UNIVERSITÄT LEIPZIG
Fakultät für Mathematik und Informatik
Institut für Informatik

Analyse der Wiederverwendung zum Schema-Matching

Diplomarbeit

Leipzig, März 2005

vorgelegt von Sabine Maßmann
geboren am 28.03.1979
Studiengang Informatik

Betreuer Prof. Dr. Erhard Rahm

Kurzfassung

Datenbanken finden in mehr und mehr Bereichen einen Einsatz. Angestrebter Datenaustausch zur Vernetzung von Informationen und Gewinnung zusätzlicher Kenntnisse ist wegen der Heterogenität von den Datenbanken problembehaftet. Schema-Matching beinhaltet durch das Erzeugen von Korrespondenzen zwischen Schemata eine Lösungsmöglichkeit für zumindestens einen Teil dieser Probleme. Viele Ansätze wurden schon für das Schema-Matching entwickelt und werden in dieser Arbeit dargestellt. Es werden Strategien zur Wiederverwendung näher erläutert und Erweiterungen für die Mapping-Wiederverwendung vorgestellt. Die neuen Strategien, die in dieser Diplomarbeit entstehen, wurden in dem Schema-Matching-Prototyp COMA++ integriert und zusammen mit anderen Strategien evaluiert.

Vorwort

Die vorliegende Arbeit wurde im Rahmen des Prototyps COMA++ entwickelt.

Folgenden Personen möchte ich besonders danken:

- Herrn Prof. Dr. E. Rahm für die Aufgabenstellung und die Unterstützung der Arbeit,
- Herrn H. Do für die umfassende Betreuung der Arbeit und die vorangegangenen Implementierungen,
- Herrn A. Thor für die hilfreichen Hinweise zur stilistischen und inhaltlichen Verbesserung.

Sabine Maßmann, März 2005

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Seitdem Computer in vielen Bereichen des Alltags eine große Rolle spielen, werden auch immer mehr Daten elektronisch gespeichert und verwaltet. Dafür werden oft Datenbank-Management-Systeme (DBMS) verwendet, die Daten in Datenbanken speichern und verwalten.
Es gibt ganz unterschiedliche Verwendungen für Datenbanken (DB) und DBMS, unter anderem: Firmen speichern die Produktinformationen sowie Kunden- und Bestellungsdaten; Forschungsgruppen z.B. in der Biologie haben riesige Mengen von Versuchsergebnissen[1], die durch DBMS erst handhabbar und auswertbar geworden sind; Banken speichern neben den Kundeninformationen alle dazugehörigen Transaktionen wie etwa Überweisungen und Lastschriften; des weiteren durchforsten Suchmaschinen Teile des Webs und erstellen Statistiken dazu, die in Datenbanken verwaltet werden, damit die Benutzer beispielsweise bei der Eingabe von Schlüsselwörtern die entsprechenden Treffer angezeigt bekommen.
Verschiedene Gründe sind denkbar, warum man auf anderswo gespeicherte Daten zugreifen will oder diese gar zusammen in einer Datenbank halten möchte. Ein Szenario ist das Zusammenführen der Buchbestandsinformationen von Zweigstellen einer Bibliothek, die ihre Information in jeweils eigenen Datenbanken verwaltet haben. Die Daten sollen möglichst automatisch in eine gemeinsame Datenbank eingefügt werden, die dann die gesamten Informationen führt. In Abbildung 1.1 ist ein Datenauszug aus zwei Bibliotheksdatenbanken zu sehen. Um die Daten vereinigen zu können, müssen herausgefunden werden, welche Daten der einen Datenbank in Beziehung zu den Daten der anderen stehen. Dies ist in der Abbildung durch Pfeile gekennzeichnet.
Ein anderes Szenario zum Zugriff auf verschiedene Datenbanken ist, wenn eine Forschungsgruppe ein bestimmtes Protein untersuchen will und vorher herausfinden möchte, welche anderen Gruppen damit experimentierten bzw. was diese schon herausgefunden haben und in welchen Regelkreisen es welche Rolle spielt. Dies sind Informationen, die über viele verschiedene Datenbanken von anderen Abteilungen und Institutionen verteilt sind.

Bei dem Austausch der Information tauchen jedoch unterschiedliche Probleme auf. Neben verwendeter ungleicher Hardware, unterschiedlichen Betriebs- und Datenbanksystemen sowie verschiedenen Modellierungssprachen, ist die logische Heterogenität von großer Bedeutung. Die Systeme wurden getrennt voneinander entwickelt und so

[1] Weltweit gibt es über 500 Bio-Datenbanken [Bax03].

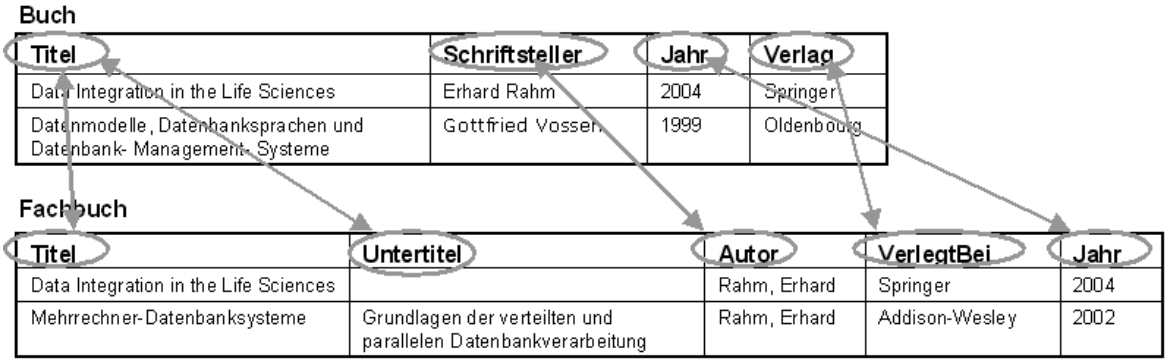


Abbildung 1.1: Auszug aus den Daten zweier Bibliotheksdatenbanken und der Beziehung der Informationen

entstanden verschiedene Darstellungen der Daten oder Unterschiede in deren Bedeutung, wie z.B. unterschiedlich verwendete Abstraktionsebenen und Namenskonflikte. Dies beruht zum Teil darauf, dass die DBMS ganz unterschiedliche Daten verwalten sollten, aber auch darauf, dass verschiedene Leute unterschiedliche Stile und Aufgaben auf ihre Weise lösen.
Auch als immer mehr Menschen den Nutzen in einheitlichen Formaten sahen, wurden für unterschiedliche Bereiche natürlich auch darauf angepasste Standards entwickelt und für diese dann noch verschiedene Versionen und Varianten. Das Chaos bei der Datenspeicherung und -übertragung wurde also nicht gelöst.
Ein Nutzer, der Daten von einem anderen System auf seinem betrachten möchte, will dies jedoch möglichst in der ihm vertrauten Darstellung tun. Diese Umformung ist erst recht nötig, wenn er die Daten abspeichern will, denn es soll nicht für jede betrachtete Quelle gleich eine neue Datenbank angelegt werden müssen. Für diese Transformation der Daten von einem System in ein anderes werden Regeln benötigt, die zuerst erstellt werden müssen. Diese Aufgabe wird häufig noch von Menschen per Hand erledigt unter teilweiser Nutzung einer grafischen Umgebung. Sind die verwendeten Datenmodelle sich sehr ähnlich und klein, ist das Aufstellen der Regeln relativ einfach. In der Praxis ist es jedoch oft so, dass die Modelle oft groß und sehr unterschiedlich sind, was beispielsweise an unterschiedlicher Verwendung von Bezeichnungen und Struktur liegt. Das Auffinden der entsprechenden Regeln ist damit ein sehr komplexes Problem, das zu lösen langwierig, zeitaufwendig und sehr kostspielig ist, weswegen man solche Probleme lieber mit dem Computer angehen will.
Das Auffinden der gesuchten Beziehungen zwischen den Schemaelementen wird mit dem *Schema Matching* angegangen. Bisher wurden bereits verschiedene Matchalgorithmen entwickelt und getestet. Der Prozess des Matchens kann zur Zeit jedoch nicht ganz ohne Menschen ablaufen, die das Endergebnis noch mal überprüfen, da kein Matchalgorithmus bisher 100% korrekt arbeitet und falsche Beziehungen findet oder nicht alle richtigen entdeckt.

1.2 Grundlagen

In diesem Abschnitt wird eine Einführung in die Thematik des Schema-Matching mit Grundbegriffen und Erklärungen anhand eines Beispiels erfolgen. Für formale Definitionen sei an [AB01], [MBDH02], [MBHR04] und [SvKJ04] verwiesen.
Ein *Schema* ist eine Menge von Elementen, die durch Strukturbeziehungen miteinander verbunden sind. Es existieren verschiedene *Schemasprachen*, wie beispielsweise relationale Datenbankschemata, XML[2] Dokumenttypdefinition (DTD), XML Schemadefinition (XSD) und Ontologien. In Tabelle 1.1 sieht man zum einen das relationale Schema *S1* und das XML Schema *S2*. *Schemaelemente*, die zusammen das Schema bilden, sind dabei Attribute, welche neben einem Namen auch Eigenschaften wie z.B. Datentyp, Wertebereich besitzen. Beispiele für Schemaelemente aus der Tabelle sind für das Schema *S1* `Titel`, dessen Datentyp eine Zeichenkette mit 100 Zeichen ist, und `Jahr`, welches eine Integer-Zahl repräsentiert.

a)

```
CREATE TABLE Buch (
  Titel          VARCHAR(100),
  Schriftsteller VARCHAR(100),
  Jahr           INT,
  ISBN           VARCHAR(10),
  Verlag         VARCHAR(100),
  PRIMARY KEY    (ISBN));
```

