt vokommende Buchstaben wie Einzelbuchstaben betrachtet. Aufeinanderfolgende Buchstaben mit gleicher Soundex-Code werden wie gleiche Buchstaben behandelt, „Dackelmann“ würde beispielsweise mit „D-245“ beschrieben. Namenszusätze wie beispielsweise „von“ können ignoriert werden.

Werden 2 Konsonanten mit gleichem Soundex-Code durch Vokale getrennt, so wird der rechte Konsonant nicht verworfen, es sei denn, das trennende Zeichen ist ein H oder ein W.

Wie gezeigt, handelt es sich hierbei um eine schnelle, grobe Einteilung, welche ebenso als Grobauswahl-Filter genutzt werden könnte.

4.4.3 Q-Gramme

Qgramme stellen eine Erweiterung der in Kapitel [Suchraumbestimmung] behandelten Bigramme dar. Hierbei wird über eine Zeichenketten σ ein Sliding-Windows (Gleitfenster) mit der Länge q > 2 gezogen. Da q-gramme am Beginn und am Ende der Zeichenkette σ weniger als $|q|$ Zeichen besitzen können, werden die Sonderzeichen # als Prefix mit q -1 auftreten und $ als Suffix mit q-1 Auftreten eingeführt.

Zusätzlich können die Auftrittspositionen der Q-Gramme ebenfalls gespeichert werden, man nennt diese auch „positionell bestimmte Qgramme“. Dadurch ist ein feinerer Abgleich möglich. Die Auftrittswahrscheinlichkeit ist, wie auch bei der folgenden BagDistanz, stets größer als die Edit-Distanz (Nachweis siehe [WoWarDasGleichNochmal?]).

4.4.4 Jaro

Jaro [Winkler91] führte eine String-Vergleichs Messung ein, welche Einträgen ein teilweises Abkommen zwischen 2 Strings gibt. Der String-Vergleich betrifft die Länge der Zeichenketten und ist für die Art der typischerweise durch Personen verursachte Fehler angepasst. Es wird beim Anpassen der genauen Abkommensgewichts genutzt, wenn 2 Strings nicht auf Zeichenbasis zusammenpassen. Genauer gesagt, wenn die Anzahl der gemeinsamen Zeichen im Zeichenkettenpaar c > 0, so berechnet sich der Jaro String-Vergleich nach:

$$\Phi = W_1 \cdot c/d + W_2 \cdot c/r + W_t \cdot (c-\tau)/c$$

wobei

W_1 = Gewichtung der Zeichen im ersten der beiden Datensätze

W_2 = Gewichtung der Zeichen im zweiten der beiden Datensätze

W_t = Gewichtung der Transposition

d = Länge der Zeichenkette im ersten Datensatz

r = Länge der Zeichenkette im zweiten Datensatz

τ = Anzahl der Transpositionen des Zeichens

c = Anzahl der gemeinsamen Zeichen im Zeichenketten-Paar

Falls $c=0$, dann $\Phi=0$.

Zwei Zeichen werden als „gemeinsam“ angesehen, wenn Sie nicht weiter als $(m/2-1)$ von einander entfernt sind. Dabei ist $m=max(d/r)$. Zeichen, welche Zeichenketten „gemeinsam“ besitzen, gelten als „zugeordnet“ (Umsetzung Sheffield nutzt „min“), die übrigen Zeichen als „unzugeordnet“. Jede Zeichenkette hat die selbe Anzahl zugeordneter Zeichen.

Die Nummer von Transpositionen wird wie folgt berechnet:

Das erste zugeordnete Zeichen in einem String wird mit dem ersten Zeichen der zweiten Zeichenkette verglichen. Falls die Zeichen nicht identisch sind, ist dies bereits eine halbe Transposition. Dann wird das zweite Zeichen der ersten Zeichenkette mit dem zweiten zugeordneten Zeichen der zweiten Zeichenkette verglichen, etc. Die Anzahl der nicht-übereinstimmenden Zeichen wird durch 2 geteilt um die Anzahl der Transpositionen zu erhalten.

Falls 2 Terme Zeichen-für-Zeichen-Basis übereinstimmen, dann ergibt sich der Jaro-Vergleicher Φ aus $W_1+W_2+W_t$, was gleichzeitig das Maximum ist, welches Φ erreichen kann. Das Minimum ist 0 und tritt auf, wenn beide Zeichenketten keine Zeichen gemeinsam haben.

Als Anfangsgewichtungen werden oftmals W_1 , W_1 , W_1 willkürlich auf $1/3$ gesetzt. Die neue Zeichen-Comparator Metrik verändert grundsätzlich den Basis-Zeichen-Comparator je nachdem ob die ersten paar Zeichen in den verglichenen Zeichenketten übereinstimmen.

--> Das gehört schon zu JaroWinkler <--

Speziell für i = 1, 2, 3, 4,

$\Phi_n=\Phi+i\cdot 0.1\cdot(1-\Phi)$, falls die ersten i Zeichen übereinstimmen.

--> Das nicht mehr –- Veränderung der Gewichtungen über die Zeit <--

Gegen die Annahme, daß die selbe Anpassungsprozedur für verschiedene Felder, wie Vorname, Nachname und Hausnummer, funktioniert, werden Prozeduren für die Modellierung der Gewichtungsanpassung als eine stückweise lineare Funktion entwickelt.

Die Prozeduren benötigen eine repräsentative Menge von Paren, von denen die Identität bekannt ist. Die neu angepasste Gewichtung w_{na} nehmen dabei die Form

Die Konstanten a_1 , a_2 , b_1 und b_2 hängen von den spezifischen Zeichenketten-Typen (wie Vorname) ab, zu denen die Gewichtungsanpassung zugeordnet wird. Generell gilt $a_1<a_2$. Beispielkonstanten sind gegeben in Tabelle 3. --> Wo hab ich die Tabelle dazu? <--

4.4.5 Winkler-Jaro

William E. Winkler nutzt die Technik Jaros, geht allerdings davon aus, dass typographische Fehler häufiger am Ende von Wörtern vorkommen, und daher der Beginn von Wörtern schwerer zur Identifizierung gewichtet wird, meist die ersten 4 Zeichen. Die teilweise Übereinstimmungsgewichtung wird daher erhöht, falls der Beginn zweier Zeichensätze gleich ist.

$$Jaro-Winkler(s,t)=Jaro(s,t)+(P'/10)\cdot(1-Jaro(s,t))$$

4.4.6 BagDistanz

Die „Bag Distance“ ist eine einfache Distanz-Messung, welche stets eine Distanz kleiner oder gleich der Edit Distanz angibt. Sie wird daher auch oft als letzte Vorauswahl-Instanz für mit Levenshtein zu testende Terme genutzt, da sie für 2 gegebene Zeichenketten s_1 und s_2 in

$O(len(s_1)+len(s_2))$ berechnet werden kann. Die Bag-Distanz wird daraufhin durch das Maximunm der Zeichenkette geteilt und ergibt so einen Wert zwischen 0.0 und 1.0. Daher wird die Bag-Distanz Ähnlichkeit immer größer sein als die der Edit Distanz.

4.4.7 Levenshtein

Die grundlegende Methode für die Festlegung des Ähnlichkeitsgrades zwischen zwei Einzelwörter basiert auf dem „Levenshtein"- oder "Edit Distance"-Algorithmus. Dabei wird versucht, den minimale Weg für die Umformung einer Zeichenketten durch Löschen oder Ersetzen von Zeichen zu finden, sodass die zwei Zeichenketten angeglichen werden können. Die einzelnen Schritte der Umformung werden ständig bewertet und ergeben nach der Umformung den Ähnlichkeitsgrad, der bezogen auf der Länge des längeren Wortes in Prozent ausgedrückt werden kann.