b)

```
<xsd:schema xmlns:xsd="http://www.w3.org/
                    XMLSchema">
<xsd:element name="Fachbuch">
 <xsd:complexType>
  <xsd:sequence>
   <xsd:element name="Titel" type="xsd:string"/>
   <xsd:element name="Untertitel" type="xsd:string"/>
   <xsd:element name="Autor" type="xsd:string"/>
   <xsd:element name="Jahr" type="xsd:integer"/>
   <xsd:element name=
            "InternationaleStandardbuchnummer"
            type="xsd:string"/>
   <xsd:element name="VerlegtBei" type="xsd:string"/>
   <xsd:element name="E002Z" type="xsd:integer"/>
  </xsd:sequence>
 </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Tabelle 1.1: Beispiel für zwei Eingabeschemata: a) Relationales Schema *S1*, b) XML Schema *S2*

Das Schema-*Matchen* von Schemata wird als eine Funktion definiert mit Eingabe von zwei Schemata und gegebenenfalls von Zusatzinformationen, bei der die Ausgabe aus einem *Mapping* zwischen den zwei gegebenen Schemata besteht. Die beiden

[2]Extensible Markup Language (XML) http://www.w3.org/xml

Schemata *S1* und *S2* bilden dabei das *Matchproblem S1-S2*, welches auch als *Matchaufgabe* bezeichnet wird, zu welchem ein Mapping erzeugt werden soll. Ein Mapping (=Abbildung) ist eine Menge von Mappingelementen und ist das Ergebnis eines *Matchprozesses*. Ein *Mappingelement* wird auch als Korrespondenz bezeichnet und gibt an, dass bestimmte Elemente des Schemas *S1* auf bestimmte Elemente des Schemas *S2* abgebildet werden. Eine Darstellungsmöglichkeit für ein Mapping ist $M : S1 \leftrightarrow S2$, wobei M der Mappingname ist und *S1* bzw. *S2* die Schemata sind, auf die sich das Mapping bezieht. Bei der verkürzten Darstellung $S1 \leftrightarrow S2$ wird der Mappingname weggelassen. Ein Mapping M für das Matchproblem $S1 - S2$ aus Tabelle 1.1 enthält als Ergebnis einer Matchfunktion fünf Mappingelemente:

1. `Buch.Titel` ≅ {`Fachbuch.Titel, Fachbuch.Untertitel`}
2. `Buch.Schriftsteller` ≅ `Fachbuch.Autor`
3. `Buch.Jahr` ≅ `Fachbuch.Jahr`
4. `Buch.ISBN` ≅ `Fachbuch.InternationaleStandardbuchnummer`
5. `Buch.Verlag` ≅ `Fachbuch.VerlegtBei`

Das erste Mappingelement `Buch.Titel` ≅ {`Fachbuch.Titel, Fachbuch.Untertitel`} bildet die Beziehung von `Buch.Titel` auf `Fachbuch.Titel` bzw. `Fachbuch.Untertitel` ab. Dabei ist zu erwähnen, dass Untertitel ein Hyponym[3] von Titel ist und damit also Titel ein Hyperonym[4] von Untertitel ist, was von einem Matchalgorithmus zunächst erkannt werden muss.
Das zweite Mappingelement `Buch.Schriftsteller` ≅ `Fachbuch.Autor` zeigt die Beziehung zwischen `Buch.Schriftsteller` und `Fachbuch.Autor`, wobei Schriftsteller und Autor Synonyme voneinander sind. Letztere Information kann einem Matchalgorithmus z.B. in Form einer Synonymtabelle vorliegen.
Mit `Buch.Jahr` ≅ `Fachbuch.Jahr` wird eine Verbindung zwischen `Buch.Jahr` und `Fachbuch.Jahr` angegeben.
Das Mappingelement `Buch.ISBN` ≅ `Fachbuch.InternationaleStandardbuchnummer` zeigt den Zusammenhang von `Buch.ISBN` und `Fachbuch.InternationaleStandardbuchnummer`. Auch die Information, dass ISBN die geläufige Abkürzung von Internationale Standardbuchnummer ist, muss einem Matchansatz zusätzlich durch beispielsweise ein Abkürzungsverzeichnis zugänglich gemacht werden.
Das letzte Mappingelement `Buch.Verlag` ≅ `Fachbuch.VerlegtBei` zeigt die Beziehung zwischen `Buch.Verlag` und `Fachbuch.VerlegtBei` auf, wobei dazu erkannt werden muss, dass Verlag und verlegt von demselben Wort abstammen und somit wahrscheinlich die gleichen Daten repräsentieren.

Zusätzliche Informationen, die Korrespondenzen besitzen können, sind Plausibilitätswerte und Mappingausdrücke. Der Plausibilitätswert beruht auf der Ähnlichkeit

[3]Mit Hyponym wird in der Linguistik der Unterbegriff eines Begriffs bezeichnet.
[4]Mit Hyperonym wird in der Linguistik der Oberbegriff eines Begriffs bezeichnet.

der Schemaelemente. Je ähnlicher diese einander sind, desto wahrscheinlicher ist eine Beziehung zwischen ihnen. Als Wertebereich ist [0,1] verbreitet, wobei 0 auf sehr unterschiedliche und 1 auf sehr ähnliche Elemente hinweist. Ein *Mappingausdruck* spezifiziert die Transformation, die ausgeführt werden muss, um das eine Element aus dem anderen zu erlangen.Der Mappingausdruck kann gerichtet oder ungerichtet sein. Im Gegensatz zu der vagen Information eines Ähnlichkeitswertes, ist der Mappingausdruck mathematisch exakt. Die existierenden automatischen Ansätze fokussieren zumeist Ähnlichkeitswerte, wobei natürlich bei manuellen Ansätzen auch Transformationsregeln spezifizierbar sind.

Die Mappingausdrücke zu den oben aufgeführten Mappingelementen lauten folgendermaßen:

1. `Fachbuch.Titel, Fachbuch.Untertitel` = `Extrahiere(Buch.Titel,...)`
2. `Fachbuch.Autor` = `Buch.Schriftsteller`
3. `Fachbuch.Jahr` = `Buch.Jahr`
4. `Fachbuch.InternationaleStandardbuchnummer` = `Buch.ISBN`
5. `Fachbuch.VerlegtBei` = `Buch.Verlag`

Der erste Mappingausdruck `Fachbuch.Titel, Fachbuch.Untertitel = Extrahiere(Buch.Titel,...)` gibt an, dass man aus dem `Buch.Titel` durch Teilung `Fachbuch.Titel` und `Fachbuch.Untertitel` erhält. Dieser Ausdruck ist gerichtet, da die Funktion Extrahiere nur genutzt werden kann, um `Fachbuch.Titel` zu erhalten. Will man jedoch `Buch.Titel` erhalten, so müssen `Fachbuch.Titel` und `Fachbuch.Untertitel` durch eine Funktion Verkette miteinander verbunden werden. Die restlichen Mappingausdrücke stellen eine einfache Abbildung dar, die ungerichtet ist. Es muss also nur ein Kopieren und Einfügen der Inhalte stattfinden, die in beide Richtungen so auszuführen ist.

Das Element `E002Z` im Schema *S2* besitzt einen unverständlichen Elementnamen und der Datentyp allein reichte nicht aus, um eine sinnvolle Korrespondenz zu finden. Während also alle Elemente von *S1* einen Matchpartner besitzen, ist dies bei Schema *S2* nur bei fünf von sechs Elementen der Fall.

1.3 Anwendungsgebiete

In [RB01] wird beschrieben, wie Schema-Matching ein grundlegendes Problem sowohl in der Schemaintegration, Data Warehouses, E-Business und auch in der semantischen Anfrageverarbeitung ist. Im folgenden sollen diese kurz umrissen werden.

Schemaintegration

Das Problem der Schemaintegration wird seit Anfang 1980 erforscht, siehe z.B. [BLN86b] und [BC86]. Gegeben ist dabei eine Menge von Schemata, die getrennt voneinander entstanden sind; gesucht wird eine globale Sicht für diese. Durch die getrennte Entwicklung unterscheiden sich die Schemata in Struktur sowie Terminologie und das nicht nur, wenn sie aus verschiedenen Bereichen kommen. Selbst wenn die Schemata in den gleichen Bereichen entstanden sind, so hat jeder Entwickler seine eigene Umgebung und einen eigenen Stil.
Ein erster Schritt, um die Schemata zu integrieren, ist das Identifizieren und Charakterisieren der Beziehungen zwischen ihnen. Das ist ein Prozess des Schema-Matchings. Ist dieser abgeschlossen und sind die Beziehungen gefunden, so können die Mappingelemente zu einem kohärenten, integrierten Schema bzw. Sicht vereinigt werden. Eine Variation dieses Problems der Schemaintegration tritt auf, wenn ein unabhängig entwickeltes Schema in ein existierendes konzeptionelles Schema integriert werden soll.

Data Warehouse

Eine Variante des Problems der Schemaintegration tauchte Anfang der 90er auf (z.B. [Inm92]). Es beinhaltet die Integration von Datenquellen in ein Data Warehouse. Ein Data Warehouse ist nach [AH00] eine physische Datenbank, die eine integrierte Sicht auf beliebige Daten darstellt, um Analysen zu ermöglichen. Für diese Sicht werden die Daten der unterschiedlichen Datenquellen extrahiert und von dem dortigen Format in das Warehouseformat transformiert. Wie in [BR00] gezeigt wird, ist die Matchoperation nützlich für das Erstellen von Transformationen. Ist eine Datenquelle gegeben, so beginnt ein Ansatz mit dem Heraussuchen der Elemente, die bereits im Warehouse vorhanden sind. Das ist eine Matchoperation. Darauf aufbauend, können die Entwickler Transformationsregeln erstellen, um die Daten der Quelle mit dem Warehouse abzugleichen.

E-Business

In der letzten Zeit hat E-Business eine neue Motivation für das Schema-Matching gebracht: den Nachrichtenaustausch (unter anderem [PVM+02], [LYHY02]). Viele Geschäftspartner tauschen häufig Nachrichten auf elektronischem Wege aus, die Geschäftstransaktionen beschreiben. Normalerweise hat dabei jeder Partner sein eigenes Nachrichtenformat. Ein Mehr an Geschäftspartnern bedeuten dann meist auch ein Mehr an Formaten, die sich in ihrer Syntax und auch in den benutzen Nachrichtenschemata unterscheiden können.
Um ein System zu befähigen, Nachrichten auszutauschen, müssen Anwendungsentwickler die Nachrichten zwischen den Formaten und damit zwischen Nachrichtenschemata umwandeln. Die Schemata benutzen möglicherweise andere Namen, unterschiedliche Datentypen und verschieden zulässige Wertebereiche. Elemente sind außerdem

zu Strukturen gruppiert, die wiederum von Schema zu Schema different sein können. Die Umformung zwischen Nachrichtenschemata ist zum Teil ein Schema-Matching-Problem, das heutzutage noch oft von den Entwicklern selbst erledigt werden muss.

Semantische Anfrageverarbeitung

Semantische Anfrageverarbeitung (Semantic query processing) stellt ein etwas anderes Szenario als die bisher genannten dar, da es erst zur Laufzeit abläuft und nicht - wie bei den oben genannten Anwendungen - schon zur Entwicklungszeit. Ein Benutzer spezifiziert dabei die gewünschte Ausgabe einer Anfrage (Query) und das System muss selber herausfinden, wie es diese Ausgabe erzeugen kann. Die Spezifikationen werden vom Benutzer in Form von ihm bekannten Konzepten angegeben, welche nicht unbedingt die gleichen z. B. Elementnamen beinhalten wie im Datenbankschema. Deswegen muss im ersten Schritt der Anfrageverarbeitung das System die benutzerdefinierten Konzepte auf die Schemaelemente der Datenbank abbilden. Dies ist natürlich auch eine Anwendung der Matchoperation, siehe beispielsweise [MHH00].

1.4 Zielsetzung der Arbeit

Das Ziel dieser Arbeit ist aufzuzeigen, wie bereits erstellte Mappings genutzt werden sollen, um neue Mappings zu erzeugen und so zur Erkenntnisgewinnung herangezogen werden. Dazu werden verschiedene Varianten vorgestellt und implementiert. Durch eine Evaluation mit verschiedenen Testszenarien soll die Qualität beleuchtet und ausgewertet werden, wobei das Hauptziel ist, Aussagen darüber machen zu können, wie die Strategien mit unterschiedlichen Konfigurationen welche Ergebnisse bringen. Zusätzlich erfolgt ein Vergleich mit anderen bisherigen Strategien. Außerdem wird die Ausführungszeit betrachtet, die in der Praxis neben den Ergebnissen eine weitere Rolle spielt. Eine praktische Umsetzung erfolgte in den Prototyp COMA++, um neben der Auswertung den Ansatz auch für Benutzer verfügbar zu machen. Bei der Implementierung werden außerdem aus der Evaluierung gewonnene Erkenntnisse genutzt. Darüber hinaus wurde für die Handhabung des Prototyps eine GUI entworfen, die sowohl zur Evaluierung als auch zur Anwendung der bereits vorhandenen und hinzugefügten Funktionalität dient.

1.5 Gliederung

Die Arbeit ist wie folgt gegliedert: In Kapitel 2 wird eine Klassifikation von Matchalgorithmen vorgestellt und dabei die einzelnen Kriterien näher erläutert. In Kapitel 3 wird der Prototyp COMA++ mit seiner Architektur, dem Ablauf zur Erzeugung von Mappings und den implementierten Matchern beschrieben. Ein Überblick über Wiederverwendungsstrategien wird in Kapitel 4 gegeben und zusätzlich eine Erläuterung zu den Implementation in den Prototyp COMA. Die im Rahmen dieser Arbeit

umgesetzten Weiterentwicklungen zur Mapping-Wiederverwendung werden in Kapitel 5 behandelt. Die Evaluation und Auswertung der Wiederverwendungsstrategien erfolgen in Kapitel 6. In Kapitel 7 wird auf die grafische Oberfläche des Prototyps COMA++ eingegangen und neben den Komponenten auch einige Anwendungsszenarien dargestellt. In einem letzten Abschnitt wird in Kapitel 8 eine Zusammenfassung der Ergebnisse und ein Ausblick auf die weitere Entwicklung gegeben.

2 Klassifikation von Matchansätzen

Dieses Kapitel beruht auf [RB01], in der eine Klassifizierung der Hauptansätze im Schema-Matching eingeführt wird. Die Abbildung 2.1 stellt einen Überblick der möglichen Klassifikationskriterien dar.

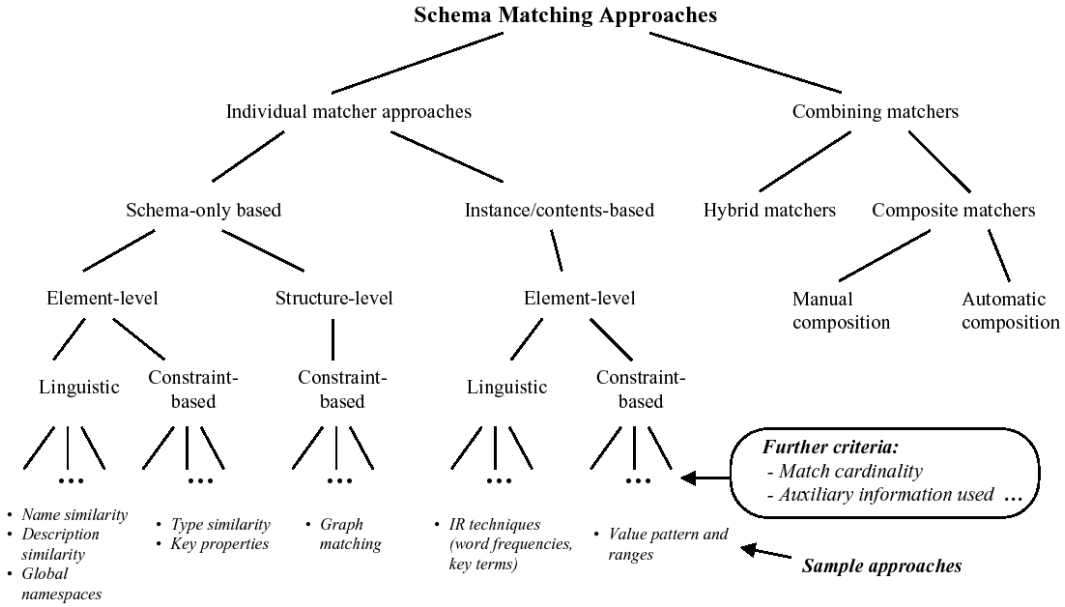


Abbildung 2.1: Klassifikation der Ansätze für das Schema-Matching nach [RB01]

Matchansätze werden zunächst danach unterschieden, ob sie einzelne Algorithmen verwenden oder mehrere Algorithmen kombinieren. Für die Einzelmatcheransätze (individual matcher approaches) werden folgende Klassifikationskriterien berücksichtigt:

- *Instanz vs. Schema:* Matchansätze können Instanzdaten (Abschnitt 2.2) oder nur Schemainformationen (Abschnitt 2.1) verwenden.
- *Element vs. Struktur:* Die Berechnung kann für einzelne Elemente, wie Attributen, oder komplexen Schemastrukturen durchgeführt werden. In den Abschnitten 2.2 und 2.1 wird darauf näher eingegangen.
- *Linguistisch vs. Bedingung:* Das Matchen kann mit Hilfe von linguistischen Methoden, die sich beispielsweise auf Elementnamen basieren, oder von bedingungs-

basierten Techniken, die z.B. Beziehungsarten und Wertebereiche betrachten, erfolgen (siehe Abschnitte 2.2 und 2.1).

- *Zusatzinformation:* Viele Matchtechniken verwenden neben den Schemata bzw. Instanzdaten noch zusätzliche Informationen wie Wörterbücher und globale Schemata (Abschnitt 2.3).

- *Matchkardinalität:* Das Gesamtmatchergebnis kann ein oder mehrere Elemente des einen Schemas in Beziehung zu einem oder mehreren Elementen des anderen Schemas setzen. Dies führt zu vier Beziehungsarten: 1:1, 1:n, n:1, n:m, die in Abschnitt 2.4 erklärt werden.

Die kombinierenden Ansätze, die in Abschnitt 2.5 näher erläutert werden, sind zum einen die *Hybridmatcher*, die mehrere Matchkriterien verwenden, wie z.B. Name und Datentyp. Zum anderen sind es die *zusammengesetzte Matcher*, die mehrere Matchergebnisse von unterschiedlichen Matchalgorithmen zu einem Gesamtergebnis kombinieren.

2.1 Schemabasierte Ansätze

Wenn ein Einzelmatcher nur Schemainformationen für die Berechnung nutzt, aber keine Instanzdaten, dann ist er *schemabasiert* (schema-only based approach). Diese Schemainformationen sind beispielsweise Eigenschaften eines einzelnen Schemaelements wie Elementnamen und Datentypen, aber auch bezüglich der Struktur des Schemas. Prototypen, die auf schemabasierten Ansätze beruhen, sind COMA [DR02], Cupid [MBR01], SF (Similarity Flodding) [MGMR02] und TranScm [MZ98].

2.1.1 Elementbasierte Ansätze

Wird die Berechnung auf den einzelnen Elementen ausgeführt, so spricht man von einem *elementbasierten* Ansatz (element-level approach). Einzelne Elemente sind beispielsweise Attribute in einem XML-Schema oder Spalten in einem Relationalen Schema. Für die Tabelle 2.1 sind `Buch.Jahr` und `Verlag.Verlagsname` zwei Elemente. Das Matchen der Elemente ist sinnvoll, da sie die kleinsten Komponenten eines Schemas sind und so das gesamte Schema abgedeckt wird. Von Vorteil ist dies besonders, wenn die Strukturen in den zu matchenden Schemata sehr unterschiedlich sind und auf diese Weise trotzdem korrekte Korrespondenzen gefunden werden.

Ein häufig benutzter Ansatz ist die Anwendung von linguistischen Methoden (linguistic-based approach) , die textuelle Komponenten der Elemente, wie Namen und Beschreibungen, vergleichen, um die Elementähnlichkeit zu ermitteln. Dazu kann ein Matchalgorithmus die Zeichenkette in eine kanonische Form bringen durch Stammbestimmung und Spaltung in Wortteile, z.B. Fachbuch→Fach, buch. Die Ähnlichkeit von Worten lässt sich etwas anhand der Editierdistanz oder mit n-Grams [BS01] ermitteln. Die Editierdistanz (die Levenshtein-Distanz[LAFP66]) entspricht dabei der Anzahl der

Elemente von Schema *S1*	Elemente von Schema *S2*
Buch - Titel *varchar (300)* - Schriftsteller *varchar (100)* - Jahr *int* - ISBN *varchar (10)* - VlgNr *int* - Preis *decimal* - MwSt *decimal* Verlag - VlgNr *int* - Verlagsname *varchar (50)*	Fachbuch - Titel *varchar (500)* - Untertitel *varchar (100)* - Autor *varchar (100)* - Jahr *int* - InternationaleStandardbuchnummer *varchar (10)* - VerlegtBei *varchar (100)* - Kosten *decimal* - E002Z *varchar (100)*

Tabelle 2.1: Matchbeispiel

Editieroperationen (Einfügen, Löschen, Ändern), die mindestens benötigt werden, um eine Zeichenkette in eine andere zu ändern. Um die Zeichenkette Verlegt in Verlag zu ändern, werden zwei Operationen benötigt: Löschen vom t (Verlegt→Verleg) und Änderung von e zu a (Verleg→Verlag). Bei der Verwendung von n-Grams werden Worte in alle möglichen Substrings der Länge n zerlegt. Für das Wort Verlag ergeben sich vier Trigrams (n-Grams mit $n = 3$): Ver, erl, rla, lag. Für die Berechnung der Ähnlichkeit werden die n-Grams der Wörter miteinander verglichen und je mehr Substrings in beiden Wörtern vorhanden sind, desto ähnlicher sind sie sich. Verlag und Verlegt haben immerhin zwei gemeinsame Trigrams , nämlich Ver und erl.
Außerdem kann eine Matchtechnik Synonyme und Hyperonyme bzw. Hyponyme mit Hilfe von generischen und bereichsspezifischen Thesauren identifizieren. Für das Matchbeispiel aus Tabelle 2.1 könnten mit Hilfe einer sprachbasierten Methode so die Synonyme Schriftsteller und Autor erkannt werden und durch Stammbestimmung die Verwandtschaft der Wörter Verlag und Verlegt ermittelt werden.
Für den Vergleich von Textbeschreibungen können Techniken des Information Retrieval (IR) genutzt werden. Die sprachbasierten Ansätze ermöglichen es dem Computer, mehr Informationen aus Namen und Texten herauszuholen, als es nur aufgrund der Zeichenkette möglich wäre, deren Bedeutung einem Computer nicht bekannt ist.

Schemata besitzen meistens die Möglichkeit, verschiedene Bedingungen zu definieren, wie Datentypen, Wertebereiche, Schlüssel, Beziehungsarten, usw. definieren. Wenn beide Schemata des Matchproblems diese Informationen besitzen, können sie genutzt werden, um die Ähnlichkeit von Schemaelementen zu bestimmen. Dies ist der *bedingungsbasierte* Ansatz (constraint-based approach).
In dem Beispiel der Tabelle 2.1 legt die Typinformation nahe, dass `Buch.Preis` und `Buch.MwSt` aus *S1* das Element `Fachbuch.Kosten` matchen, da beide decimal als Datentyp haben. Diese Korrespondenzen wären korrekt. Mit der Typinformation als Kri-

terium matchen sowohl `Buch.Jahr` als auch `Buch.VerlagNr` das Element `Fachbuch.Jahr`, wobei nur ersteres stimmt. Den Datentyp varchar gibt es bei beiden Schemata mehrfach und es würde der bedingungsbasierte Ansatz zu vielen Korrespondenzen führen, von denen allerdings ein großer Teil falsch wäre.

2.1.2 Strukturbasierte Ansätze

Ein *strukturbasierter* Ansatz (structure-level approach) betrachtet die Nachbarschaft von Elementen, um die Elementähnlichkeit zu berechnen. Das Betrachten der Struktur ist lohnenswert, wenn Elemente ähnliche Strukturen besitzen, die aber beispielsweise unterschiedliche Namen haben. Es sind verschiedene Fälle möglich, die abhängig davon sind, wie komplett und genau der Strukturmatch benötigt wird. Im Idealfall matchen alle Komponenten der zwei Schemastrukturen einander (*full structural match*). In anderen Fällen werden jedoch nur einige der Komponenten einen Matchpartner besitzen (*partial structural match*). Dies tritt z. B. auf, wenn beide Schemastrukturen aus unterschiedlichen Bereichen stammen. In Tabelle 2.2 sind Beispiele für den vollen bzw. partiellen Strukturmatch dargestellt.

S1-Elemente	*S2*-Elemente	
Buch - Titel - Schriftsteller - Jahr - ISBN	Fachbuch - Titel - Autor - Jahr - InternationaleStandardbuchnummer	Voller Strukturmatch von Buch und Fachbuch
Buch - Titel - Schriftsteller - Jahr - ISBN	Buchpreis - Titel - Autor - Auszeichnung	Partieller Strukturmatch von Buch und Buchpreis

Tabelle 2.2: Beispiele für einen vollen und einen partiellen Strukturmatch

2.2 Instanzbasierte Ansätze

Die *instanzbasierten* Ansätze (instance/contents-based approach), wie z.B. LSD [DDL00], SemInt [LC00], GLUE [DMDH02] und Autoplex [BM01] arbeiten mit konkreten Datensätzen eines Schemas. Instanzen sind dabei beispielsweise der Inhalt von Tabellen. Für das Schema `Buch` mit seinen vier Elementen `Titel, Autor,Jahr` und `ISBN` stehen Instanzdaten - in diesem Fall Daten über 2 Bücher - in Tabelle 2.3.

Instanzen sind besonders hilfreich, wenn nützliche Schemainformationen begrenzt sind. Im Extremfall ist gar kein Schema gegeben, und es muss eines aus den Instanz-

Fachbuch

Titel	Data Integration in the Life Sciences	Mehrrechner-Datenbanksysteme
Untertitel		Grundlagen der verteilten und parallelen Datenbankverarbeitung
Autor	Erhard Rahm	Erhard Rahm
Jahr	2004	2002
Internationale-Standardbuchnummer	3540213007	3486243632
VerlegtBei	Springer	Addison-Wesley
Kosten	EUR 44,94	EUR 59,70
E002Z	237B	150A

Tabelle 2.3: Instanzbeispiel zum Schema `Fachbuch`

daten konstruiert werden. Dabei ist jedoch zu bedenken, dass Instanzdaten auch nur einen Teil des Schemas abdecken können. Ein aus ihnen konstruiertes Schema wird also nicht das gesamte dazugehörige Schema abdecken und darauf berechnete Mappings können auch unvollständig sein.

Die Instanzdaten können einen wichtigen Einblick in den Inhalt und die Bedeutung von Schemaelementen geben. So untersucht ein *elementbasierter* Instanzmatcher (element-level approach), ob bestimmte Zeichenketten sich wiederholen (z.B. EUR in den Instanzen vom Element `Kosten` der Tabelle 2.3) oder nur bestimmte Zeichen verwendet werden (z.B. nur Zahlen in den Instanzen von `InternationaleStandardbuchnummer` der Tabelle 2.3).
Darüber hinaus können diese Daten helfen, falsche Interpretationen der Schemainformation aufzudecken. Dies ist der Fall, wenn Elemente zwar ähnlich heißen und so eine Korrespondenz wahrscheinlich ist, aber die Elemente vollkommen unterschiedliche Instanzwerte besitzen und so die Korrespondenz als falsch erkannt werden kann.

Ferner können hier - ähnlich wie bei schemabasierten Ansätzen - ebenfalls linguistische Methoden eingesetzt werden. Ein instanzbasierter Ansatz basiert dabei seine Untersuchung auf den vorhandenen Werten für ein Schema - den Instanzen. So kann er neben den Elementnamen Schlüsselwörter oder Themen basierend auf Worthäufigkeiten und Wortkombinationen für z. B. `Fachbuch.Titel` und `Fachbuch.Untertitel` aus Tabelle 2.3 extrahieren.

Bei der Nutzung von Instanzdaten kann eine bedingungsbasierte Technik durch die Auswertung der Daten Bedingungen für ein Schema ermitteln. Diese beinhaltet beispielsweise die Bestimmung von Wertebereichen, die Ermittlung von Zeichenformaten und die Berechnung von Wertekennzahlen wie z. B. Minimum, Mittelwert. So bestehen beispielsweise Instanzen für das Element `InternationaleStandardbuchnummer` aus Tabelle 2.3 aus einer zehnstelligen Zahl. Die Berechnungen sind natürlich umso

sinnvoller, je mehr Instanzdaten vorhanden sind. Auf diese Weise werden Bedingungen gefunden, die der Entwickler eventuell gar nicht im Schema festgelegt hat, mit denen aber Korrespondenzen sicherer als korrekt oder falsch bestimmt werden können.

2.3 Zusatzinformation

Ein Matchansatz benutzt *Zusatzinformation*, sobald er irgendeine andere Information benutzt als die beiden Schemata bzw. die Instanzdaten. Zusatzinformation ist oft unerlässlich, um das Wissen von Menschen nachzubilden und so auch z.B. Synonyme und Bedeutungsunterschiede zu finden, die ein Computer von sich aus nicht erkennt. Um die Matchqualität zu steigern, werden daher von Matchansätzen zusätzliche Informationen, wie beispielsweise Wörterbücher und globale Schemata, einbezogen. In diesen Zusammenhang fällt auch die Wiederverwendung von Schemata und Mappings, da sie zusätzliche Informationen beinhalten. Dieses Thema wird in Kapitel 4 vertieft.

2.4 Matchkardinalität

Die *Matchkardinalität* (matching cardinality) als ein weiteres Kriterium gibt an, ob im Matchergebnis ein oder mehrere Elemente des einen Schemas mit einem oder mehreren Elementen des anderen Schemas in Verbindung stehen. Es gibt dabei vier Fälle: 1:1, 1:n, n:1 und n:m, für die in Tabelle 2.4 Beispiele aufgeführt sind. Es kann natürlich auch Elemente geben, die gar nicht in eine Verbindung eingehen. Diese Elemente werden nicht im Matchergebnis aufgeführt, da für sie keine Korrespondenz existiert.
So bedeutet der Fall 1:n, dass ein Element des ersten Schemas mit n Elementen des zweiten Schemas verbunden ist. Ein Mappingausdruck gibt an, wie die Elemente miteinander zusammenhängen. Ein typisches Beispiel ist der "Name" eines Kunden in einer Kundendatenbank, der als ein einzelnes Attribut den vollständigen Namen enthalten kann oder aber auch auf zwei Attribute aufgeteilt sein kann. So ergibt sich im zweiten Schema von Tabelle 2.4 `FirstName` und `LastName` durch das Extrahieren aus dem Element `Name` vom ersten Schema.
Um ein n:m Mapping zu erlangen, benötigen die Elemente normalerweise eine strukturelle Einbettung und somit ein Matching auf Strukturebene. So entspricht in Tabelle 2.4 das n:m Matching auf der Elementebene einem n:1 Matching auf der Strukturebene. Das n:m Matching bezieht sich dabei auf die vier Attribute des ersten Schemas `B.Title, B.PuNo, P.PuNo, P.Name`, welche die zwei Attribute des zweiten Schemas `A.Book, A.Publisher` matchen. Auf der Strukturebene steht ein Matching der zwei Tabellen `B` und `P` zur Tabelle `A` dahinter.