Im allgemeinen wird diese Umformung durch einen Dynamic-Programming-Ansatz gelöst. Eine Matrix wird initialisiert, die für jede (m, N) - Zelle die „Edit-Distanz" zwischen dem m-Buchstabenpräfix des einen Wortes und den n-Präfixes des zweiten Wortes enthält. Wird dies Tabelle beispielsweise von der oberen linken Ecke zur unteren rechten Ecke gefüllt, so kostet jeder Sprung hochizontal oder virtuell einen virtuellen Betrag, da es einem Editieren, Einfügen oder Löschen eines Zeichens entspricht. Die Zahl am rechten unteren Ende dieses Beispiels entspricht somit dem Levenshtein-Abstand zwischen beiden Wörtern.

Gleiche Zeichen ergeben auf der Zeichenebene eine Bewertung von 1.0. Die Punkte für Strafe und Score werden über Experten vorgegeben oder anhand vielen Beispielen eingestellt.

4.4.8 Needleman-Wunsch / Smith-Waterman

Als erste natürliche Erweiterung führten Needleman und Wunsch zusätzliche Parameter ein. So können die Einzelkosten für Zeichenersetzung, -einfügungen und -löschungen individuell definiert werden. Weiterhin ist die Strafpunktzahl für Löcher (sog. „gaps") einzeln definiert. Dies ist von Bedeutung, wenn in einer Zeichenkette ein Begriff abgekürzt, in der zweiten jedoch ausgeschrieben wurde. Der Needleman-Wunsch-Algorithmus findet im Bereich der Bioinformatik zur Identifikation von Proteinen rege Anwendung.

Bisher ist es nicht möglich, die längste übereinstimmende Teilsequenz beider Zeichenketten zu finden und anzupassen. Um dieses Problem zu beheben wurden von T.Smith und M. Waterman entsprechende Modifikationen am Needleman-Wunsch-Algorithmus eingefügt, die genau dies ermöglichen. Der modifizierte Algorithmus wird Smith-Waterman-Algorithmus genannt. [Vldb01]

Die erweiterte EditDistanz besitzt zusätzlich die Möglichkeit, Substitutionen zu nutzen und Blöcke fortlaufender Zeichen zu jeweils bestimmten Strafpunkten zu bewegen. Diese Blöcke müssen in beiden Zeichenketten identisch sein und $0 \leqslant extendedEditDistance(\sigma_{1,}\sigma_2) \leqslant max(|\sigma_1|,|\sigma_2|)$.

Die Laufzeitkomplexität beträgt in allen Fällen O(n * m).

4.4.9 Substring

Giorgos Stoilos beschreibt in [Stoilos05] eine weitere Variante, welche auf der Substring-Metrik beruht und für Bezeichnungs-Übereinstimmungen, wie es beispielsweis beim Ontology-Alignment (Kapitel 3.3) auftritt, gute Erfolge verspricht.

$$Sim(s1,s2) = Comm(s1,s2) - Diff(s1,s2) + winkler(s1,s2)$$

Grundsätzlich werden hier beide Zeichenketten iterativ nach den maximal übereinstimmenden Teil-Zeichenketten durchsucht und aus den Strings entfernt. Konkanetierten Zeichenketten können so sehr gut erkannt werden.

$$Comm(s1, s2) = \frac{2 * \sum_i length(maxComSubString_i)}{length(s_1) + length(s_2)}$$

Die nicht erkannten Zeichenreste werden im Differenz-Term über die Hamacher-Funktion mit der Originalstringlänge ins Verhältnis gesetzt.

$$Diff(s_1, s_2) = uLen_{s_1} \times uLen_{s_2} * p + (1 - p) * (uLen_{s_1} + uLen_{s_2} - uLen_{s_1} * uLen_{s_2})$$

Um zu verhindern, daß am Ende jedes einzelne, nicht zugeordnete Zeichen als Substring erkannt wird und damit einer BagDistanz equivalent wäre, ist die SubString-Länge auf q>=2 zu setzen.

4.4.10 Kompression

Die Nutzung von Kompressionsalgorithmen, um Ähnlichkeiten zwischen Strings zu berechnen, basiert auf deren Informationsgehalt. Die 1965 von A.N. Kolmogorov propagierte Kolmogorov-Komplexität quantifiziert die Zufälligkeit von Zeichenketten und anderen Objekten in einer objektiven und absoluten Art und Weise.

Sie wird für verschiedene Data-Mining Algorithmen, inklusive Clustering und time-seriers-Analysen genutzt, unter anderem wegen ihrer annähernden Parameterfreiheit genutzt. [Keogh04] geht darauf näher ein.

Eine Ähnlichkeitsmessung zwischen zwei Zeichen s_1 und s_2 kann wie folgt berechnet werden:

Sei $K(s_1)$ die Länge des kürzest möglichen Programms, welches die Zeichenkette s_1 an einem Universal-PC (Turing-Maschine) erstellen kann. Die Länge des kürzesten Programms auf s_2 unter der Voraussetzung von s_1 als unterstützende Eingabe wird mit $K(s_2|s_1)$ angegeben.

Die Kolmogorov-Komplexität ist die unterste Grenze aller Informationsmessungs-Algorithmen. Da sie nicht direkt berechnet werden kann, wird diese durch Kompressions.Algorithmen angenähert. Genau genommen ist $K(s_1)$ die beste Kompression, welche auf einen Text s_1 angewendet werden kann.
Mit einem gegebenen Algorithmus bezeichnet man $c(s_1)$ als die Größe der gepackten Zeichenkette s_1 .

$$C(s_1) = len(compress(s_1)) \approx K(s_1)$$
$$C(s_2) = len(compress(s_2)) \approx K(s_2)$$

$C(s_1|s_2)$ erhalten wir als Kompression, wenn der Lompressor zunächst mit s_2 trainiert wird und darauf s_1 packt. Falls der Kompressor also (wie LZW) auf einer textuellen Substitution basiert, würde s_1 unter Nutzung des Wörterbuchs von s_2 gepackt.

Verallgemeinert man dies nun auf die gepackte Version einer Zeichenkette, so ergibt sich

$$C(s_{12}) = \frac{C(s_1|s_2) + C(s_2|s_1)}{2} = \frac{len(compress(s_1 + s_2)) + len(compress(s_2 + s_2))}{2}$$

Zu einer Ähnlichkeit normalisiert kann man diese Formel nutzen.

$$sim(s_1, s_2) = 1.0 - \frac{C(s_{12}) - min(C(s_1), C(s_2))}{max(C(s_1), C(s_2))}$$

wobei $compress()$ die Compressionsfunktion darstellt.

4.4.11 Zusammenfassung

Wie bereits in der Einleitung angesprochen, variieren die Mess-Ergebnisse der einzelnen vorgestellten Algorithmen stark, abhängig von der gewählten Einsatz-Domäne.

Für Zahlenwerte genügt eine einfache Hamming-Distance komplett.

Bei Namen ist der Soundex eine schnelle Methode zur Vorsortierung, da Vokale in Namen eine eher untergeordnete Rolle spielen und aufgrund der Verkürzung auf bestimmte Konsonanten eine gute Vorauswahl erreicht wird.

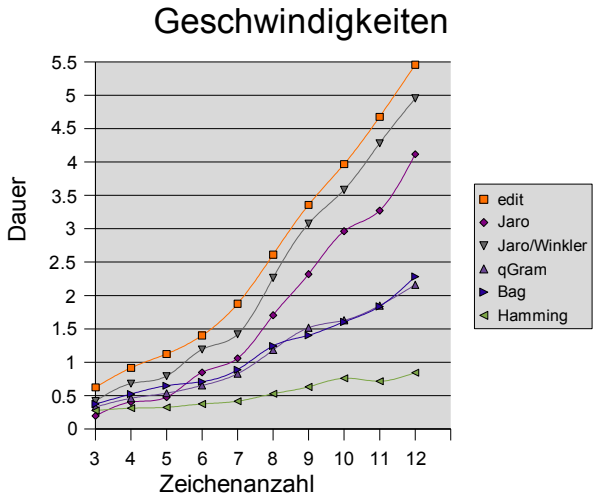
Die Kompression stellt eine kombinierte, rapiade Methode dar, da gesamte Datensätze damit gepackt werden, und die Position der einzelnen Bestandteile keine Bedeutung besitzt.

SubString-Metriken werden gerne bei der Duplikatefindung innerhalb des Onthology-Alignment genutzt, da dort meist nur die Position miteinander konkatenierter Wortsegmente variiert.

Der JaroWinkler-Algorithmus ist in der Abarbeitung schneller als die Levenshtein'sche Herangehensweise, arbeitet allerdings auch leicht schwächer.

	Soundex	SubStr	jaro	jaWi	Levensht	bagDist
„Sam J Chapman“ - „Samuel Chapman“	0,50	0,79	0,88	0,91	0,79	0,82
„Sam J Chapman“ - „S Chapman“	0,50	0,62	0,68	0,71	0,69	0,85
„Samuel Chapman“ - „S Chapman“	0,25	0,57	0,46	0,51	0,64	0,82
„Sam J Chapman“ - „Sam J Chapman“	1,00	1,00	1,00	1,00	1,00	1,00
„Samuel John Chapman“ - „John Smith“	0,25	0,26	0,37	0,37	0,26	0,66
„John Smith“ - „Richard Smith“	0,25	0,46	0,57	0,57	0,54	0,65
„John Smith“ - „Sam J Chapman“	0,25	0,00	0,32	0,32	0,01	0,58
„Sam J Chapman“ - „Richard Smith“	0,00	0,00	0,29	0,32	0,00	0,46
„Sam J Chapman“ - „Samuel John Chapman“	0,75	0,57	0,70	0,79	0,68	0,84
„Sam J Chapman“ - „Samuel John Chapman“	0,75	0,74	0,77	0,88	0,74	0,87
„Cunningham“-“Cunnigham“	0,75	0,90	0,97	0,98	0,90	0,95
„Campell“-“Campbell“	0,75	0,88	0,81	0,89	0,87	0,94
„Nichleson“-“Nichulson“	1,00	0,78	0,93	0,96	0,78	0,89
„Massey“-“Massie“	1,00	0,67	0,78	0,87	0,67	0,83

Abweichung gegenüber Messergebnissen von Simmetrik, da Formulierungen dort anders.
BagDistance sowie qGram stets größer als Levenshtein, kann daher als Vorfilter für Edit-Distance genutzt werden.



	3	4	5	6	7	8	9	10	11	12	
edit	0.624	0.916	1.124	1.400	1.876	2.612	3.356	3.968	4.676	5.456	
Jaro	0.196	0.404	0.476	0.848	1.060	1.704	2.320	2.964	3.272	4.116	
Jaro/Winkler	0.420	0.680	0.792	1.192	1.420	2.264	3.076	3.584	4.280	4.952	
qGram	0.328	0.460	0.536	0.652	0.828	1.184	1.516	1.628	1.848	2.160	
Bag	0.376	0.524	0.648	0.708	0.888	1.244	1.404	1.604	1.836	2.284	
Hamming	0.276	0.312	0.324	0.376	0.416	0.528	0.632	0.760	0.716	0.844	

4.5 Klassifizierung

Die letzte Stufe einer Harmonisierung – nachdem die Datensätze verglichen und Gewichtungsvektoren erstellt wurden, ist die Klassifizierung der Paare in „Link“, „Nicht-Link“ oder ob die Entscheidung durch einen menschliches Review überprüft werden soll, ein „possible link“.

- Klassen vorgeben
- Merkmale auswählen
- Klassengrenzen ziehen

4.5.1 Fellegi-Sunter-Klassifikator

Die Verlinkungsregeln von Fellegi-Sunter sind optimal für eine Menge Q von geordneten Paaren mit $A \times B$, falls die Fehlerwahrscheinlichkeiten P_Q und Verlinkungsregeln L_Q in Abhängigkeit von Q gezielt gewählt werden.

Der klassische Fellegi-Sunter-Klassifikator summiert alle log2-Gewichte einfach in einen Gewichtsvektor auf und nutzt zwei Grenzwerte, um ein Datensatz-Paar in eine der 3 Klassen einzuteilen.

A_1 Link ,

A_2 möglichen Link oder

A_3 Nicht-Link

Das Resultat der Klassifikation wird in einer Daten-Struktur gespeichert, welche dann genutzt werden kann, um verschiedene Ausgabepräsentationen zu produzieren.

4.5.2 Flexible Klassifikatoren

Über flexible Klassifikatoren können unterschiedlichste Methoden genutzt werden, um die finale Übereinstimmungsgewichtung für einen Gewichtungsvektor zu errechnen. Ebenfalls 2 Grenzwerte werden genutzt um das Datensatzpaar in eines der 3 Klassen zu klassifizieren. Anstatt einer einfachen Aufsummierung werden oftmals individuelle Kombinationen aus Minima, Maxima, Durchschnittswerte, Addition und Multiplikation genutzt.

4.5.3 Entscheidungsbaum

Der unter anderem in [Anan03] vorgeschlagene Algorithmus des „Decision Tree Learnings“ konstruiert aus gegebenen Informationen einen Binärbaum, welcher die Regeln für die Zuordnung in vorgegebene Klassen beinhaltet. Jeder innere Knoten des Baumes repräsentiert dabei einen Test, der auf zu klassifizierende Objekte angewendet wird. Auf den Stufen sind dabei jeweils die Tests zu verwenden, welche die gegebene Menge an Trainingsdaten bestmöglich in Bezug auf ihre Klassifizierung als „mapped“ oder „not mapped“ aufteilen. Je nachdem , ob der Test positiv oder negativ verlaufen ist, wir der jeweilige Baum-Knoten weiter traversiert, bis ein Blattknoten und somit ein Klassifikation des jeweiligen Datums erreicht wurde.

Dieser Ansatz einer dimensionalen Hierarchie vergleicht lediglich Tuple innerhalb kleiner Gruppen jeder Relation miteinander. So werden exemplarisch nur zwei Staaten-Tupel miteinander verglichen, wenn sie sich im selben Landestupel befinden oder die Landestupel wiederum lediglich Duplikate von einander sind. Da solche Gruppen meist kleiner sind als die Gesamtrelation, erlaubt diese Gruppierungsstrategie, Paare von Tupeln in allen Gruppen zu vergleichen und dennoch sehr effizient zu sein.

Dabei wird eine top-down-Traversierung der Hierarchie verwendet. Bei der obersten Relation startend, wird jede Relation gruppiert und die duplikateFindungsprozedur an jeder Gruppe vererbt (invoke). Dafür ist das Primärziel der Traversierung, jede Relation entsprechend zu gruppieren.

4.5.4 Expectation Maximization – Algorithmus

Winkler beschreibt in [Winkler91] eine weitere Möglichkeit, die Kategorie-Einteilung vorzunehmen. Ein Computersystem kann darin eine Determinierungs-Schätzung der Auftrittswahrscheinlichkeiten $m(\gamma)$ und $u(\gamma)$ für 2 Paare ausführen, indem es die Wahrscheinlichkeit berechnet, mit welcher 2 Datensätze identisch sind. Diese Wahrscheinlichkeit kan als Ober- bzw. Untergrenze genutzt werden.

Da diese über Stichproben ermittelt wird, handelt es sich um eine Maxima-Likelihood-Schätzung. Um Sie zu erhalten, wird der EM-Algorithmus genutzt. Über das folgende iterative Verfahren wird er eingesetzt, um für jedes Paar von Einträgen die Wahrscheinlichkeit der Übereinstimmung zu schätzen.