In der Praxis ist es durchaus üblich, nur 1:1 Matches im Ergebnis haben zu wollen. Dies hat den Vorteil, dass die Ergebnismenge nur eine bestimmte Größe erreicht und zu jedem Element nur das Element zugeordnet wurde, dass am wahrscheinlichsten ist. Dabei gehen jedoch Korrespondenzen verloren, wenn ein Element aber zwei oder mehreren Elementen zugeordnet werden muss, wie in Tabelle 2.1, in welchem `Buch.Titel`

lokale Match-kardinalität (Elementebene)	Elemente von Schema *S1*	Elemente von Schema *S2*	Mappingausdruck
1:1	Buch.Jahr	Fachbuch.Jahr	Fachbuch.Jahr = Buch.Jahr
n:1	Buch.Preis, Buch.MwSt	Fachbuch.Kosten	Fachbuch.Kosten = (1+Buch.MwSt)* Buch.Preis
1:n	Buch.Titel	Fachbuch.Titel, Fachbuch.Untertitel	Fachbuch.Titel, Fachbuch.Untertitel = Extract(Buch.Titel,...)
n:m	Buch.Schriftsteller, Buch.VlgNr, Verlag.VlgNr, Verlag.Verlagsname	Fachbuch.Autor, Fachbuch.Verlag	Fachbuch.Autor, Fachbuch.Verlag = Select Buch.Schriftsteller, Verlag.Verlagsname From Buch, Verlag Where Buch.VlgNr = Verlag.VlgNr

Tabelle 2.4: Matchkardinalitäten, die sich auf das Matchproblem $S1 - S2$ aus Tabelle 2.1 beziehen

vom Schema *S2* sowohl die Informationen des Elementes `Fachbuch.Titel` als auch `Fachbuch.Untertitel` beinhaltet. Für solche Fälle ist es empfehlenswert, auch 1:n und n:1 als Matchkardinalität zuzulassen.
Die Matchkardinalität hat auch einen Einfluss auf die Evaluation von Matchergebnissen. Es spielt eine Rolle bei der Auswertung, ob ein Match von n zu m Elementen als eine Korrespondenz mit der Matchkardinalität n:m angesehen wird oder beispielsweise als viele einzelne Korrespondenzen mit der Matchkardinalität 1:1. Im ersten Fall müssen wirklich alle n bzw. m Elemente mit den Matches gefunden werden, damit die Korrespondenz als korrekt gefunden gilt.

2.5 Kombinierende Matcher

Einzelne Matchtechniken sind eingeschränkt, da sie nur einen Teil der vorhanden Information nutzen und daher oft unerwünschte Ergebnisse liefern. Aus diesem Grund werden Matchansätze kombiniert, so dass die Teilmatcher die Schwächen untereinander ausgleichen, möglichst ohne dabei ihre Stärken einzubüßen. Damit wird ein Endergebnis erreichen, dass qualitativ sehr hochwertig ist. Zur Kombination von Matchansätzen können sowohl *Hybridmatcher* (hybrid matcher) als auch *zusammengesetzte Matcher* (composite matcher) verwendet werden.

2.5.1 Hybridmatcher

Zur Kombination kann man einerseits Hybridmatcher verwenden, ein Ansatz auf dem Cupid [MBR01] basiert. Es werden dabei mehrere Matchkriterien, wie z. B. Name- und Typgleichheit, überprüft. Hybridmatcher liefern mit geringerem Aufwand bessere Kandidaten als bei einer getrennten Ausführung der entsprechenden Einzelmatcher. Der Aufwand ist geringer, da nicht für jedes Kriterium ein Durchlauf aller möglichen Kombinationen erfolgen muss, sondern gleich mehrere Kriterien, wie z.B. Ähnlichkeit der Namen und Datentypen, auf einmal getestet werden können. Durchläufe werde so eingespart, wobei die Berechnungen selber natürlich noch stattfinden.
Die Effektivität lässt sich erhöhen, da ungeeignete Matchkandidaten, die keinem oder nur einem Matchkriterium entsprechen, frühzeitig herausgefiltert werden können. So braucht man Elemente nur auf ihre Struktur hin vergleichen, wenn sie schon eine gewisse Ähnlichkeit im Namen bzw. Pfad haben. Es werden so Matchberechnungen eingespart, ohne jedoch das Ergebnis zu verschlechtern.
Bei den Hybridmatchern werden typischerweise Kombinationen von bestimmten Matchtechniken verwendet, die gleichzeitig oder in einer festen Ordnung ausgeführt werden. Da die Kombination meist festverdrahtet ist, sind Hybridmatcher im allgemeinen unflexibel.

2.5.2 Zusammengesetzte Matcher

Zum anderen gibt es zusammengesetzte Matcher, die mehrere Matchergebnisse anhand von Algorithmen kombinieren. Dieser Ansatz wird von COMA [DR02] umgesetzt. Zuerst werden also einzelne Matchergebnisse mit Einzelmatchern berechnet und dann ein Gesamtergebnis erzeugt, beispielsweise durch die Auswahl nur solcher Korrespondenzen, die in jedem Einzelergebnis auftreten.
Die Möglichkeit zur Kombination von Matchmethoden ist bei zusammengesetzten Matchern flexibler als bei Hybridmatchern, da sie die Wahl aus einer Menge von modularen Matchansätzen ermöglichen. Hinzukommend ist dann noch festzulegen, ob die Ausführung gleichzeitig oder nacheinander ablaufen soll. Erfolgt die Ausführung nacheinander, so kann das Ergebnis von der ersten Matchtechnik als Input für die zweite gegeben werden. Diese erweitert das erhaltene Ergebnis, und es erfolgt eine iterative Verbesserung des Mappings, indem zum Beispiel im ersten Durchlauf ermittelte Korrespondenzen mit Hilfe eines instanzbasierten Ansatzes durchgeht und diejenigen bevorzugt, deren Instanzen einander ähnlicher sind.

3 COMA++ Ansatz

Es wurden viele Prototypen entwickelt und getestet, die oft auf den in Kapitel 2 erwähnten Ansätzen aufbauen. Diese wurden auch schon größtenteils in [RB01] vorgestellt und in [DMR03] verglichen.
Dieser Diplomarbeit geht nur auf COMA bzw. COMA++ ein, da alle Implementierungen und Erweiterungen an diesem Prototyp vorgenommen wurden. Die folgenden Abschnitte sind angelehnt an [DR02], [RDM04] und [ADMR05]. Zunächst wird in Abschnitt 3.1 die Architektur des Prototypen COMA++ vorgestellt. Danach wird die Schemarepräsentation in Abschnitt 3.2, die Matcherausführung in Abschnitt 3.3 und die Kombination der Matchergebnisse in Abschnitt 3.4 erläutert. Anschließend wird in Abschnitt 3.5 auf die Matcher der Matcherbibliothek und in Abschnitt 3.6 auf die implementierten Matchstrategien eingegangen.

3.1 Architektur

Der Prototyp COMA++ basiert auf dem Prototypen COMA, welcher in [DR02] wird COMA (Combining Match Algorithm) vorgestellt wird. COMA++ ist ein generisches Matchsystem. Wie COMA verfolgt es ebenfalls einen zusammengesetzten Ansatz, bei welchem verschiedene Matcher und unterschiedliche Kombinationsmöglichkeiten für die Matchergebnisse genutzt werden können. Wie in Abschnitt 1.2 erklärt wurde, werden für eine Matchoperation zwei Schemata als Eingabe genommen und unter möglicher Verwendung von Zusatzinformationen ein Mapping als Ausgabe berechnet.

In Abbildung 3.1 nach [ADMR05] ist die Architektur von COMA++ zu sehen, die aus fünf Komponenten besteht: Schema Pool, Match Customizer, Mapping Pool, Execution Engine und Repository. Ihre Funktion und das Zusammenspiel zwischen ihnen soll im Folgenden dargelegt werden.
Zuerst werden **Schemata** von externen Quellen, wie beispielsweise relationalen Datenbanken oder XML-Dateien, importiert. Sie werden dazu auf einen gerichteten azyklischen Graphen abgebildet. Die Graphen werden sowohl für die Visualisierung genutzt als auch auf ihnen die Matcher angewendet. Für XML ist neben dem DTD-Import auch eine erweiterte XSD-Unterstützung implementiert. **Ontologien** werden mit den W3C-Formate OWL-Lite und RDF unterstützt. COMA++ benutzt intern ein generisches Datenmodell, welches in einem relationalen DBMS zur einheitlichen Unterstützung von Schemata und Ontologien implementiert wurde.
Wie auch schon in COMA, wird ein datenbankgestütztes **Repository** zur Speicherung von importierten Schemata und Ontologien genutzt. Darüber hinaus werden auch

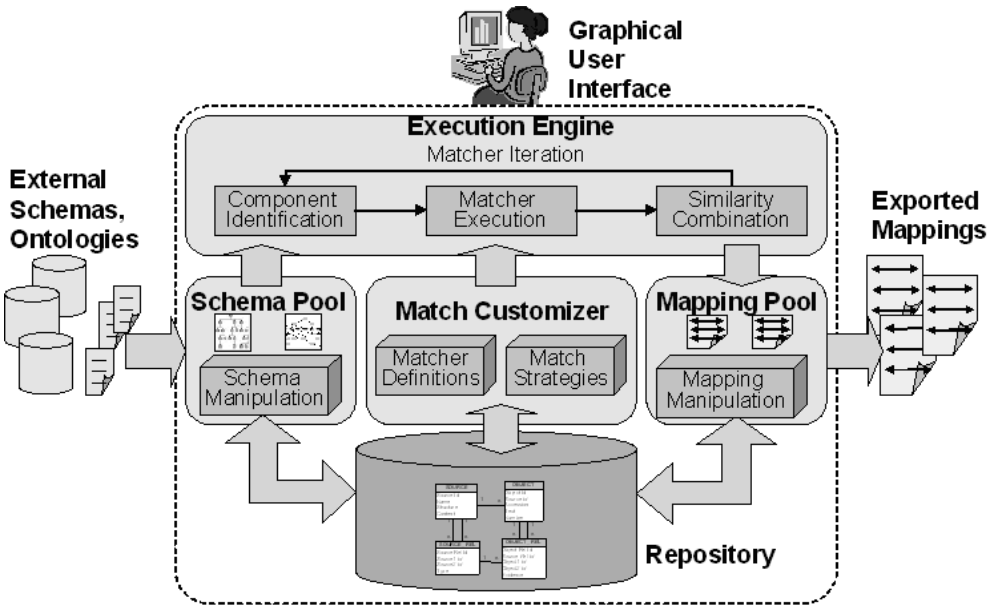


Abbildung 3.1: Architektur von COMA++

Synonyme, Abkürzungen und zuvor erzeugte Mappings dort abgelegt. Vom Repository können Ontologien und Schemata - als komplette Schemata bzw. Subschemata - für das Matchen geladen werden. Alle werden in dem **Schema Pool** verwaltet, welches den Import und das Laden vom bzw. Speichern ins Repository ermöglicht.
Der **Match Customizer** hält eine erweiterbare Bibliothek mit unterschiedlichen Matchern bereit und ermöglicht verschiedene Strategien zum Kombinieren der Matchergebnisse. Es stehen bereits eine Menge von implementierten Matchern zur Verfügung, die unter anderem auch die Wiederverwendung unterstützen. Außerdem können existierende Matcher einfach zu einem neuen, mächtigeren Matcher kombiniert werden.
Der Matchprozess läuft in der **Execution Engine** ab. An einer Matchoperation können mehrere Matcher beteiligt sein, die flexibel von der Matcherbibliothek auswählbar sind. Zuerst werden die Elemente der zu matchenden Schemata bestimmt und auf diesen die Matcher ausgeführt. Danach ergibt die Aggregation der Ähnlichkeitswerte der einzelnen Matcher unter Beachtung der Richtung ein Ranking. Jeder Match, der einen zuvor bestimmten Schwellenwert erreicht, ist im Ergebnismapping enthalten. Der Ablauf wird detaillierter in Abschnitt 3.3 und 3.4 beschrieben.
In dem **Mapping Pool** werden erzeugte oder aus dem Repository geladene Mappings verwaltet. COMA++ realisiert eine Vielzahl von Operatoren auf den Matchergebnissen, wie das Vereinigen oder Vergleichen von unterschiedlichen Mappings.
COMA++ beinhaltet eine umfassende **graphische Benutzeroberfläche** (GUI, Graphical User Interface). Der Matchprozess kann interaktiv auf verschiedenen Wegen beeinflusst werden, nämlich *vor dem Matchen* durch das Konfigurieren einer Matchstrategie, *während des Matchens* für iterative Verbesserung und *nach dem Matchen* durch die Manipulation der erhaltenen Matchergebnisse.

Die Graphische Benutzeroberfläche (Graphical User Interface = GUI) ist in Java Swing implementiert. Die GUI visualisiert Schemata und Matchergebnisse und ermöglicht einen interaktiven und iterativen Matchprozess. Sie wird in Kapitel 6 näher erläutert. Der Austausch (Import/Export) von Mappings wird durch Dateiaustausch mit einem XML/RDF und einem CSV-Format unterstützt. Dies bietet den Vorteil der datenbankunabhängigen Speicherung.

COMA++ kann - wie auch schon COMA - als Plattform zur Evaluierung unterschiedlicher Matchalgorithmen und -strategien verwendet werden. Wie bereits erwähnt, ist es möglich, existierende Matcher zu konfigurieren und interaktiv neue Matcher aus den bereits existierenden zu konstruieren.

3.2 Schemarepräsentation

Schemata werden nach dem Einlesen in die interne Darstellung - einen gerichteten[5], azyklischen Graphen[6] - transformiert. Abbildung 3.2 zeigt ein XML Schema und seine interne Repräsentation.

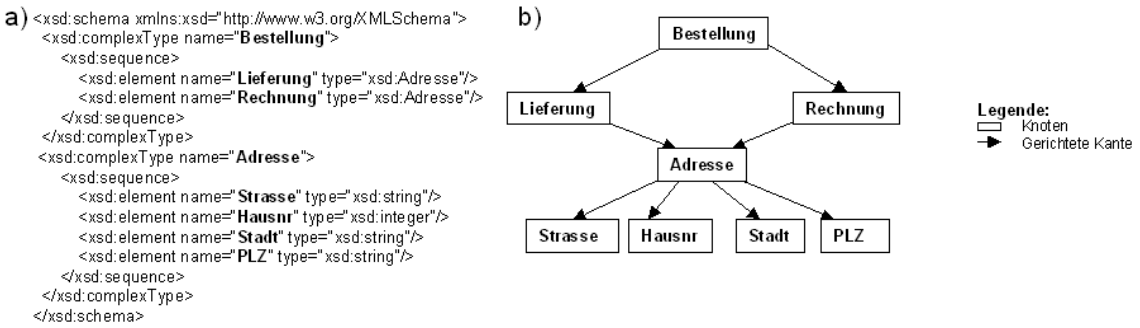


Abbildung 3.2: Die Transformation eines a) externen XML Schemas in die b) entsprechende, interne Graphrepräsentation [DR02]

In dem Graphen werden die Schemaelemente durch Graphknoten repräsentiert, die durch gerichtete Verbindungen verknüpft sind. Schemaelemente werden in dem Prototypen durch ihre Pfade repräsentiert, welche Folgen von Knoten - beginnend bei der Wurzel und endend bei dem entsprechenden Knoten - entsprechen. Für das Schemaelement `Lieferung` ist der Pfad `Bestellung.Lieferung`. Werden Elemente mehrfach verwendet (shared elements), so existieren mehrere Pfade von der Wurzel bis hin zu dem Element. Dies gilt beispielsweise für das Schemaelement `Adresse`, welches sowohl `Bestellung.Lieferung.Adresse` als auch `Bestellung.Rechnung.Adresse` als Pfade

[5]Ein Graph G heißt gerichtet, wenn er Kanten enthält, die den verbundenen Knoten eine Ordnung auferlegen (gerichtete Kanten).
[6]Ein Graph G, der keinen Kreis enthält, heißt azyklisch.

hat. Jeder Pfade eines Elementes entspricht einem Kontext, in dem es auftreten kann. Für eine Bestellung ist die Rechnungsadresse unabhängig von der Lieferungsadresse. Deswegen werden mehrfach verwendete Schemaelemente so behandelt, als würden sie mehrfache Vorkommen haben. Durch dieses Vorgehen werden unterschiedliche Schemadesigns - ob mit oder ohne mehrfache Verwendung von Elementen - vereinheitlicht. Aus diesem Grund werden im Matchprozess auch die Matchkandidaten für jeden Pfad berechnet.

Der Prototyp ist in der Lage, mit großen Schemata umzugehen, die auch über mehrere Dokumente und Namensräume verteilt sein können. Der Import eines XSD-Schema ist eine komplexe Operation, in welcher das Schema geparst und in eine konstante gerichtete Graphrepräsentation transformiert wird. Diese Aufgabe wird von einem Parser übernommen, der sowohl Schemata unterstützt, die in einer Datei oder auch in einer Sammlung von Dateien gespeichert sind. Die unterschiedlichen Designs - nämlich Elementwiederverwendung, Typwiederverwendung und Schachtelungen von Typen - werden auf eine einheitliche Struktur hin abgebildet. Außerdem werden Vorverarbeitungsschritte wie Identifizierung von Subschemata, Berechnung von Strukturstatistiken und Erkennung bzw. Entfernung von Graphzyklen durchgeführt.

3.3 Matcherausführung

In Abbildung 3.3 wird der Matchprozess gezeigt, wie er in COMA++ abläuft. Zuerst werden zwei beliebe Schemata vom Repository für eine Matchoperation geladen.

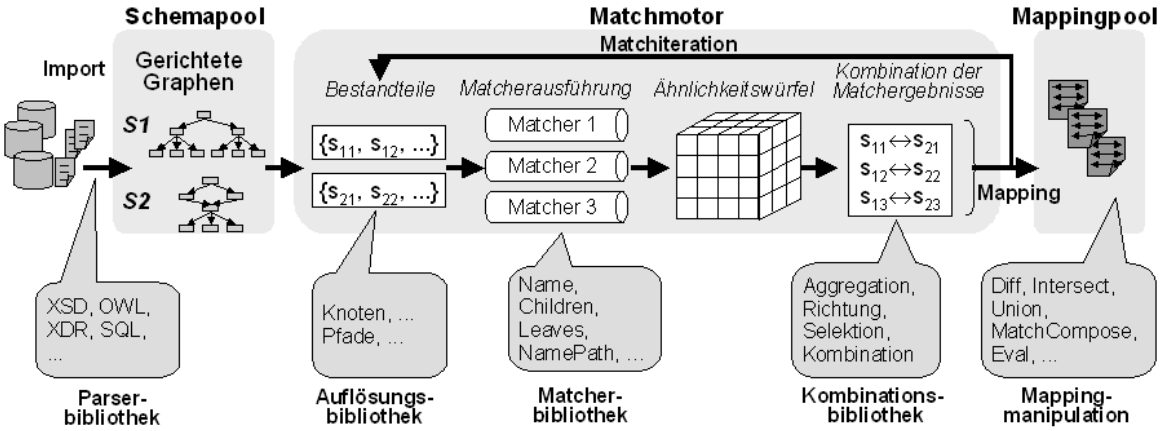


Abbildung 3.3: Matchprozess in COMA++