Dazu benötigt wird lediglich

- die Möglichkeit, alle Objekte als Vektoren einer Dimension n darzustellen,
- eine bekannte Funktion zur Mittelwertberechnung
- eine beliebige domänenspezifische Anzahl von Clustern, in die Objekte eingeteilt werden sollen. Dabei besitzt jeder Cluster einen Mittelpunkt, nämlich den Vektor mit der Dimension n ($m(\gamma)$ und $u(\gamma)$).

In der ersten beider Stufen, der Estimation-Stufe, wird für jedes Objekt nach einer bestimmten Verteilung die Wahrscheinlichkeit bestimmt, mit welcher es zu jedem der Cluster gehört, und diese Wahrscheinlichkeit für alle Objekte und Cluster abgespeichert.

In der zweiten, der **Maximierung**-Stufe, werden die Parameter, welche die Cluster bestimmen (z.B. der Mittelvektor), anhand der ermittelten Zuordnung neu berechnet.

Die gesamte Iteration wird abgebrochen, falls die Änderung der Likelihood unter einen bestimmten Schwellenwert fällt oder eine Maximale Iterationsanzahl erreicht wurde.

Da bei diesem Algorithmus mit jeder Wahrscheinlichkeit jedes Objekt zu jedem Cluster gehört, wird er als „weiche Clusterzuordnung“ bezeichnet.

4.5.5 Support Vector Machine

Quelle [Bilenko, M., Mooney, R.J.: Learning to combine trained distance metrics for duplicate detection in databases. Technical Report AI 02-296, University of Texas at Austin (2002)]

Nutzen Ergebnisse von verschiedenen Metriken, da die Resultate einzelner Metriken oftmals an Konsistenz mangeln.

Die Support-Vector-Machine ist ein weiterer Lernalgorithmus zu Klassifizerung von Objekten.

In der Trainingsphase werden ihm eine Menge von TrainingsTupel übergeben. Dabei besitzt das erste Element x_i das Trainingsbeispiel, während Element Nummer zwei , der Label-Teil, die Klassenzugehörigkeit angibt. Durch diese Beispiele der Form

$$\{(x_{1,} l_1), ...(x_n, l_n), x_i \in X, l_i \in \{-1,1\}\}$$

berechnet der Algorithmus eine Hyperebene, welche die beide Klassen so voneiander trennt, daß der margin, der kleinste Abstand zur Hyperebene, für die Beispiele beider Klassen maximiert wird.

Diese kann nun als Entscheidungsfunktion genutzt werden und teilt so Objekte zuverlässig in die entsprechende Klasse ein.

Die besondere Eigenschaft der SVM stellt der Zustand dar, daß sie von allen möglichen trennenden Hyperebenen, von denen es bei linear separierbaren Objekten i.A. unendlich viele gibt, diejenige mit der minimalen quadratischen Norm auswählt.

Da Trainingsbeispiele u.U. aufgrund von Meßfehlern (Nutzerfehler, etc) oder natürlichem Überlappen der beiden Klassen nicht stets streng linear separierbar sind, ist über eine Schlupfvariable eine geringe Verletzung der Nebenbedingungen möglich.

Die SVM ist innerhalb verschiedenen Bibliotheken [yale, libsvm, svmlight] verfügbar und kommt beim unter Kapitel [KapitelBilenkoMooney] vorgestellten Konzept Mooneys zum Einsatz.

4.6 Zusammenfassung

Wie in diesem Kapitel verdeutlicht wurde, kann Datenharmonisierung in verschiedenen Dimensionen durchgeführt werden.

- Es gibt die Gewichtung einzelner Attribute, was sehr häufig getan wird.
- Innerhalb der Attributwerte können durch TF-IDF häufiger und seltener auftretende Terme gewichtet werden.
- Parallel angewendete Metriken können gegeneinander gewichtet werden, bzw. als sequentieller Entscheidungsbaum „top-down-traversierung“ aufgebaut werden.

Abschließend soll ein kleiner Überblick zu frei verfügbaren Datenharmonisierungs-Tools bzw. Frameworks gegeben werden. Er stellt keinerlei Anspruch auf Vollständigkeit, sondern eine während der Bearbeitung explorativ aufgefundene Auswahl dar. Zunächst werden die Methoden-Sammlungen von Wilhelm Cohen und Sam Chapman genannt. Anschließend folgen einige Worte zu einem Recorde-Linkage-Framework von Peter Christen und Tim Churches. Abschließend wird das mit Lernmethoden aus Kapitel „Klassifizierung“ arbeitende Marlin-Framework angesprochen, und mit ähnlichen Ansätzen von Cohen et.al und Minton et.al. verglichen.

4.6.1 SecondString/SimMetrik

[secstring06] SecondString ist eine Sammlung von Java-Klassen, welche eine große Auswahl an String-Vergleichs-Algorithmen enthalten. Von W. Cohen für Zeichenvergleichs-Experimente entwickelt, umfasst SecondString Methoden zum Tokenisieren und Grobblockieren, statische Zeichenketten-Metriken sowie Gewichtungsmethoden, jedoch keine Methoden zur Daten-Standardisierung oder Klassifizierung.

[simmetric06] stellt den europäischen Gegenpart dar. Der Umfang ist vergleichbar, der Code jedoch übersichtlicher gestaltet. Beide Pakete sind als OpenSource-Download unter [SF06] zu finden.

4.6.2 Ferbl

Die Herkunft des in objektorientierten Python implementierten Frameworks [Febrl05] liegt in der Bioinformatik. An derUniversität Canberra, Australien von Peter Christen und Tim Churches entwickelt, stellt es die vollständige Mächtigkeit für Harmonisierungsaufgaben zur Verfügung. Ferbl teilt die Datenbearbeitung in die Schritte Datenanalyse/Standardisierung und Record-Linkage/Deduplication ein.

Der erste Schritt umfasst im einzelnen die Wort-Ersetzung unter Nutzung von Lookup-Tables und Korrekturlisten. Anschließende werden Datums- und Zeitformate standardisiert und die Ergebnisse zwischengespeichert.

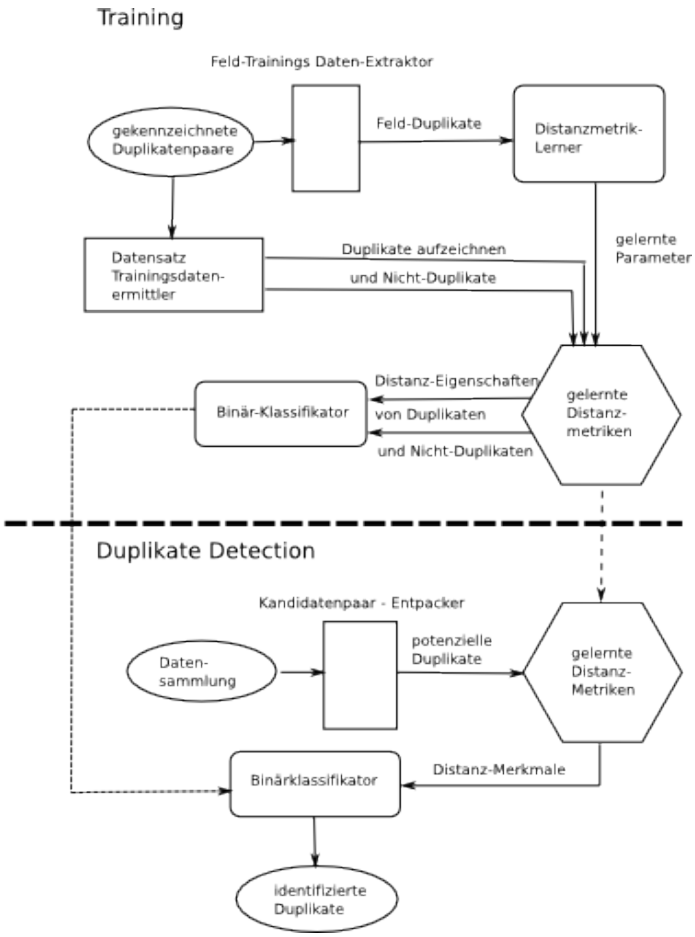
Im zweiten Schritt werden auf die Datenmenge Blockier- und Sortieralgorithmen angewendet und mögliche Treffer einem Feldvergleich mit individuellen, im Kapitel 4.4 vorgestellten Methoden zugeführt. In der Klassifizierung werden die Ergebnisse ausgewertet und ein ausführliches Logging für Protokoll-Untersuchungen erstellt.