Der Matchprozess läuft in Matchiterationen ab, die von der Execution Engine ausgeführt werden. Eine Matchiteration besteht aus mehreren Teilschritten:

1. Bestimmung der zu matchenden Bestandteile

2. Ausführung von Matchern

3. Kombination der Matchergebnisse

Zuerst werden die Bestandteile (*Constituents*) bestimmt anhand derer die Ähnlichkeit berechnet werden soll. Die einzelnen Methoden zur Bestimmung von Komponenten eines Schemas, wie z. B. den Knoten oder den Fragmenten, aber auch von Komponenten selber, wie z. B. dem Datentyp und den Kindknoten, sind in der Auflösungsbibliothek enthalten. In Abbildung 3.1 sind einige der Pfade zu sehen, die für die Schemata aus Tabelle 2.1 bestimmt wurden.

S1-Elemente	*S2*-Elemente
Buch.Jahr	Fachbuch.Autor
Buch.ISBN	Fachbuch.Jahr
Buch.VlgNr	Fachbuch.InternationaleStandardbuchnummer

Tabelle 3.1: Auszug aus den Pfadmengen, die für die Schemata *S1* und *S2* aus Tabelle 2.1 ermittelt wurden

Der Hauptschritt während der Matchiteration ist die Ausführung von verschiedenen, voneinander unabhängigen Matchern. Die von COMA++ unterstützen Matcher lassen sich in drei Klassen einteilen: einfache, hybride und wiederverwendungsorientierte Matcher. Sie werden in Abschnitt 3.5 vorgestellt.
Jeder Matcher errechnet ein Zwischenmatchergebnis, welches Ähnlichkeitswerte zwischen 0 und 1 für jede Kombination der Elemente des Schemas *S1* und *S2* beinhaltet, die zusammen eine Ähnlichkeitsmatrix bilden. Das Ergebnis der Matcherausführung von k Matchern bei m *S1*-Elementen und n *S2*-Elementen ist ein $k \times m \times n$ Würfel von Ähnlichkeitswerten, der Ähnlichkeitswürfel. Dieser wird im Repository für die spätere Kombination und Selektion gespeichert.
In Tabelle 3.2 ist ein Auszug aus dem Ähnlichkeitswürfel zu sehen mit Werten für die Matcher Name und DataType.

3.4 Kombination von Matchergebnissen

Um die Korrespondenzen zu erhalten, erfolgt die Kombination der Ähnlichkeitswerte als dritter Teilschritt. Als Ausgangsdaten werden dabei die Zwischenmatchergebnisse der Einzelmatcher genommen, die im Ähnlichkeitswürfel gespeichert sind. In Abbildung 3.4 sind die folgenden Teilschritte dargestellt: Aggregation der matcherspezifischen Ergebnisse, Selektion der Matchkandidaten unter Beachtung der Richtung und Kombination zu einer Ähnlichkeit, wobei dieser letzte Teilschritt optional ist. Die Strategien für jeden Schritt sind in der Kombinationsbibliothek gespeichert.

Matcher	*S1*-Elemente	*S2*-Elemente	Ähnlich-keitswert
Name	Buch.Jahr	Fachbuch.Autor	0.15
		Fachbuch.InternationaleStandardbuchnummer	0.05
		Fachbuch.Jahr	1.0
DataType	Buch.Jahr	Fachbuch.Autor	0.6
		Fachbuch.InternationaleStandardbuchnummer	0.6
		Fachbuch.Jahr	1.0

Tabelle 3.2: Auszug aus dem Ähnlichkeitswürfel für das Matchproblem $S1 - S2$ aus Tabelle 2.1

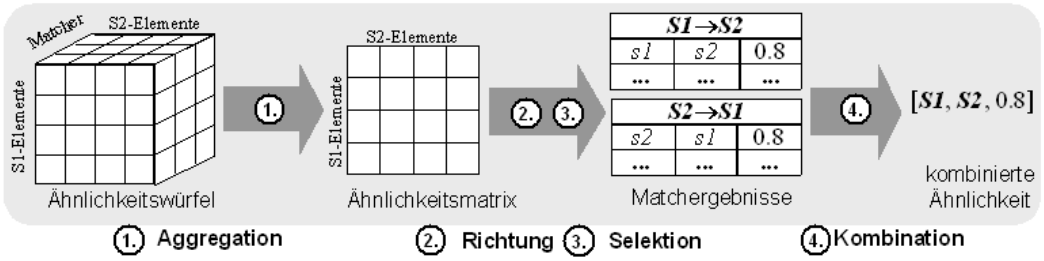


Abbildung 3.4: Die Kombination von Matchergebnissen [DR02]

Zuerst werden für jede Kombination der *S1*-Elemente mit den *S2*-Elementen die matcherspezifischen Ähnlichkeitswerte zu einem einzigen Wert aggregiert. Als Aggregationsstrategien stehen zur Verfügung: das Maximum (optimistisch), das Minimum (pessimistisch), ein gewichteter Wert oder der Mittelwert. Aus dem $k \times m \times n$ Ähnlichkeitswürfel wird so eine $m \times n$ Ähnlichkeitsmatrix ermittelt.
In Tabelle 3.3 ist ein Auszug aus der Ähnlichkeitsmatrix mit kombinierten Werten der Tabelle 3.2 zu sehen.

S1-Elemente	*S2*-Elemente	Ähnlichkeitswert
Buch.Jahr	Fachbuch.Autor	$\frac{0.15+0.6}{2} = 0.38$
	Fachbuch.InternationaleStandardbuchnummer	$\frac{0.05+0.6}{2} = 0.33$
	Fachbuch.Jahr	$\frac{1.0+1.0}{2} = 1.0$

Tabelle 3.3: Auszug aus der Ähnlichkeitsmatrix für das Matchproblem $S1 - S2$: die mit der Aggregationsstrategie Mittelwert kombinierten Werte von Tabelle 3.2

Nach der Aggregation werden die Matchvorschläge nach ihren Ähnlichkeitswerten geordnet und eine Selektionsstrategie angewendet, um die ungeeigneten Matchkandidaten auszusortieren. Das Matchergebnis dieses Teilschrittes ordnet jedem Element 0, 1 oder mehrere Matchkandidaten zu.

Als Selektionsstrategien, die wie Filter wirken, wurden folgende Möglichkeiten implementiert: Auswahl der n besten Matchkandidaten (MaxN); Auswahl der Matchkandidaten, die einen bestimmten Schwellwert überschreiten (Threshold); Auswahl des Matchkandidaten mit dem größten Ähnlichkeitswert und aller weiteren Kandidaten, deren Werte sich um höchstens δ[7] unterscheiden (MaxDelta) oder Kombinationen dieser.
Ein einzelner Ansatz liefert oft eine ungenügende Anzahl von Matchkandidaten, wobei ein niedriger Threshold eher zu viele und ein hoher Threshold bzw. MaxN und MaxDelta eher zu wenige Matchkandidaten zurückliefern. Deswegen sind Kombinationen wie Threshold+Maxn und Threshold+MaxDelta sinnvoll, um diese Effekte auszugleichen.

Für die Beispielwerte aus Tabelle 3.3 würde Threshold=0.5 für das *S1*-Element `Buch.Jahr` als Matchkandidaten `Fachbuch.Jahr` liefern.

COMA++ unterstützt die Bestimmung von *ungerichteten* und *gerichteten* Matchergebnissen. Im ungerichteten Fall werden Matchkandidaten für beide Eingangsschemata bestimmt (Both). Ein *S1*-Element *s1* wird nur als Matchkandidat für das *S2*-Element *s2* akzeptiert, wenn auch *s2* ein Matchkandidat für s1 ist.
Auf unsere Beispieldaten in Tabelle 3.3 bezogen bedeutet dies, dass `Fachbuch.Jahr` nur als Matchkandidat für `Buch.Jahr` akzeptiert wird, wenn es für `Fachbuch.Jahr` keinen besseren Matchkandidaten als `Buch.Jahr` gibt.
Das Ziel im gerichteten Fall ist es, alle Matchkandidaten in Hinblick auf ein Schema zu finden. Entscheidet man sich für Schema *S1* (SmallLarge), so wird versucht, für alle *S1*-Elemente Matchkandidaten zu finden, wobei man in Kauf nimmt, dass *S2*-Elemente keinen Matchpartner haben. Umgedreht kann man sich auch für das Schema *S2* entscheiden (LargeSmall), wobei hier dann möglicherweise *S1*-Elemente ungematcht bleiben.
Dieser Ansatz ist motiviert durch die Tatsache, dass viele Anwendungen solche gerichteten Matches benötigen. Im Beispiel der Integration einer neuen Datenquelle mit Schema *S1* in ein Data Warehouse mit dem globalen Schema *S2* würde man sinnvoller Weise SmallLarge als Option nehmen.

Der letzte Teilschritt besteht aus dem Berechnen eines kombinierten Ähnlichkeitswertes und ist optional. Hierbei wird die Ähnlichkeitsmatrix zu einem einzigen Wert aggregiert, die als *Schemaähnlichkeit* bezeichnet wird. Als Aggregationsstrategien sind Dice und Mittelwert implementiert worden. Dice basiert auf dem Dice-Koeffizienten aus [CAFP98] und liefert das Verhältnis der Elementanzahl, welche gemacht werden können, zur Gesamtanzahl der Mengenelemente. Im Gegensatz zum Mittelwert wird Dice nicht durch die einzelnen Ähnlichkeitswerte beeinflusst. Da er stets den höheren Ähnlichkeitswert aufweist, ist Dice die optimistischere Aggregationsstrategie. Nur wenn alle Elementähnlichkeiten auf 1.0 gesetzt werden, ergeben beide Strategien den gleichen Wert.

[7] δ ist relativ oder absolut spezifizierbar

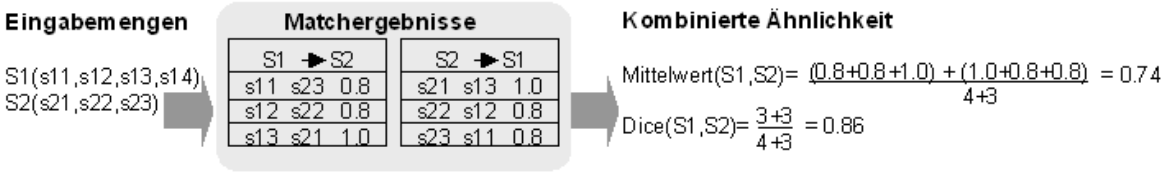


Abbildung 3.5: Beispiel zur Berechnung des kombinierten Ähnlichkeitswertes [DR02]

Ebenso wie die Kombination der Ähnlichkeitswerte in den gerade beschriebenen Teilschritten wurde der Vorgang für die Hybridmatcher implementiert, um die Ergebnisse der Teilmatcher (Constituent matcher, [DR02]) zu einem Wert zu kombinieren. Auch für diesen Fall werden Aggregations- und Selektionsoperationen auf einem Ähnlichkeitswürfel ausgeführt, der kombinierte Ähnlichkeitswert ergibt hier jedoch die *Elementähnlichkeit*.

3.5 Matcherbibliothek

In Tabelle 3.4 findet sich eine Übersicht über alle Matcher, die in COMA++ implementiert wurden. Klassifiziert wurden diese nach der zur Ausführung benutzten Information.

Im Folgenden werden die Einzel- und Hybridmatcher kurz vorgestellt. Der Wiederverwendungsorientierte Schema-Matcher wird in Kapitel 4 näher erläutert.

3.5.1 Einfache Matcher

In COMA++ gibt es zwei einfache Stringmatcher, die auf den Elementnamen arbeiten und keine zusätzliche Information verwenden.

n-gram Zeichenketten werden von diesem Matcher anhand ihrer n-Gramme, das sind Sequenzen von n-Zeichen, verglichen. Das führt zu Varianten wie Digram (n=2) und Trigram (n=3).

EditDistance Die Ähnlichkeit der Strings wird von diesem Matcher durch die Anzahl der Änderungsoperationen berechnet, die nötig ist, um eine Zeichenkette in die andere zu transformieren. Dies ist die Levenshtein-Metrik nach [HD80].

Desweiteren gibt es einen Matcher, der Statistiken über das Schema und seine Elemente verwendet:

Statistics Dieser Matcher berechnet die Ähnlichkeit von Elementen anhand ihrer Statistiken. Das sind beispielsweise die Anzahl der Kindelemente oder Geschwister.

Matchertyp	Matcher	Schemainformation	Zusatzinformation
Einfach	n-gram	Elementnamen	-
	EditDistance	Elementnamen	-
	Synonym	Elementnamen	Wörterbücher
	DataType	Datentypen	Kompatibilitätstabelle der Datentypen
	Taxonomy	Elementnamen	Taxonomie
	Statistics	Statistiken	-
	Comment	Kommentare	Wörterbücher, Taxonomie
Hybrid	Name	Elementnamen	Wörterbücher
	NameType	Elementnamen + Datentypen	Wörterbücher
	NameStat	Elementnamen + Statistiken	Wörterbücher
	NamePath	Pfade	
	Children	Kinderelemente	
	Leaves	Blätterelement	
	Siblings	Geschwisterelemente	
	Parents	Elternelemente	
Wiederverwendungs-orientiert	Reuse	-	existierende Matchergebnisse

Tabelle 3.4: In der Matcherbibliothek implementierte Matcher ([DR02])

Einfache Matcher, die jedoch Zusatzinformation verwenden, sind:

Synonym Dieser Matcher schätzt die Ähnlichkeit ab, indem er ein spezifisches Wörterbuch nach den Elementnamen durchsucht und so ihre terminologische Beziehung ermittelt. Implementiert wurden einfache Ähnlichkeitswerte, welche von der Art der Beziehung abhängen, wie beispielsweise 1.0 für Synonyme und 0.8 für Hyperonyme.

DataType Hier verwendet der Matcher eine Synonymtabelle, welche den Grad der Kompatibilität zwischen vordefinierten Datentypen spezifiziert. Zur Berechnung der Ähnlichkeitswerte werden die Datentypen der Schemaelemente den generischen Datentypen zugeordnet und die dazugehörigen Kompatibilitätswerte zurückgeliefert.

Taxonomy Eine Taxonomie wird von diesem Matcher wie eine zwischengeschaltete Ontologie verwendet. Sie dient zum Nachschlagen der Verwandtschaft der Elemente. In Abhängigkeit von der Distanz der Elemente innerhalb der Taxonomie wird ein Ähnlichkeitswert berechnet.

Comment Für jedes Element können Kommentare vorhanden sein, die das Element und seine Bedeutung beschreiben. Dieser Matcher matcht Elemente anhand dieser Kommentare, wobei er sowohl Synonym- und Abkürzungsverzeichnisse als auch Taxonomien verwendet.

3.5.2 Hybride Matcher

Der Ansatz der Hybridmatcher wurde in Abschnitt 2.5 vorgestellt. Diese Matcher benutzen eine feste Kombination von einfachen und anderen hybriden Matchern, um präzisere Ähnlichkeitswerte zu erhalten.
COMA++ unterstützt insgesamt acht Hybridmatcher, davon arbeiten zwei auf der Elementebene:

Name Dieser Matcher betrachtet zwar nur die Elementnamen, ist jedoch ein Hybridmatcher, da er verschiedene einfache Stringmatcher kombiniert. Zuerst werden Vorverarbeitungsschritte ausgeführt, wie das Zerlegen der Zeichenkette in einzelne Teile (*Token*), z. B. `VerlegtBei`→{`Verlegt, Bei`}, oder das Ersetzen von Abkürzungen, z. B. `Nr`→`Nummer`. Danach werden auf den Teilzeichenketten einfache Matcher ausgeführt, wie Affix, Trigram und Synonym. Die von den einzelnen Matchern zurückgelieferten Werte werden kombiniert und ergeben damit den Ähnlichkeitswert der Elementnamen.

NameType Dieser Matcher kombiniert die Ähnlichkeitswerte der Datentypen und der Namen. Dazu nutzt er die Matcher DataType und Name.

Die anderen sechs Hybridmatcher befassen sich mit der Strukturebene:

NameStat Für die Berechnung der Ähnlichkeitswerte verwendet dieser Matcher sowohl die Elementnamen als auch die Statistiken, die zu jedem Element berechnet wurden.

NamePath In diesem Fall werden Elemente mit Hilfe ihrer hierarchischen Namen gematcht. Es werden sowohl strukturelle Aspekte als auch die Elementnamen betrachtet. Zuerst wird eine lange Zeichenkette aus dem Pfad gebildet, indem alle Elementnamen des Pfades aneinander gehängt werden. Um die Ähnlichkeit zweier Zeichenketten zu berechnen, wird *Name* ausgeführt. So können zum Beispiel Kontexte von Elementen betrachtet und unterschieden werden, wie bei `Adresse` aus Abbildung 3.2, welches zwei Kontexte besitzt: `Bestellung.Lieferung.Adresse` und `Bestellung.Rechnung.Adresse` .

Children Dieser strukturelle Matcher bestimmt die Ähnlichkeit zweier Elemente basierend auf den kombinierten Ähnlichkeiten ihrer Kindelemente. Diese Kindelemente können Blattelemente und innere Kinder sein. Die Ähnlichkeit der Blätter wird mit Hilfe eines Matchers der Blattebene berechnet, wobei TypeName die Standardeinstellung bildet. Die Ähnlichkeit der inneren Kinder wird dann rekursiv berechnet anhand der Ähnlichkeit ihrer Kinder.

Leaves Im Gegensatz zu Children betrachtet dieser Matcher nur die Ähnlichkeit der Blattelemente, wenn zwei Elemente gematcht werden. Ein Vorteil dieses Vorgehens ist die größere Stabilität gegenüber Strukturunterschieden. Auch hier wird ein Matcher der Blattebene benötigt, wobei NameType die Standardeinstellung ist.

Siblings Dieser Matcher berechnet die Ähnlichkeit von Elementen anhand der Ähnlichkeit ihrer Geschwisterelemente. Als Standardeinstellung wird der Leaves verwendet.

Parents Die Ähnlichkeit der Elternelemente werden von diesem Matcher betrachtet, um die Ähnlichkeit von Elementen zu bestimmen. Leaves ist dafür die Standardeinstellung.

3.6 Matchstrategien

Um mit großen Schemata zurechtzukommen, setzt COMA++ eine fragmentbasierte Matchstrategie Fragment um, welche in [RDM04] erläutert wird. Dabei wird das gegebene Matchproblem in kleine Teilprobleme getrennt, indem man auf der Ebene von Schemafragmenten matcht. So wird der Suchraum reduziert, was eine bessere Ausführungszeit zur Folge hat und vielleicht auch eine bessere Qualität, da weniger falsche Korrespondenzen erzeugt werden können. Außerdem ermöglicht die Teilung des Matchprozesses in einzelne Schritte eine bessere Kontrolle bei der Nutzerinteraktion. Die GUI-Darstellung von großen Schemata ist problematisch, kann aber durch Teilschritte organisiert und verbessert werden. Der fragmentbasierte Matchvorgang besteht aus vier Schritten, die in Abbildung 3.6 zu sehen sind:

1. *Schemateilung*:
   Ein Fragment ist ein Teilgraph, der von einem als Wurzel bestimmten Knoten, alle Knoten bis hin zur Blattebene beinhaltet. Das Matchen von kompletten Schemata ist ein Sonderfall des Fragmentmatching und war auch schon im vorherigen Prototyp möglich. Zur Bestimmung von Fragmenten in einem Schema sind viele Strategien möglich, jedoch werden zur Zeit nur drei einfach Fragmenttypen unterstützt: Subschemata, Shared und benutzerdefinierte Fragmente. Mit Subschemata sind Teile des Schemas gemeint, die eigenständig instanziiert werden können. Ein Beispiel für Subschemata sind die einzelnen Nachrichtenformate in einem XSD-Schema. Shared Fragmente sind mehrfach genutzte Strukturen. Sie beinhalten ein hohes Potential für Matchkandidaten aufgrund ähnlicher Nutzung. Fragmente, die durch einen Nutzer definiert werden, sind für ein interaktives Matching geeignet.

2. *Identifizierung ähnlicher Fragmente*:
   Nachdem man die Fragmente für jedes Schema gefunden hat, werden diese verglichen und ihre Ähnlichkeit berechnet. Dazu werden Fragmente zu einer Fragmentwurzel abstrahiert und diese dann miteinander gematcht. Mit den erlangten Werten dieses Matchprozesses können ähnliche Fragmente identifiziert werden.

3. *Matchen der Fragmente*:
   Die Elemente der im vorherigen Schritt gefundenen ähnlichen Fragmente werden nun gematcht. Wenn nicht zu viele Fragmente vorhanden sind, dann ist dieser

Matchprozess stark verkürzt. Das Ergebnis der Matchoperationen ist eine Menge von Mappings mit Korrespondenzen zwischen den Fragmentkomponenten.

4. *Kombination der Ergebnisse*:
   Die Matchergebnisse auf der Fragmentebene werden jetzt zu einem globalen Matchergebnis zusammengefügt. Die Lage der Fragmente, wie z. B. ihr Kontext, muss auch berücksichtigt werden, damit global gültige Pfade für die Fragmentkomponenten entstehen.

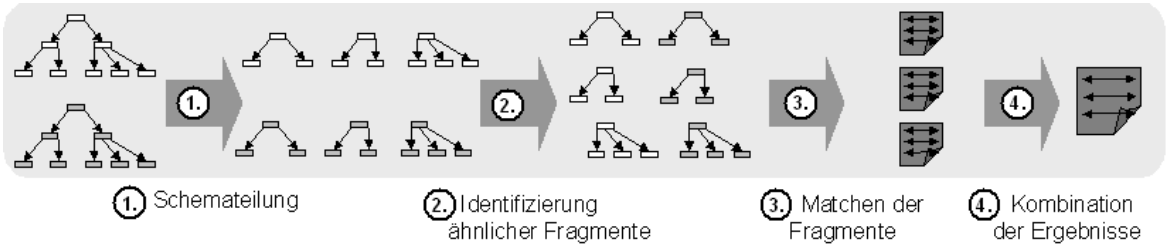


Abbildung 3.6: Die vier Schritte des fragmentbasierten Matchens

Diese vier Schritte können in der graphischen Umgebung nacheinander ausgeführt werden, wobei es dem Benutzer möglich ist, nach jedem Schritt Änderungen vorzunehmen. Dabei ist es unter anderem möglich die Strategie zur Fragmentauswahl zu spezifizieren, die Matchstrategie festzulegen und Matchergebnisse zu kombinieren.
Es gibt auch eine automatische Ausführung der Strategie Fragment, bei welcher alle Schritte nacheinander und ohne Unterbrechung abgearbeitet werden. Das Ergebnis ist auch hier das kombinierte Matchergebnis.

3.6.1 AllContext

Die Matchstrategie AllContext ist eine Strategie für kontextabhängiges Matchen. Abbildung 3.7 stellt den Ablauf dar. Zuerst werden alle Kontexte identifiziert und dann für jeden Teilmatcher des gewählten Matchers miteinander gematcht. Da jeder Pfad gegen jeden gematcht wird, ergibt sich eine quadratische Komplexität der Berechnung in Abhängigkeit der Pfadanzahl. Für große Schemata mit vielen mehrfach verwendeten Komponenten kann somit die Komplexität problematisch werden, was längere Ausführungszeiten mit sich bringt. Für die Berechnung der Pfadähnlichkeiten wird eine Operation genutzt, die in einem Schritt aus den Knotenähnlichkeiten die Pfadähnlichkeiten berechnet. Dies stellt eine deutliche Optimierung gegenüber der Berechnung der Ähnlichkeit für jede einzelne Pfadkombination dar, bei der die einzelnen Ähnlichkeiten immer wieder berechnet werden müssen.
Für diese Strategie wird genau ein Matcher benötigt, der sowohl die Knoten als auch Kontexte matcht. Die Standardeinstellung resultiert aus Evaluationen von [DR02] und [ADMR05] und besteht aus einem Hybridmatcher, der vier Teilmatcher verwendet: Name, Leaves, Parent und NamePath. Zur Kombination der Matchergebnisse wird als

Strategie für die Aggregation Mittelwert, für die Richtung Beide und als Selektion Threshold=0,5 + MaxDelta=0,02 verwendet.

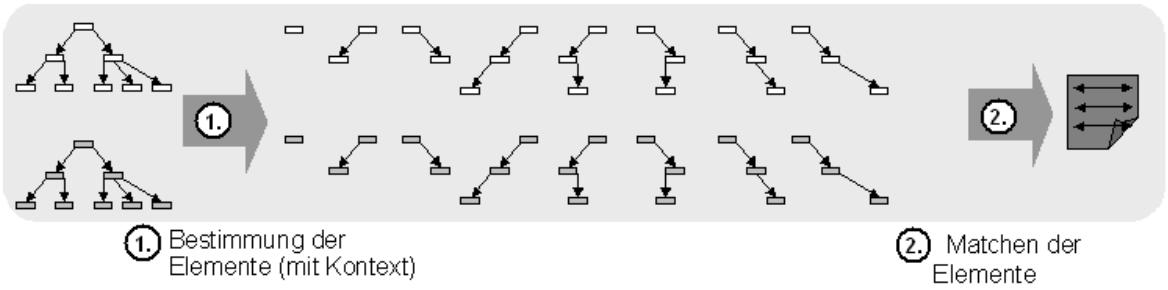


Abbildung 3.7: Das Vorgehen bei der Matchstrategie AllContext

3.6.2 FilteredContext

Auch die Matchstrategie FilteredContext ist eine Strategie fürs kontextabhängige Matchen, stellt jedoch eine Optimierung zu AllContext dar. FilteredContext besteht aus zwei Phasen. Zuerst werden die ähnlichsten Knoten durch ein Matchen aller Knoten miteinander identifiziert und dann deren Kontexte miteinander gematcht. Der Vorteil dabei ist, dass eine aufwendige Berechnung der Pfadähnlichkeit nur noch ausgeführt wird, wenn die Knoten auch ähnlich sind. Durch diese Filterung werden Berechnungen eingespart gegenüber einem Vergleich aller Pfade für die Knoten, die einander nicht ähnlich sind. Auch für diese Strategie wird die in Abschnitt 3.6.1 erwähnte Optimierung verwendet wird, welche die Pfadähnlichkeiten aus den Knotenähnlichkeiten berechnet. Insgesamt ergibt sich für FilteredContext auch eine quadratische Komplexität - allerdings in Abhängigkeit der Knotenanzahl, die im allgemeinen aufgrund von mehrfach verwendeten Komponenten deutlich geringer ist als die Pfadanzahl. Dadurch ist diese Strategie auch für große Schemata schnell.

Das Vorgehen ist in Abbildung 3.8 illustriert.

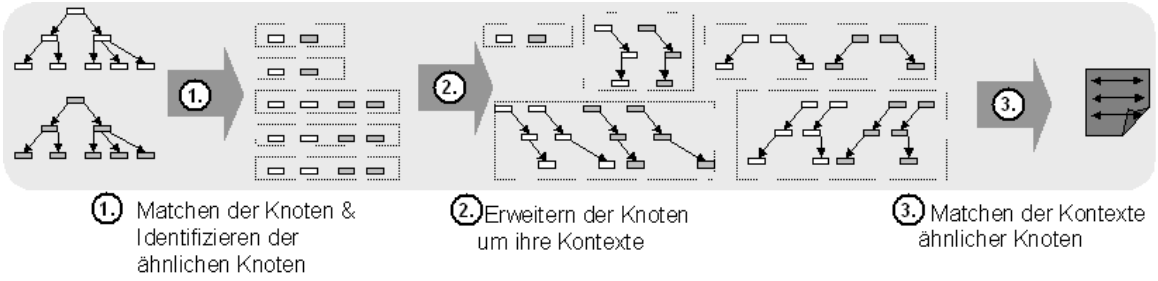


Abbildung 3.8: Das Vorgehen bei der Matchstrategie FilteredContext

Von dem vorgestellten, fragmentbasierten Matchen fällt der Schritt 1 weg, da ja alle Knoten bekannt sind und jeder ein Fragment darstellt, der Schritt 2 wird ausgeführt,

die Schritte 3 und 4 fallen zusammen in das Pfadmatchen.
Für diese Strategie werden zwei Matcher benötigt, einer - den *Knotenmatcher* - für die Identifizierung der ähnlichen Knoten und ein weiterer für das Matchen der Pfade, den *Kontextmatcher*. Die Standardeinstellungen resultieren - wie schon bei AllContext - aus Evaluationen von [DR02] und [ADMR05]. Für den Knotenmatcher wird eine Matcherkombination aus Name, Leaves und Parents verwendet und für den Kontextmatcher ein Matcher, der auf NamePath basiert. Beide Matcher verwenden zur Kombination die gleichen Strategien: *Mittelwert* für die Aggregation, *Beide* für die Richtung und als Selektion *Threshold*=0,5 + *MaxDelta*=0,01.

4 Wiederverwendung

Wiederverwendungsorientierte Matcher sind motiviert durch die Erwartung, dass viele Schemata, die gematcht werden sollen, ähnlich den früher gematchten Schemata oder sogar identisch sind. In Projekten und Unternehmen findet eine Weiterentwicklung statt, die zu Veränderungen von Daten und damit Schemata führen kann. So können im Laufe der Zeit neue Attribute hinzukommen, oder vorhandene Attribute werden nicht mehr gebraucht. Für ein Schema *S1*, das zur Datenspeicherung in Datenbanken dient, gibt es also unterschiedliche Versionen *S1', S1"....* Nun sollen benötigte Mappings zu anderen Schemata, die sich nicht verändert haben, möglichst nicht erneut erstellt werden. Für den Teil der Schemaversionen *S1',S",etc.*, der identisch zu *S1* ist, können die vorhandenen Korrespondenzen übernommen werden. Es müssen desweiteren Anpassungen zu den Schemaänderungen erfolgen, indem zusätzliche Korrespondenzen hinzugefügt werden.
Eine weitere Motivation für die Wiederverwendung ergibt sich für ein Matchproblem *S1* und *S2* durch das Vorhandensein von Mappings, die zwar nicht zwischen diesen beiden Schemata sind sondern von jeweils einem der beiden zu anderen Schemata führen. Gibt es nun beispielsweise die Mappings $S1 \leftrightarrow Si$ und $Si \leftrightarrow S2$, so lassen diese sich aneinander fügen, um ein neues Mapping zu erzielen.

4.1 Schema-Wiederverwendung

Ein allgemeinerer Ansatz als die Wiederverwendung von global definierten Namen ist die *Wiederverwendung ganzer Schemafragmente* mit den Eigenschaften wie Datentypen, Schlüsseln und Bedingungen. Dies ist besonders lohnenswert für oft verwendete Komponenten wie z. B. Adresse, Kunde und Bestellung. Diese Komponenten sollten am besten in einer Schemabibliothek definiert und gepflegt werden. Es ist zwar unwahrscheinlich, dass sich alle Anwender auf ein solches Schema einigen, aber es kann für ein Unternehmen und deren Handelspartner festgelegt werden. Überdies macht es Sinn, dass ähnliche Unternehmen auch dieses Schema benutzen, um die Anzahl ähnlicher Schemata zu reduzieren. Verfasser solcher Schemata sollten Zugang zu bereits bestehenden Schemabibliotheken besitzen, um vorhandene Schemafragmente und Begriffe wiederzuverwenden. Die Originalherkunft solcher Fragmente und Begriffe sollte durch z. B. einen XML Namensraum (engl. Namespaces) kenntlich gemacht werden, um bei der Matchoperation diese zu identifizieren und einfacher zuordnen zu können. Die Verwendung solcher standardisierten Schemata würde zwar noch immer nicht die Transformationen und damit das Matchen an sich überflüssig machen, da es weiterhin unterschiedliche Schemata gibt, die zudem weiterentwickelt werden. Es würden jedoch

die Matchprobleme einfacher zu lösen sein und somit die Mappings eine höhere Qualität erreichen, wenn Schemata Schemabibliotheken verwenden.
Eine weitere Möglichkeit zur Wiederverwendung von existierenden Schemata wird von [MBC+03] behandelt. Dabei wird das Lernen und die Verwendung von Komponentstatistiken in einer Sammlung von Schemata fokussiert. Das extrahierte Wissen wird dann angewendet, um neue Matchaufgaben zu lösen.

4.2 Mapping-Wiederverwendung

4.2.1 Thesaurus

Bei der Wiederverwendung von Mapping sind verschiedene Varianten möglich. Zum einen können früher erzeugte Matches auf der Elementebene zu einem Thesaurus hinzugefügt werden. Dazu wird zu jedem Match einfach ein Eintrag in den Thesaurus hinzufügt bzw. bereits bestehende erweitert.Zum anderen können ganze Strukturen wiederverwendet werden. Dies ist von Vorteil, wenn man unterschiedliche, aber ähnliche Schemata zu einem gemeinsamen Zielschema matchen möchte. Dieser Fall tritt beispielsweise auf, wenn neue Quellen in ein Data Warehouse integriert werden sollen. Die Wiederverwendung früherer Matchergebnisse wird eventuell nur für Teile eines neuen Schemas möglich sein. Ein Hauptproblem ist dabei zu bestimmen, welche Teile des neuen Schemas ähnlich zu Teilen bisheriger Matchergebnisse sind. Dies ist wiederum auch ein Matchproblem. Darüber hinaus sollte die Wiederverwendung auf ähnliche Anwendungsgebiete eingeschränkt werden, da ansonsten zuvor bestimmte Ähnlichkeitswerte aufgrund des Vorhandenseins von Homonymen nicht mehr gültig wären.

4.2.2 Maschinelles Lernen

Eine Variante für die Wiederverwendung von Mappings ist das *maschinelle Lernen* (Machine learning). Maschinelles Lernen ist ein Oberbegriff für die "künstliche" Generierung von Wissen aus Erfahrung: Ein künstliches System lernt aus Beispielen und kann nach Beendigung der Lernphase verallgemeinern. In diesem Fall werden korrekte Mappings als Trainingsdaten für ein System genutzt, welches aus ihnen lernt. Beim Matchen werden dann Korrespondenzen gefunden, die denen in den Trainingsdaten ähnlich sind. Probleme bei diesem Ansatz sind die Erstellung der Trainingsdaten, der Trainingsaufwand selber und differente Schemata. Außerdem kann es sein, dass die Trainingsdaten eventuell nicht 100% korrekt sind und somit auch dadurch zu falschen Ergebnissen führen.