4.6.3 Marlin

Die Ähnlichkeitsschätzung zwischen Strings ist enormen Schwankungen unterlegen. Diese hängen von der Anwendungsdomäne und Betrachtungsweise (Aspekt) der Daten ab. Anstatt diese Gewichtungen von Hand anzulernen sind trainierbaren Ähnlichkeitsberechnungen der Vorzug zu geben.

Die Folgende drei Ansätze, wovon „Marlin“ als Einziger umgesetzt zu sein scheint, arbeiten mit einer vom Nutzer vorher gegebenen Traingsmenge an Daten. Diese beinhaltete gekennzeichnete korrekte sowie inkorrekte Vergleichsdatensätze und berechnet daraus zunächst bestimmte Parameter für die Abarbeitung der eigentlichen Daten.

Marlin („Multiple Adaptive Record Linkage with Induction“)ist eine auf dem Weka-Framework [weka06] basierende, automatische Trainingsumgebung der Universität Texas [Bilen03]. Sie arbeitet in den 2 Phasen „Duplikate-Training“ und „Duplikate-Erkennung“.



In der Trainingsphase bekommt Marlin lernbare DistanzMetriken für jedes Datenfeld präsentiert. Homonyme („noisy trainingsdata“) wie beispielsweise „Asian“ und „Seafood“ werden dabei als Trainingspaare erkannt und erhalten eine Ähnlichkeit zugewiesen.

Nach dem Erlernen dieser Ähnlichkeitsmetriken für jedes individuelle Datensatzfeld wird das gewonnene Wissen genutzt, um die Distanz für jeden Datensatz zu berechnen. Diese Vektoren bestehend aus Distanz-Merkmalen und werden als Trainingsdaten für den Binärklassifikator genutzt. Bei diesem Vorgang kommt der unter Kapitel 4.5.5. vorgestellte „Support Vector Machine“-Algorithmus zur Anwendung, welche traditionelle Entscheidungsbäume übertrifft [kdd03].

In der „Erkennungsphase“ werden alle Datensätze miteinander verglichen. Da dies eine Anzahl von $O(n^2)$ Vergleichen bedeuten würde, nutzt „Marlin“ einen aus Canopies-Clustering via Jaccard bestehenden Pre-Blocking-Algorithmus.

Anschließend werden die gelernten Distanz-Metriken verwendet, um jedes Feld von jedem Paar möglicher Duplikat-Datensätze auszuwerten, also der Erstellung von Distanz-Feature-Vektoren für den Klassifizierer. Glaubhafte Schätzungen zur Einordnung in die Gruppe der Duplikate oder Unikate werden durch den Binär-Klassifikator vorgenommen, und Paare werden mit aufsteigender Reihenfolge sortiert.

4.6.4 Cohen und Richman

Der von Cohen und Richmann genutzte Ansatz verwendet ebenso den im Abschnitt [Canopy] vorgestellten Canopy-Algorithmus. Der Algorithmus wählt zu Beginn aus dem kompletten Datenbestand bestimmte Datensatzpaare aus, für die der Benutzer die Einteilung „match“ oder „non-match“ übernimmt. Diese werden darauffolgend als „Mittelpunkte“ genutzt.

Der Canopy-Algorithmus teilt nun mit Hilfe der Parameter T_{tight} und T_{loss} alle weiteren Paare den identifizierten Paaren zu.

Mit den gewonnenen Informationen bildet der Klassifikations-Lerner zwei Funktionen. Zum einen eine boolean-Funktion h(a,b), die zurückgibt, ob die Datensätze a und b übereinstimmen, zum zweiten eine Funktion c(a,b), die die Wahrscheinlichkeit liefert, mit der die Klassifizierung h(a,b) positiv ist. Je höher c, umso sicherer ist der Klassifikations-Lerner mit der Einstufung der beiden Records als Duplikate.

Um den Maschinenlearner, meist ein „Expectation Maximization Learner“, zu trainieren, werden aus den Kandidatenpaaren m Paare ausgewählt, für die der Nutzer die Übereinstimmung bewertet.

Hierbei werden aus den beiden Records mehrere Eigenschaften zusammen betrachtet, um daraus die Hypothese c(a,b) möglichst genau anzupassen. Eigenschaften sind binär, also ein Datensatz, und besitzt eine Eigenschaft oder nicht. Mögliche PaarEigenschaften sind SubstringMatch, PrefixMath, EditDistance, MatchAToken, MatchBToken, MatchABigram, JaccardDistance, StrongNumberDistance.

Die Spalten „Titel“ und „Adresse“ könnten folgende Eigenschaften genutzt werden: $Substring_{Titel}$, $SubString_{Adresse}$, $PrefixMatch_{Titel}$, $PrefixMatch_{Adresse}$, $StrongNumberMatch_{Titel}$, $StrongNumberMatch_{Adresse}$. Der Learner gewichtet die Nutzerauswahl, und stärkt somit sein internes Klassifikations-Modell. Abschließend wird auf die Gesamtmenge an Werten ein „GreedyAgglomerativeClustering“-Algorithmus [CallumNigamUngar] angewendet, welcher die Gesamtmenge aufteilt, bis eine vorgegebene Teilmengenanzahl erreicht wurde.

4.6.5 Tejada-Knobloch-Minton

Dieser Ansatz arbeitet äquivalent zu Cohen und Richman, baut allerdings nach der 2. Lernphase einen Entscheidungbaum („Decision Tree“) aufgebaut, und alle weiteren Vergleiche über diesen traversierend durchgeführt.

5 Implementierung

„Mediator – definiert ein Objekt, welches die Interaktion einer Menge von Objekten kapselt. Mediatoren ermöglichen eine loose Kopplung, indem Objekte vom expliziten verlinken aufeinander abgehalten werden und es ermöglicht die unabhängige Veränderung ihrer Interaktion.“ [designPattern]

Das durch Xcerpt vorgestellte Prinzip erlaubt es, einfach aus Web-Dokumenten mit einer Toleranz in Breite und Tiefe Inhalte zu excerpieren.

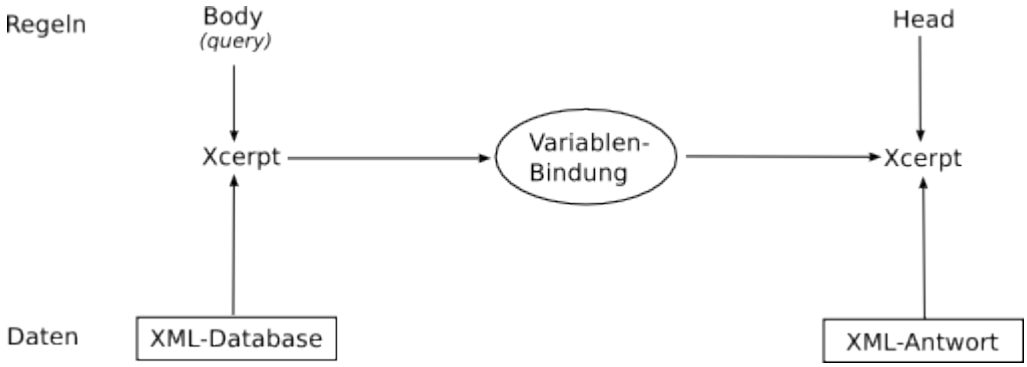


Illustration 8: Abarbeitungsaufbau einer Xcerpt-Regel

Um dieses System mit Harmonicing-Capabilities auszustatten, können folgende Punkte bedacht werden:

- Komponenten in der Xcerpt-Abfragesprache implementieren
- Komponenten in den Xcerpt-Parser aufnehmen und die Sprache dementsprechend erweitern
- Xcerpt als Information-Retrieval nutzen und die Harmonisierung in einer darüberliegenden Schicht lösen

Diese 3 Stufen sollen nun auf Ihre Eignung für die Datenharmonisierungs-Schritte

- Schema-Integration,
- Ontologie-Integration,
- Record/Entity-Integration/Resolution

näher untersucht werden.

5.1.1 Komponenten direkt in Xcerpt

5.1.1.1 Schema-Integration

Xcerpts Sprachumfang umfaßt eine Vielzahl von Möglichkeiten im statistischen Bereich sowie bei der Auswahl der zu selektierenden Informationen.

Durch reguläre Ausdrücke ist eine gezielte Auswahl von Abfragebestandteilen aus dem Quellmaterial möglich, zumal deren Formatierung innerhalb einer Domäne im Allgemeinen konstant bleibt. Datums- und Zahlenformatierungen können auf diese Weise umgesetzt werden.

```
GOAL
date { &join(var DATE_11, „20“, var DATE_12) }
FROM
orig_date {
    /^(var DATE_11 ->[0-9]{1,2}\.[0-9]{1,2}\.)(var DATE_12 ->[0-9]{2})$/
}
END
GOAL
```

```
  date { var DATE_21 }
  FROM
  orig_date {
      /^(var DATE_21 ->[0-9]{1,2}\.[0-9]{1,2}\.[0-9]{4})$/,
  }
  END
```

Ebenfalls der Abgleich mit Ersetzungslisten ist möglich.

5.1.1.2 Ontologie-Integration

Durch den deduktiven Ansatz Xcerpts kann auf Ontologie-Resourcen bei Bedarf zurückgegriffen werden. Ist diese in Bezeichnungslogik (Deskription Logic), eine entscheidbare Untermenge der Prädikatenlogik, notierte, so kann beispielsweise das auf XML basierende Web Ontology Format OWL Verwendung finden.

Dieses besteht aus einen Definitions- oder Terminologieteil „TBox“ sowie einem erweiterten Wissensteil (asserted knowledge, „ABox“).

Im Terminologie-Teil wird der Aufbau der Ontologie näher beschrieben.

Durch folgendes Beispiel können Datentypen und Subklasse einer Ontologie abgefragt und anschliessend auf die Subclass-Elemente übertragen werden:

```
  CONSTRUCT
    subClasses { all classes { var Class,
      optional sub [ all var SubClass    ],
      optional datatypes [ all var dataType ]
  } }
  FROM
  ... Namespace - Definition ...
      in {  resource { "file:data/Countries_iso.owl", "xml" },
      or {
          desc owl:Class {{
            attributes {{ rdf:ID [ var Class ] }},
                rdfs:subClassOf {{
                  desc owl:Class {{
                        attributes {{ rdf:about [ /^#(var SubClass ->.*)$/ ] }}
                  }}
                }}
            }},
          desc owl:DatatypeProperty {{
            attributes {{ rdf:ID [ var dataType ] }},
            desc  rdfs:domain {{
                attributes {{ rdf:"resource" [ /^#(var Class ->.*)$/ ] }}
            }},
            desc rdfs:range {{
                attributes {{ rdf:"resource"
[ "http://www.w3.org/2001/XMLSchema#string" ] }}
            }}
          }},
          desc owl:FunctionalProperty {{
            attributes {{ rdf:ID [ var dataType ] }},
            desc rdfs:domain {{
                attributes {{ rdf:"resource" [ /^#(var Class ->.*)$/ ] }},
            }}
          }}
      }
      }
  END
```

5.1.1.3 Record/Entity-Integration/Resolution

Für Abfragen baut Xcerpt intern eine Matrix auf, in welcher mögliche fehlende Resourcen durch Konkanetation mit zusätzlichen Regeln gelöst werden. Da besonders String-Metriken oftmals iterativ arbeiten, müssen diese nicht nur durch eine Vielzahl einzelner Regeln dargestellt werden (was die Source-Übersicht nicht erhöht), sondern sind ebenfalls langsamer und speicherlastiger als eine native/implementierte Lösung (siehe Kapitel [in Xcerpt integriert]). Als Beispiel wurde der Bigramm-Algorithmus sowie eine Term-Frequency-Übersicht in Xcerpt angefertigt.

5.1.2 Komponenten in Xcerpt-Sprachumfang

5.1.2.1 Schema-Integration

Teile der Standard-Formatierung können grundlegend in Xcerpt implementiert werden. Ähnlich der Datums-Parse-Funktion unter Java, könnte eine Xcerpt-Arithmetic ein korrekt formatiertes Datumsformat zurückliefern.

Da dafür regulärer Ausdücke zum Einsatz kommen und die Eingangsformatierung Schema-abhängig ist, wird diese Aufgabe besser innerhalb Xcerpts gelöst.

5.1.2.2 Ontologie-Integration

Hans Eric Svensson stellt in seiner Diplomarbeit [Sven04] eine erweiterte Xcerpt-Version vor, welche Ontologie-Abfragen innerhalb einzelner Regeln akzeptiert und an einen OWL Reasoner weiterleitet.

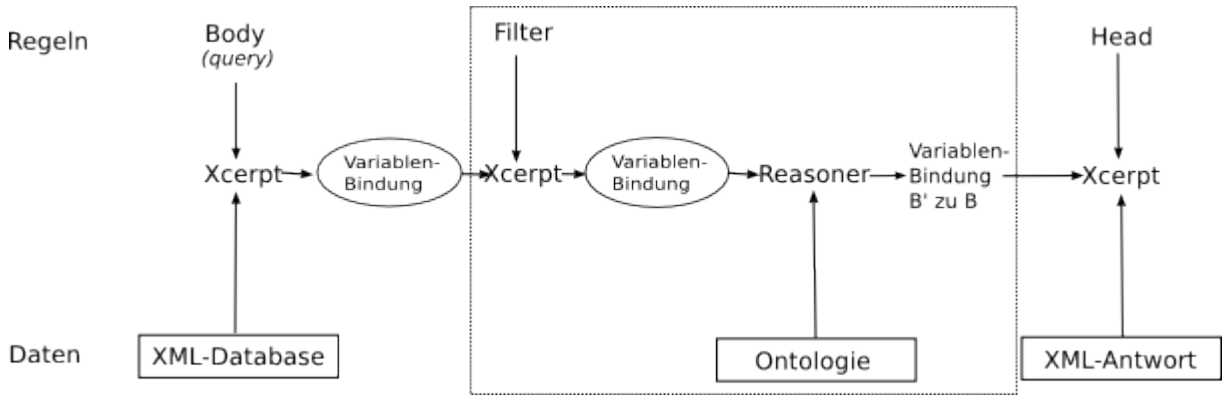


Illustration 9: Ablauf Xcerpt unter Anwendung eines externen Reasoners

5.1.2.3 Record/Entity-Integration/Resolution

Distanz-Metriken empfehlen sich für eine Implementierung direkt in die Sprache selbst. Höhere Verarbeitungsgeschwindigkeit, übersichtlicherer Quellcode sind Argumente dafür.