Auf den Ansatz des *Maschinellen Lernens* wird in dieser Diplomarbeit nicht näher eingegangen. In den Prototypen LSD [DDH01, DDL00] und GLUE [DMDH02] sind diese Ansätze umgesetzt. weitere Informationen zu finden.

4.2.3 Transitivität

Eine andere Variante ist die Ausnutzung der Transitivität von Ähnlichkeit, durch die aus vorhandenen Mappings neue erstellt werden. Die MatchCompose-Operation nutzt diese Transitivität, um aus vorhandenen Mappings neue zu berechnen. MatchCompose hat als Eingabe zwei Matchergebnisse $M_1 : S_1 \leftrightarrow S_i$ und $M_2 : S_i \leftrightarrow S_2$, die das Schema S_i gemein haben. Als Ausgabe produziert MatchCompose ein neues Matchergebnis $M : S_1 \leftrightarrow S_2$ zwischen den Schemata S_1und S_2, indem aus den bestehenden Korrespondenzen neue für das Matchproblem $S_1 - S_2$ abgeleitet werden. In Abbildung 4.1 ist ein Beispiel für die Anwendung von MatchCompose auf ein Mappingpaar $M1$ und $M2$ zu sehen, wobei Si das Zwischenschema ist.

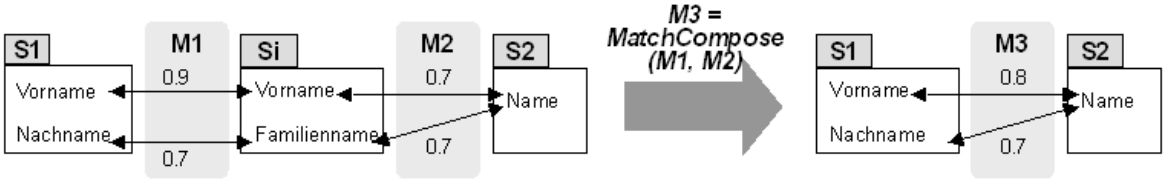


Abbildung 4.1: Ein Beispiel zur Erzeugung des Mappings *M3* durch die Wiederverwendung der Mappings *M1* und *M2*

Die Operation MatchCompose setzt voraus, dass es eine *Transitivität* bei der Ähnlichkeit von Elementen gibt, wenn also `a` ähnlich `b` ist und `b` ähnlich `c`, dann ist auch `a` ähnlich `c`. Außerdem wird die Ähnlichkeit zwischen Elementen als *symmetrisch* angenommen, das heißt, wenn `a` dem Element `b` mit Wert k ähnlich ist, so ist auch `b` dem Element `a` mit Wert k ähnlich. Dies ist auch der Grund dafür, dass Mappings *transponierbar* sind, das heißt, aus einem Mapping $M : S1 \leftrightarrow S2$ kann ein Mapping $M' : S2 \leftrightarrow S1$ erzeugt werden, indem man Ausgangs- und Zielschema miteinander vertauscht und die Korrespondenzen umdreht.
Zur Berechnung des neuen Ähnlichkeitswertes aus den vorhandenen gibt es verschiedene Möglichkeiten. Ein weit verbreiteter Ansatz ist die Multiplikation der Einzelwerte. Dies kann jedoch zu einer starken Verkleinerung der Ähnlichkeitswerte führen. Für das Beispiel $Vorname \overset{0.9}{\leftrightarrow} Vorname \overset{0.7}{\leftrightarrow} Name$ ergibt sich durch Multiplikation $0.9 * 0.7 = 0.63$, wobei 0.63 nicht den Wert wiederspiegelt, den man erwarten würde als sich ergebene Ähnlichkeit zwischen $KontaktVorname$ und $Vorname$. So wurden als Alternativen zur Multiplikation der *Mittelwert (Average)*, das *Maximum* und das *Minimum* implementiert. Für das erwähnte Beispiel ergibt *Mittelwert* 0.8, *Maximum* 0.9 und *Minimum* 0.7.
Die Operation MatchCompose kann iterativ angewendet werden und und so aus mehreren Mappings *M1-M2-...-Mi*, die jeweils über ein Zwischenschema verbunden sind, ein Endmapping berechnen. Dazu erzeugt es zuerst ein Mapping M' aus M1 und M2, dann aus M' und M3 ein weiteres Mapping M'' usw. Als letzter Schritt wird aus $M^{(i-2)}$ und Mi das Endergebnis $M^{(i-1)}$ generiert.

MatchCompose generiert jedoch nicht immer das gewünschte Ergebnis. Fehlen beispielsweise die Join-Kandidaten in dem Zwischenschema, dann können korrekte Korrespondenzen nicht erzeugt werden, da die Teilmappings diese Korrespondenzen nicht beinhalten. Es können auch Korrespondenzen erzeugt werden, die nicht korrekt sind, durch z.B. n:1 und 1:n Matches. Ein Beispiel ist in Abbildung 4.2 dargestellt. Dabei beinhalten die Ausgangsmappings jeweils zwei Korrespondenzen und durch MatchCompose wird ein Endmapping generiert, welches vier Korrespondenzen beinhaltet, von denen jedoch nur zwei korrekt sind ($Vorname \cong Vorname$ und $Nachname \cong Familienname$).

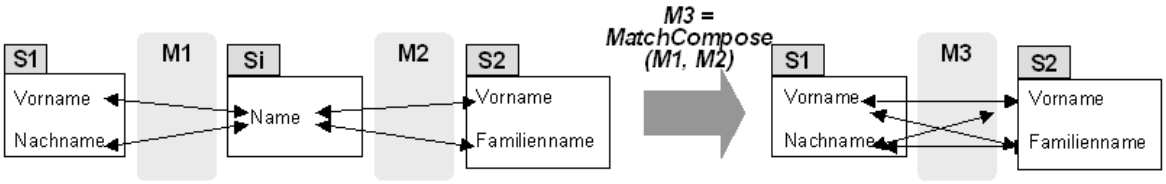


Abbildung 4.2: Ein Beispiel zur Erzeugung von korrekten und falschen Korrespondenzen durch die MatchCompose-Operation

Der in COMA implementierte Schema-Matcher [DR02] benutzt diesen Ansatz. Sein Funktionsweise ist in Abbildung 4.3 dargestellt. Zuerst bestimmt der Nutzer das Matchproblem, indem er zwei Schemata *S1* und *S2* angibt. Der Matcher *Schema* durchsucht sämtliche Matchergebnisse des Repository und selektiert jene heraus, die als eines der beiden Schemata *S1* oder *S2* haben. Danach bildet er alle möglichen Kombinationen von Mappingpaaren, die so verbunden sind, dass man von *S1* zu *S2* über ein Zwischenschema gelangt. In der Abbildung 4.3 sind dies *Si, Sj* und *Sk*, wobei diese verschieden von *S1* und *S2* sind. Für jedes dieser Paare wird die Funktion MatchCompose angewendet, welches ein Ergebnismapping berechnet. Diese einzelnen Mappings werden wiederum durch Aggregation, Richtung und Selektion zu einem Matchergebnis kombiniert. Diese Kombination mildert die erwähnten Schwächen der MatchCompose-Operation, da Korrespondenzen gefiltert werden können und nicht unbedingt in jedem Teilmapping vorhanden sein muss.

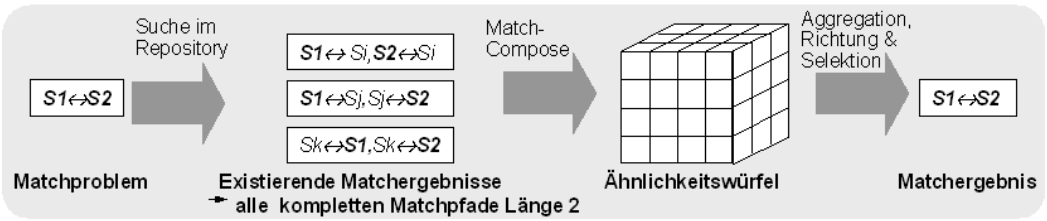


Abbildung 4.3: Wiederverwendung im Schema-Matcher [DR02]

4.3 Pivotschema

Die Verwendung eines globalen Schemas bzw. eines Masterschemas ist eine Thematik, die schon auf anderen Gebieten wie der Datenintegration oder bei Mediatoren zur Vereinfachung der Integration seit geraumer Zeit eine wichtige Rolle spielen. In der Datenintegration gibt es das *mediated Schema*, auf welches heterogene Informationsquellen (Datenbanken, Dateien, Web-Quellen) abgebildet werden, um einen einheitlichen, integrierten Zugriff auf sie zu ermöglichen. Abhängend von der Gestaltung der Korrespondenzen zwischen den lokalen Schemata der Quellen und dem globalen Schema, wird zwischen einem Global-as-View und einem Local-as-View Ansatz unterschieden. [Hal01, BLN86a]. Darüber hinaus steht die Thematik in der Nutzung von Ontologien oder Taxonomien [BMW01] im Vordergrund.

Durch das Matchen zu einem Standardschemas, dem Pivotschema, werden bei der Mapping-Wiederverwendung die Mappingpfade kurz gehalten. Sie beinhalten immer zwei Mappings. Alle Schemata, die Teil eines Matchproblems sind, werden also - statt direkt zu dem anderen Schema - zu einem vorgegebenen Pivotschema gematcht. Die Mappings von den Schemata zu dem Pivotschema werden dann unter der Annahme der Transitivität genutzt, um das gewünschte Mapping zu erzeugen. Dies entspricht einem Mappingpaar mit einem gemeinsamen Zwischenschema, auf welches die MatchCompose-Operation angewendet wird.

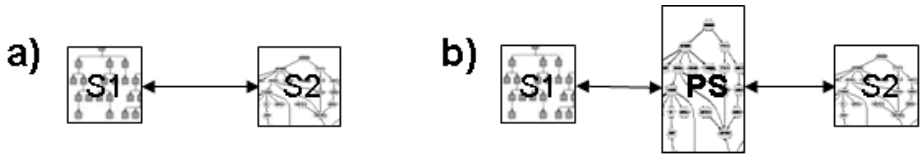


Abbildung 4.4: Zwei Schemata *S1* und *S2* werden a) direkt gematcht b) unter Verwendung eines Pivotschemas (PS)

Im ersten Moment scheint es schwer nachvollziehbar zu sein, warum man ein Matchproblem durch zwei neue ersetzen sollte, wie es in Abbildung 4.4 illustriert ist. Sobald jedoch nicht nur eine sondern mehrere Matchaufgaben gelöst werden sollen, beinhaltet die Verwendung eines Pivotschemas Einsparungen an Arbeit. Abbildung 4.5 illustriert, wie n Schemata miteinander gematcht werden. Matcht man diese direkt, so ergeben sich $\frac{n^2+n}{2}$ Kombinationen für $n > 2$, die dann zu $\frac{n^2+n}{2}$ Matchaufgaben führen. Für das Beispiel von fünf Schemata müssen also zehn Matchaufgaben gelöst werden. Schaltet man ein Pivotschema dazwischen, muss für jedes der Schemata ein Mapping zum Pivotschema erzeugt werden, was n Matchaufgaben - für das Beispiel also fünf - bedeutet. Die gesuchten $\frac{n^2+n}{2}$ Mappings lassen sich hier unter Anwendung von MatchCompose aus den vorhanden Mappings erzeugen. Insgesamt müssen dabei $\frac{n^2-n}{2}$ weniger Mappings erzeugt werden. Im Beispiel werden also fünf Matchaufgaben weniger gelöst, was eine Einsparung von 50% darstellt.
Kommt nun ein sechstes Schema hinzu, ergeben sich im direkten Fall fünf weitere

Matchaufgaben. Im anderen Fall muss nur ein Mapping vom neuen Schema zum Pivotschema erstellt werden. Die fünf Mappings zu den anderen Schemata lassen sich wiederum durch MatchCompose erzeugen.
In der Praxis wird sicher nicht für jede Kombination der Schemata ein Mapping benötigt werden. Da aber die Schemaanzahl meist nicht gerade klein ist, lohnt sich der Einsatz eines Pivotschemas. Allerdings wurde dieser Ansatz bisher noch nicht in einem Prototypen realisiert.

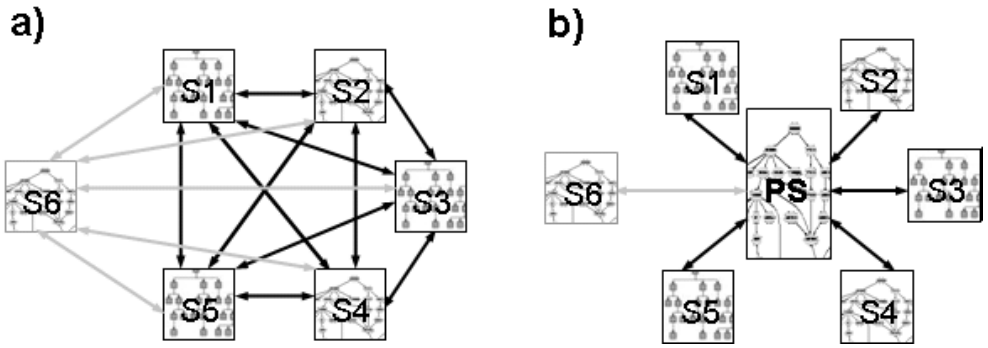


Abbildung 4.5: Fünf Schemata *S1* bis *S5* wurden a) direkt gematcht b) unter Verwendung eines Pivotschemas (PS); kommt *S6* hinzu, müssen neue Mappings erzeugt werden (graue Pfeile)

Problematisch ist die Abdeckung der Schemata durch das Pivotschema, denn davon hängt die Qualität der zu erzeugenden Mappings ab. Wenn ein Pivotschema einige Schemaelemente nicht abdeckt, dann können für diese Elemente auch keine Korrespondenzen erzeugt werden. Deckt es andererseits wirklich alle Elemente jedes importierten Schemata ab, dann erschwert dies das Erzeugen eine korrekten Mappings von einem neuen Schema zu dem Pivotschema, da mehr falsche Korrespondenzen im automatischen Matchprozess erzeugt werden.

5 Wiederverwendung von Mappings

In dem Prototyp COMA wurde durch den implementierten Schema-Matcher Mappings wiederverwendet, um neue Mappings zu erzeugen. Dabei werden jedoch immer genau zwei Mappings betrachtet, die über ein Zwischenschema durch MatchCompose zur Erzeugung eines Mappings genutzt werden. Darüber hinaus konnten Mappings bisher nur verwendet werden, wenn sie entweder vom Ausgangsschema oder zum Zielschema führten. Sobald es sich um andere Schemata handelt, auch wenn diese vorherige Versionen von den gegebenen Schemata sind, ist eine Wiederverwendung nicht möglich.
Aus diesen Gründen werden in dieser Arbeit weitere Möglichkeiten betrachtet. Einerseits werden von MatchCompose auch mehr als zwei Mappings verwendet, da die Operation iterativ anwendbar ist. Andererseits wird durch das Ersetzen von der gegebenen Matchaufgabe durch eine neue, möglichst einfacheren, die Verwendung von Mappings ermöglicht. Zusätzlich wird die Verwendung eines Pivotschemas zur Einsparung von Berechnung untersucht.

5.1 Mappingpfade

In Matchssystem, in denen viele Schemata gematcht werden, entstehen viele Mappings. Diese bieten das Potential zur Wiederverwendung. Die Grundlage für die Mapping-Wiederverwendung sind die Mappingpfade. Eine Menge von Mappings $\{M_1, M_2,...\}$ wird als Mappingpfad bezeichnet, wenn für die Mappings M_i: $S_{i,1} \leftrightarrow S_{i,2}$ und M_{i+1} : $S_{i+1,1} \leftrightarrow S_{i+1,2}$ gilt, dass sie ein gemeinsames Schema, nämlich $S_{i,2} = S_{i+1,1}$, besitzen. Dies ist das in Kapitel 4 erwähnte Zwischenschema. Weitere Darstellungsmöglichkeiten des Mappingpfades $M_1 - M_2$ sind $\{M_1 : S_i \leftrightarrow S_j, M_2 : S_j \leftrightarrow S_k\}$, $\{S_i \leftrightarrow S_j, S_j \leftrightarrow S_k\}$ und $S_i \leftrightarrow S_j \leftrightarrow S_k$. Die Länge eines Mappingpfade wird durch die Anzahl seiner Mappings bestimmt, das heißt ein Mappingpfad der Länge i besteht aus i Mappings. Ist jedes dieser Mappings nur einmal im Mappingpfad vorhanden, so ist dieser *zyklenfrei*. Im Folgenden wird jeder Mappingpfad als zyklenfrei angenommen.
Ein Mappingpfad, bei dem alle Mappings bereits vorhanden sind, indem es für den Matchprozess z.B. in einer Datenbank zur Verfügung steht, ist ein *vollständiger Mappingpfad*. Der Schema-Matcher von COMA verwendet somit vollständige Mappingpfade. Ist mindestens eines der Mappings nicht vorhanden und muss somit noch erzeugt werden, so handelt es sich um einen *unvollständiger Mappingpfad*. Für diese Arbeit wird implizit angenommen, dass für jeden unvollständigen Mappingpfad mindestens ein Mapping vorhanden ist, da ansonsten keine Wiederverwendung von Mappings

stattfindet. Ein unvollständiger Mappingpfad, der aus i Mappings besteht, von denen j noch benötigt werden, besitzt die Länge *i_j*.

Die Berechnung von Mappingpfaden entspricht der Wegbestimmung in einem Graphen [Die00]. Dabei sind die Schemata die Knoten des Graphen und die Mappings die Kanten. Ist ein Mapping vorhanden, so ist auch die Kante vorhanden. Es kann überdies zwischen zwei Schemata verschiedene Kanten geben, wenn es mehrere Mappings gibt. Mehrere Kanten zusammen ergeben einen Mappingpfad, der von einem Ausgangschema über ein oder mehrere Zwischenschemata zum Zielschema führt. Wird ein Pfad über die Kanten bestimmt, so ergibt sich damit ein vollständiger Mappingpfad. In dieser Arbeit werden nur vollständige Mappingpfade der Länge 2 bis 4 betrachtet. Längere Mappingpfade werden nicht untersucht, da anzunehmen ist, dass mit längeren Pfaden die Information - in diesem Fall die berechneten Ähnlichkeiten von Elementen - unschärfer wird.