Laut Kapitel 3 ist die Unvollständigkeit eine Eigenschaft des Anfrageterms, so dass eine Implementierung in diesem Bereich zu präferieren ist.

conditional clause

String-Metriken können als neue Arithmetic-Funktionen implementiert werden. Die Verwendung entweder im Body- und Head-Teil, als auch eine Möglichkeit zur Implementierung eines Zwischenschritts (CONSTRUCT-FILTER-FROM-END) ist möglich.

```
and {
    var LOCATION1,
    var LOCATION2
```

```
} where (&jaroMetrik(var LOCATION1, var LOCATION2) gt int(0.7) )
```

unscharfe Abfrage

```
veranstaltungsDatenbank{{
veranstaltung{{ var Location -> unscharf( 80% , "Blauer Engel") ,
titel[[ ...
}}
}}
```

unscharfes Grouping

```
...
```

5.1.2.4 Nutzung eines externen Tools

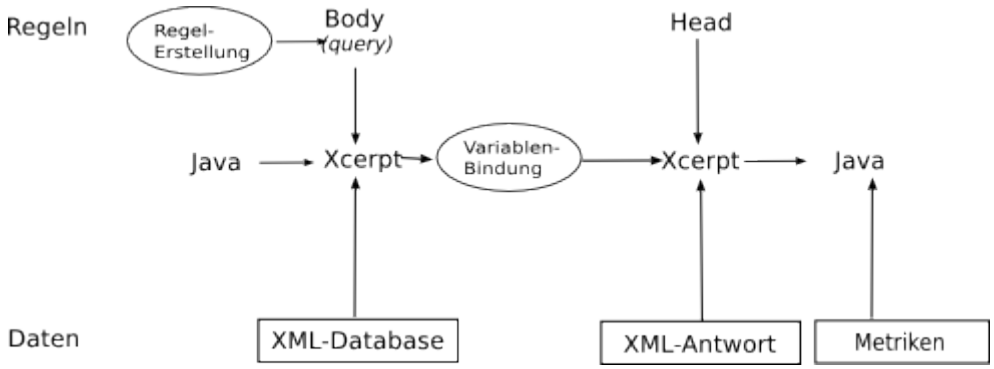
Mit den in Kapitel [...] vorgestellten OpenSource-Lösungen existieren starke Lösungen für Metrik-Berechnungen, welche unabhängig von Xcerpt weiterentwickelt werden.

Eine Outsourcing-Lösung ist sinnvoll, sobald die dadurch erreichten Ergebnisse von höherer Güte sind, als dies durch interne Lösungen möglich ist. Bei großen Ontologie-Abfragen, wo eine kleine Datenmenge einem großen Rechenaufkommen entgegensteht ist dies sinnvoll.

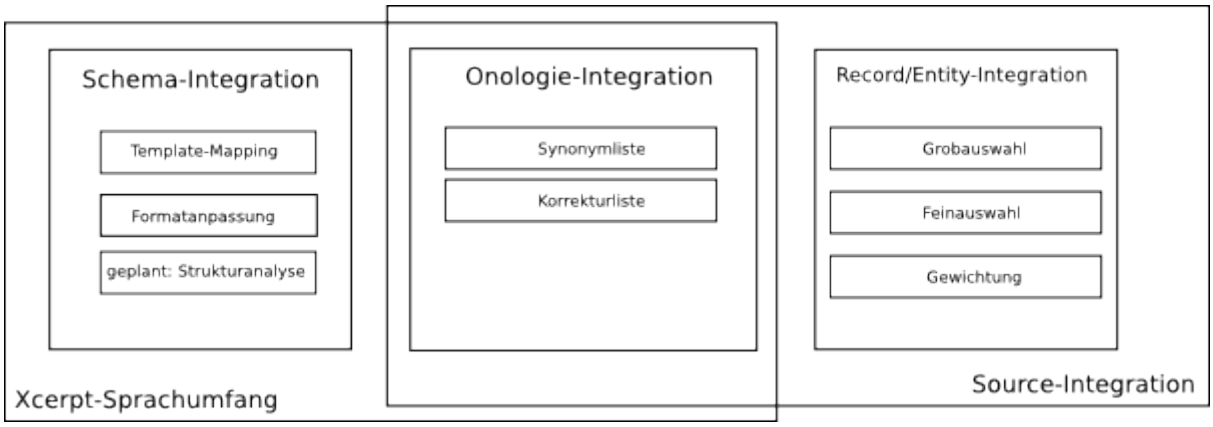
Da es sich bei der Duplikate-Berechnung meist um einen Abgleich großer Datenmengen handelt, ist die Übertragung der entsprechenden Rohdaten oft langsamer als die direkte interne Lösung. Nach der Umsetzung Xcerpts in Java ist eine Nutzung der vorgestellten Bibliotheken von Vorteil.

5.1.3 Xcerpt für bloßes Retrieval

Über verschiedene APIs kann Xcerpt innhalb anderer Software-Komponenten genutzt werden. Da zum Zeitpunkt dieser Diplom-Anfertigung kein funktionstüchtiges XcerptV2-API vorhanden war, soll hier nicht näher auf diese Möglichkeit eingegangen werden.



5.2 Beispielimplementierung



Schema-Integration

Template Mapping

Format-Anpassung

Struktur-Analyse

Ontologie-Integration

Recod-Entity-Integration

unscharfes Matching

unscharfes Grouping

Diskussion:

Datenharmonisierung in Xcerpt muss ohne manuelle Eingriffe erfolgen

Datenmaterial wird heterogen sein, allerdings oftmals bestimmte Untergruppen beinhalten... Namen, Titel, Adressen, Geburtsdaten...

Datenmengen sind ebenfalls vergleichsweise flexibel, kleiner bis mittlere Mengen zu erwarten.

-> selbständige Identifikation der jeweiligen Variablen-Inhalte

-> erstellen eines entsprechenden Binärbaumes für Abarbeitung mit zu Formaten passenden Methoden und Parametern (bspw. selbständige Erkennung Datum, Anwendung Hamming-Distance)

-> Verfügbarmachung bestimmter Lernalgorithmen-Ergebnisse und Abfrage dieser

-> wichtiger: hohe Verarbeitungsgeschwindigkeit wg. Echtzeitnutzung

-> Diskussion: verschiedene Algorithmen vorher anlernen und entsprechende Parameter implementieren?

5.2.1 Implementierung

5.2.1.1 Strukturanpassung

durch einfache Aggregation „&replaceSpecialChars“ werden in Termen Umlaute und Sonderzeichen entfernt. Ausländische Buchstabenkombinationen, wie ĉ werden in ihre entsprechenden europäischen Normen umgestellt. Problematisch innerhalb Haskells ist dabei die Darstellung von Umlauten aus 2 Sonderzeichen. So wird ein „ü“ bspw. durch „\\195\184“ dargestellt, was die Betrachtung als Char-

Werte verhindert und die Aggregation zu einem SubString-Vergleich erweitert.

Die Aggregation „split“ teilt einen Term in durch ein Zeichen markierte Subterme.

5.2.1.2 Ontologien

einfache Methode, Attribute einer Datenmenge einer Ontologischen Gruppe zuzuordnen:

Termfrequenz- Inverse Document Frequenz für Anfangs- bzw. Endbuchstaben der Werte berechnen. Dies ist einfach über die Xcerpt-Bordmittel „reguläre Ausdrücke“ sowie entsprechend angeschlossenen Ontologien möglich.

Das Eingangsbeispiel „Nick Street“, „Wall Street Journal“, „Main Street“ kann mit Instanzen einer Ontologien-Datenbasis auf deren Auftreten verglichen werden. Werden viele häufig auftretende Teile in einer bestimmten Ontologie als Sub- bzw. Supkonzepte aufgefunden, so ist eine Zugehörigkeit zu dieser Ontologie sehr wahrscheinlich. Durch die Aggregation „&count“ kann die Anzahl eines Terms ausgegeben werden. Xcerpt aggregiert auftretende Terme selbständig.

Das überprüfen innerhalb Xcerpts ist durch die NS-Implementierung sowie unvollständiges Matching in Teife und Breite sehr gut möglich.

5.2.1.3 Datenharmonisierung

unscharfes Matching

einen Unscharf-Vergleich zwischen 2 Werten kann durch den im Xcerpt-Sprachumfang enthaltenen Conditional-Cause leicht umgesetzt werden. Form: and {...} where (&jaroDistance(Wert1, Wert2) > Threshold).

Dieser Fall ist jedoch trivial, da keinerlei Parametrisierung möglich ist und es bei höheren Wertepaaren in bezug auf Zeitkosten unrentabel wird.

unscharfes Grouping

Eine Umsetzung der Harmonisierung als unscharfes Grouping ist anzustreben. Ein Beispiel sei kurz dargestellt:

mgroup by [Variable1, Variable2]

--> Gewichtung einzelner Attributwerte

--> Gewichtung einzelner Term-Werte nach TF-IDF

--> Selektion passender Ähnlichkeitssmetriken

Diese Annäherung stellt ein klassisches Clustering-Problem ohne vorher gegebene Clusteranzahl dar, bei welchem die im Kapitel 4.3. - 4.6. vorgestellten Schritte angewendet werden kann.

Xcerpt kennt bereits einen Sprachkomponenten für Grouping nach SQL-Standard. im Konstruktionsteil kann über „group by“ bereit das Ergebnis nach bestimmten Werten gruppiert werden. Dieser bezieht sich auf eine Equivalenz-Relation, welche bei Ähnlichkeitsmessungen nicht gegeben werden kann. Weiterhin sind Ergebnisse einer Ähnlichkeitsmessung zwar reflexiv und bijektiv, jedoch nicht unbedingt hart transitiv, was folgendes Beispiel belegen soll: falls „Tim“ zu „Tom“ ähnlich ist, sowie „Tom“ zu „Rom“ (unter Auslassung der Ontologie-Zugehörigkeit) ähnlich, so ist dennoch „Tim“ unähnlich „Rom“.

Es wurde daher zusätzlich ein neuer Term „mgroup by“ eingeführt, welcher für die selbe Aufgabe Werte-Unschärfen zulässt. Innerhalb Xcerpts wird durch diesen Term die Regel/Funktion „groupSubstitutions“ aufgerufen, welche als 1. Argument die Liste der nach denen-zu-gruppierenden

Variablen namen enthalt. Der 2. Parameter beinhaltet eine Liste der ungebundenen Substitutionen. Als Ergebnis wird die Gruppierung dieser Listen erwartet.

Da der Nutzer bei der Ausgabe einer Gruppierung die Ergebnisse unverändert und vollständig erwartet, können Standardisierungsaufgaben nicht am original-Datenset durchgeführt werden. Weiterhin kann das Original-Datenset Variablen beinhalten, welche zur Harmonisierung nicht herangezogen werden dürfen, jedoch trotzdem nicht verloren gehen sollten, andere Werte sind lediglich optional vorhanden.

Es ist daher nötig, dieses zu kopieren, eine Version für die Datenharmonisierung aufzubereiten, und dieses am Ende mit dem Original-Set wieder in Verbindung zu bringen.

Weiterhin ist ein möglichst plakativer Repräsentant aus jedem Ergebnis-Cluster auszuwählen.

Vorteilhaft ist in Haskell das Lazy-instantiating, welches einmal berechnete Werte zusätzlich cached.

Eine übergebene Ergebnismenge wird zunächst vorsortiert und auf die benötigten Variablen reduziert. Nicht zum Gruppierung genutzte Bestandteile entfallen, Instanzen optionaler Variablenmengen werden durch leere Terme ersetzt.

Im nächsten Schritt werden die Instanz-Werte von Sonderzeichen befreit und das Currying aufgehoben. Attributwerte einzelner Substitutionen können nun in Matrizen-Spalten gleicher Attributwerte transponiert werden.

Diese Werte werden daraufhin, unter ZuHilfename des Variablennamens auf deren Inhalt analysiert. Zahlenformate bekommen dabei als Ähnlichkeitsmetrik eine harte Hammingdistanz-Überprüfung zugeschrieben, Stringwerte eine Reihe von Ähnlichkeitssmetriken.

In Abschnitt [BagDistance-Levenshtein] wurde gezeigt, daß Bigram- und BagDistanz echte Untermengen von Levenshtein darstellen. Daher werden nun nach Canopese-Methodel und GreedyAlgorithmus die Ähnlichkeiten der einzelnen Werte gegeneinander aufgerechnet.

Die letzliche Gruppierung wird durchgeführt, indem für jedes VariablenTupel ein String mit eindeutiger Gruppierung geliefert wird, welches als Ergebnis für groupSubstitution die Gruppierung darstellt.

6 Ausblick

<!--AB HIER-->Über verschiedene M-Tree und Cluster-Ansätze wird ebenfalls versucht, (Zwischen-) Ergebnisse eines Stringvergleiches für den Vergleich folgender Zeichenketten zu nutzen und damit Zeitkosten zu sparen. <!-- in den Schluss -->

Literaturverzeichnis

[Noy05] Natasha NoyOntology Mapping and Alignment2005
[Kramer06] Andre KramerHerrscher über das Chaos
[Ferber03] Reginald Ferber2003
Peter Christen, Tim ChurchesFebrl - freely extensible biomedical record linkage
[Fellegi69] Fellegi, I. P., Sunter, A. B.A Theory for Record Linkage1969
[Wiki06] Precision and Recall2006http://de.wikipedia.org/wiki/Precision
[vanReijsenbergen75] Cornelis Joost van RijsbergenInformation retrieval1975
[HS95] M. Hernandez, S. StolfoThe merge/purge problem for large databases1995
M. G. Elfeky, V.S.Verykios, A. K. ElmagarmidTAILOR: A Record Linkage Toolbox2002
[Nigam2000] Andrew McCallum, Kamal Nigam, Lyle H. UngarEfficient Clustering of High-Dimensional Data-Set with Application to Refer
[Kdd03] Rohan Baxter, Peter Christen, Tim ChurchesA Comparison of fast Blocking Methods for Record Linkage2003
[Inte03] M.Bilenko, R.Mooney, W.Cohen, P. Ravikumar, St.FienbergAdaptive Name Matching in Information Integration2003
[Soundex00] The U.S. National Archives and Records AdministrationThe Soundex Indexing System2000http://www.archives.gov/genealogy/census/soundex
[Winkler91] William E. Winkler
[Vldb01] Luis Gravano, Panagiotis G. Ipeirotis, H. V. Jagadish, Nick KoudasApproximate String Joins in a Database (Almost) for Free2001
[Stoilos05] Giorgos Stoilos, Giorgos Stamou, and Stefanos KolliasA String Metric for Ontology Alignment2005
[Keogh04] Eamonn Keogh, Stefano Lonardi, Chotirat Ann RatanamahatanaTowards Parameter-Free Data Mining2004
[Anan03] Rohit Ananthakrishna, Surajit Chaudhuri, Venkatesh GantiEliminating Fuzzy Duplicates in Data Warehouses2003
[secstring06] William W. Cohen, Pradeep Ravikumar, Stephen FienbergSecondString2006http://secondstring.sourceforge.net/
[simmetric06] Sam ChapmanSam's String Metricshttp://www.dcs.shef.ac.uk/~sam/stringmetrics.html
[SF06] SourceForge2006http://www.sourceforge.net
[Bilen03] Mikhail Bilenko, Raymond J. MooneyAdaptive Duplicate Detection Using Learnable String Similarity Measures
[Sven04] Hans Eric SvenssonExtending Xcerpt with Ontology Queries2004