Die Berechnung der unvollständigen Mappingpfade unterscheidet sich insofern von der Bestimmung der vollständigen, als dass auch Kanten in einem Pfad fehlen dürfen. Wie schon bei den vollständigen Mappingpfade werden nur kurze unvollständige Pfade und zwar der Länge 2_1, 3_1 und 3_2 betrachtet, da mit längeren Pfaden die Informationsunschärfe zunimmt. Abbildung 5.1 stellt einen vollständigen und einen unvollständigen Mappingpfade mit je zwei Mappings dar.

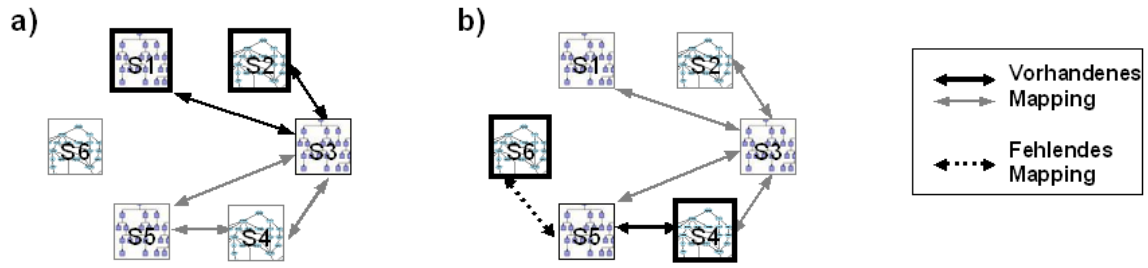


Abbildung 5.1: Ein Graph mit Schemata als Knoten und Mappings als Kanten mit a) einem vollständigen Mappingpfad zwischen *S1* und *S2*, b) ein unvollständiger Mappingpfad zwischen *S4* und *S6*

In Abbildung 5.2 ist der Algorithmus zu sehen, mit dem die untersuchten vollständigen und unvollständigen Mappingpfade berechnet werden. Angenommen das Matchproblem lautet $S1 - S2$. Zuerst werden alle Mappings herausgesucht, die entweder das Schema S1 oder das Schema S2 beinhalten, jedoch nicht beide. Für jedes Mapping $S1 \leftrightarrow Si$ ergibt sich ein vollständiger Mappingpfad der Länge 2, wenn $Si \leftrightarrow S2$ vorhanden ist und ansonsten ein unvollständiger Mappingpfad der Länge 2_1, wobei $Si \leftrightarrow S2$ noch zu erzeugen ist. Für jedes Mapping $Si \leftrightarrow S2$, für dass kein $S1 \leftrightarrow Si$ existiert, ergibt sich auch ein unvollständiger Mappingpfad der Länge 2_1 mit der Matchaufgabe $S1 - Si$. Darauffolgend werden alle Mappings $Si - Sj$ betrachtet, die keines der Schemata des gegebenen Matchproblems beinhalten. Existieren sowohl $S1 \leftrightarrow Si$

als auch $Sj \leftrightarrow S2$, dann ergibt sich daraus ein vollständiger Mappingpfad der Länge 3 $S1 \leftrightarrow Si \leftrightarrow Sj \leftrightarrow S2$. Ist nur eines der beiden Mapping vorhanden, dann ergibt sich ein unvollständiger Mappingpfad mit einem Matchproblem. Falls keines vorhanden ist, so handelt es sich um einen unvollständigen Mappingpfad mit den Matchaufgaben $S1 - Si$ und $Sj - S2$. Liegen sowohl $S1 \leftrightarrow Si$, $Sj \leftrightarrow Sk$ und $Sk \leftrightarrow S2$ oder $S1 \leftrightarrow Sk$, $Sk \leftrightarrow Si$ und $Sj \leftrightarrow S2$ vor, dann lässt sich daraus ein vollständiger Mappingpfad der Länge 4 bilden.

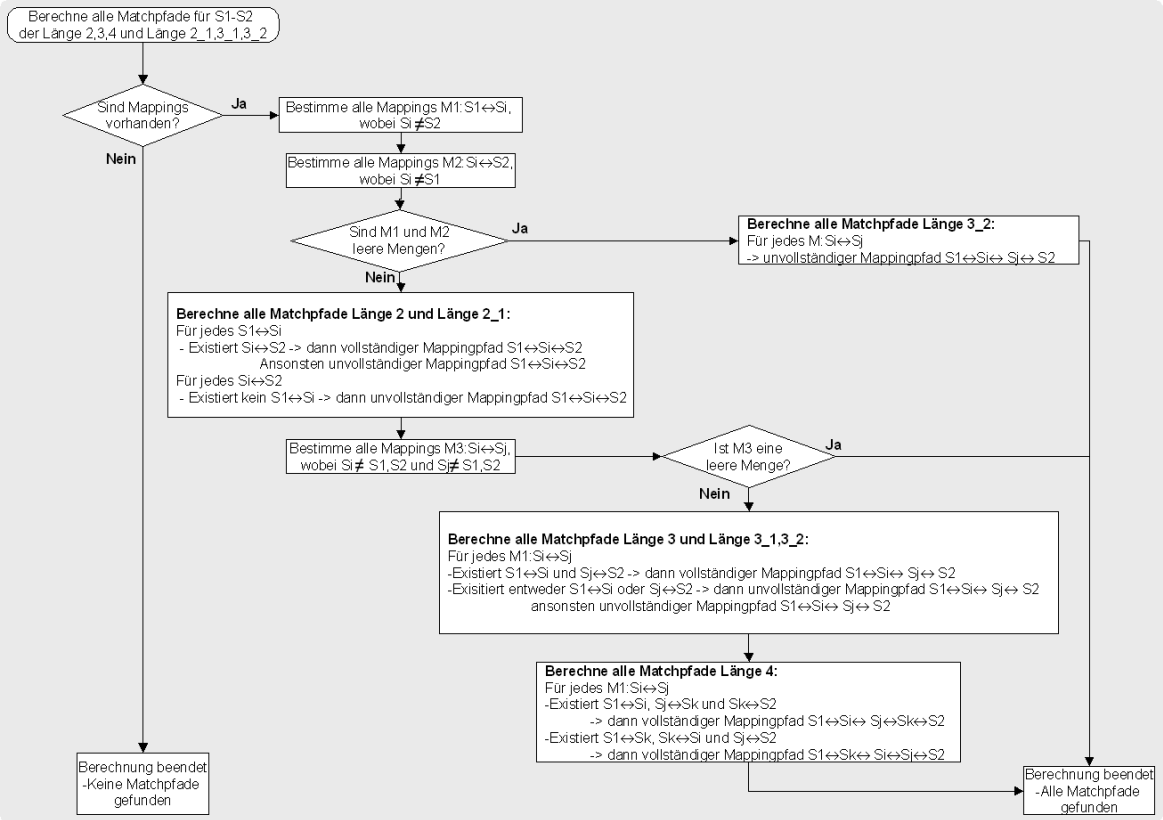


Abbildung 5.2: Algorithmus zur Bestimmung der vollständigen Mappingpfade Länge 2 bis 4 und Länge 2_1,3_1,3_2 für das Matchproblem *S1-S2*

5.2 Wiederverwendungsstrategien

Die Wiederverwendung von Mappings lässt sich durch zwei orthogonale Merkmale kennzeichnen. Zum einen kann sie vollständig, siehe Abschnitt 5.2.1, wenn bereits alle Mappings zur Wiederverwendung vorliegen. Sie ist unvollständig, siehe Abschnitt 5.2.2, wenn noch mindestens ein Mapping erzeugt werden muss. Zum anderen kann die Wiederverwendung kombiniert und unkombiniert ablaufen, abhängig davon ob die Mappings für mehrere Mappingpfade miteinander kombiniert werden oder nicht. Dies wird in Abschnitt 5.2.3 bzw. 5.2.4 vorgestellt.

5.2.1 Vollständige Mapping-Wiederverwendung

Bei der vollständigen Mapping-Wiederverwendung werden die vollständigen Mappingpfade genutzt und durch die Anwendung von MatchCompose aus den vorhanden Mappings ein neues generiert.
Ein vollständiger Mappingpfad der Länge 1 ist ein Sonderfall und entspricht zum Matchproblem $S1 - S2$ allen Mappings $S1 \leftrightarrow S2$ bzw. wegen der Transponierbarkeit von Mappings natürlich auch $S2 \leftrightarrow S1$, die bereits im Repository vorhanden sind. Dies ist eine direkte Wiederverwendung, bei der keine zusätzliche Operation mehr angewendet wird, sondern die Mappings direkt an den Nutzer weitergegeben werden können.
Wenn hier im Folgenden vollständige Mapping-Wiederverwendung erwähnt wird, dann ist dies auf Mappingpfade der Länge 2 und mehr bezogen, es sei denn, dass dies ausdrücklich erwähnt wird.

5.2.2 Unvollständige Mapping-Wiederverwendung

Die unvollständige Mapping-Wiederverwendung stützt sich auf die unvollständigen Mappingpfade und beinhaltet damit eine Kombination von möglicherweise bestätigten Mappings mit unbestätigten Mappings, die erst während der Abarbeitung des Mappingpfade berechnet werden. Für diesen Matchprozess können selbstverständlich sämtliche Matchstrategien und Matcher verwendet werden.
Ein Beispiel für einen unvollständigen Mappingpfad ist $\{S1 \leftrightarrow S2, S2 \leftrightarrow S2'\}$, wenn Schema *S1* zu einer älteren Version von *S2'*, nämlich *S2*, gematcht wurde und das Mapping $S1 \leftrightarrow S2$ im Repository zur Verfügung steht, aber $S2 \leftrightarrow S2'$ noch nicht vorhanden ist. Zuerst muss das Mapping $S2 \leftrightarrow S2'$ erstellt werden, um dann MatchCompose auf den Mappingpfad anwenden zu können. In diesem Fall wird das neue Matchproblem $S2 - S2'$ einfacher zu lösen sein als das gegebene $S1 - S2'$, da sich die alte Version *S2'* und die neue Version *S2* sehr ähnlich sind. Dies muss nicht automatisch für jeden unvollständigen Mappingpfad zutreffen.
Es ist sinnvoll, sich auf unvollständige Mappingpfade zu beschränken, in denen nur ein Mapping fehlt. Das gegebene Matchproblem wird damit durch ein neues ersetzt. Es sind jedoch auch Szenarien vorstellbar, in denen es sich lohnt, zwei neue Matchproblem anzugehen statt das alte direkt zu lösen. Dies ist der Fall, wenn beide neuen Matchprobleme einfacher zu lösen sind, als es für das gegebene möglich ist. Angenommen, das gegebene Matchproblem lautet $S1' - S2'$ und im Repository ist das Mapping $S1 \leftrightarrow S2$ vorhanden, wobei *S1* (*S2*) ältere Versionen von *S1'* (*S2'*) sind: Um das vorhandenen Mapping $S1 \leftrightarrow S2$ wiederzuverwenden, werden Mappings der neuen Schemaversionen zu den alten Versionen benötigt. Die sich ergebenden Matchprobleme lauten *S1'- S1* und *S2-S2'*, wobei sich die Schemata jeweils bis auf (meist geringe) Versionsunterschiede gleichen und somit die Matchaufgaben "leichtgewichtig" sind. Nach dem Erzeugen dieser Mappings kann MatchCompose auf den Mappingpfad $S1' \leftrightarrow S1 \leftrightarrow S2 \leftrightarrow S2'$ angewendet werden, um das gewünschte Mapping $M : S1 \leftrightarrow S2$ zu erhalten.

Selektion von Matchaufgaben

Um eine hohe Qualität bei der unvollständigen Mapping-Wiederverwendung zu erreichen, macht es Sinn, sich auf unvollständige Mappingpfade zu beschränken, bei denen die Matchaufgabe möglichst leicht zu lösen ist. Dabei soll "leicht zu lösen" bedeuten, dass mit den bisherigen im Prototypen implementierten Matchalgorithmen qualitativ hochwertige Mappings erzeugt werden können. Allgemein kann gesagt werden, dass Schemata, die einander ähnlich sind und somit wenig Heterogenität aufweisen, ein "leichtgewichtiges" Machproblem darstellen.
Die Schwierigkeit besteht darin festzustellen, welche neuen Matchaufgaben denn tatsächlich leicht zu lösen sind. Es muss identifiziert werden, wie ähnlich die Schemata einander sind, um somit die Schwierigkeit des Matchproblems abschätzen zu können. Die Berechnung der Schemaähnlichkeit kann anhand von Metadaten wie z. B. der Versionsnummern oder den Schemanamen geschehen oder natürlich anhand der Schemata selber.
Für diese Arbeit werden drei Möglichkeiten betrachtet:

- Ähnlichkeit der Schemanamen
- Ähnlichkeit der Namensräume der Schemata
- Ähnlichkeit der Elementnamen

Der Schemaname beinhaltet im besten Fall Informationen über das Schema, wie die Verwendung, Domäne oder Versionsnummer. Im ungünstigen Fall stellt der Name nur eine beliebige Zeichenkette dar. Der Ähnlichkeitswert ergibt sich durch eine Ähnlichkeitsberechnung auf den Zeichenketten, die für diese Arbeit mit dem n-gram Matcher erfolgt, der in Abschnitt 3.5.1 vorgestellt wurde.
Der Namensraum besitzt im allgemeinen wichtige Informationen, da er für die Bildung von abgegrenzten Gültigkeitsbereichen für Elemente und Attribute genutzt wird. Benutzen Schemata dieselben Namensräume, so bedeutet dies, dass sie die gleichen Elemente für die gleichen Sachen benutzen und das sie auch gleiche Strukturen verwenden. Wenn sich Namensräume stark ähneln und nicht gleichen, kann das z. B. daran liegen, dass sie unterschiedliche Metainformationen wie Datum oder Prozessdomäne enthalten. Dies würde trotzdem noch auf eine mögliche hohe Schemaähnlichkeit hinweisen. Deshalb wird die Ähnlichkeit der Namensräume und nicht nur ihre Gleichheit betrachtet. Die Berechnung der Ähnlichkeit erfolgt auch hier mit dem n-gram Matcher.
Die Ähnlichkeit der Schemata selber kann durch erdenklich viele Varianten ermittelt werden, die sich z.B. auf die Schemaelemente oder die Struktur stützen. Für diese Arbeit wurde ein Ähnlichkeitswert berechnet, der die Ähnlichkeitswerte für die einzelnen Elementnamen der Schemata kombiniert. Diese Einzelwerte werden mit dem Name-Matcher ermittelt.
Jeder unvollständige Mappingpfad erhält einen Wert, der für die Ähnlichkeit der Schemata der fehlenden Mappings steht und somit die Schwierigkeit des zu lösenden Matchproblems repräsentiert. Ist der Ähnlichkeitswert hoch, so sind die Schemata einander

sehr ähnlich und es liegt wahrscheinlich eine leicht zu lösende Matchaufgabe vor. Pfade die einen hohen Wert haben, sind zu bevorzugen, da sie vermutlich Mappings von höherer Qualität liefern.

5.2.3 Unkombinierte Mapping-Wiederverwendung

Für ein gegebenes Matchproblem können mehrere vollständige bzw. unvollständige Mappingpfade existieren. Bei der unkombinierten Mapping-Wiederverwendung werden die Mappings, die durch die Berechnung eines Mappingpfades entstehen, direkt an den Benutzer weitergeleitet. Dieses Verfahren ist optimistisch, da jede durch MatchCompose erzeugte Korrespondenz weitergeleitet wird. Diese Strategie eignet sich für die Interaktion mit dem Nutzer, wenn dieser einen Mappingpfad auswählt und für diesen dann das entsprechende Mapping zurückgeliefert bekommt.

5.2.4 Kombinierte Mapping-Wiederverwendung

Eine andere Möglichkeit bildet die kombinierte Mapping-Wiederverwendung, bei der die für jeden Pfad erhaltenen Mappings miteinander kombiniert werden. Statt der n Mappings bei n Mappingpfaden wird nur genau *ein* Mapping an den Benutzer weitergereicht.
Zuerst wird also mit Hilfe von MatchCompose für jeden Mappingpfad ein Mapping berechnet, wobei für einen unvollständigen Pfad natürlich zuerst die fehlenden Mappings erzeugt werden müssen. Darauffolgend werden die erzeugten Mappings miteinander zu einem neuen Mapping kombiniert. Dadurch werden Korrespondenzen, die in mehreren Mappings vorhanden und somit sehr wahrscheinlich korrekt sind, beibehalten. Korrespondenzen, die nur wenige Male oder nur einmal auftauchen, sind wahrscheinlich falsch und werden durch das Kombinieren herausgefiltert.
Der Kombinationsschritt, der für jede Korrespondenz aus den Ähnlichkeitswerten einen kombinierten Wert erzeugt, entspricht der Aggregation aus Abschnitt 3.4. Als Aggregationsstrategien stehen zur Verfügung:

- Min = Minimum aller Ähnlichkeitswerte
- MW = Mittelwert aller Ähnlichkeitswerte
- Max = Maximum aller Ähnlichkeitswerte

5.2.5 Verwendung eines Pivotschemas

Die Verwendung eines Pivotschemas wurde in Abschnitt 4.3 vorgestellt. Im Idealfall sind für ein gegebenes Matchproblem für beide Schemata *S1* und *S2* Mappings zum Pivotschema bereits vorhanden. In diesem Fall ergibt sich ein vollständiger Mappingpfad der Länge 2, nämlich $S1 \leftrightarrow PS \leftrightarrow S2$. Dies entspricht der vollständigen Mapping-Wiederverwendung vom Abschnitt 5.2.1, nur das in diesem Fall das Pivotschema als Zwischenschema dient.

Die betrachteten Mappingpfade über das Pivotschema sind immer von der Länge 2. Dies gilt unter der Annahme, dass jedes Schema, für das bereits ein Mapping erzeugt wurde, zum Pivotschema gematcht wurde. Längere Mappingpfade wären ineffizient, da es immer einen kürzeren Mappingpfad gibt.
Kann es jedoch vorkommen, dass es Mappings für zwei Schemata gibt, die nicht über das Pivotschema abgewickelt wurden, so können auch für diese Wiederverwendungsstrategie sinnvoll längere Mappingpfade gebildet werden. Dabei ist dann natürlich das Pivotschema nur eines der Zwischenschemata. Gibt es beispielsweise für das Matchproblem *S1-S2* zwar ein Mapping $S1 \leftrightarrow PS$, aber nicht $PS \leftrightarrow S2$, so könnte mit den vorhandenen Mappings $PS \leftrightarrow Si$ und $Si \leftrightarrow S2$ der vollständige Mappingpfad $S1 \leftrightarrow PS \leftrightarrow Si \leftrightarrow S2$ gebildet werden. Es müsste also keine Matchaufgabe gelöst werden, sondern lediglich MatchCompose auf einem etwas längeren Pfad ausgeführt werden.

Gibt es für das Matchproblem *S1-S2* eines der beiden Mappings zum Pivotschema noch nicht, dann ergibt sich ein unvollständiger Mappingpfad der Länge 2_1, nämlich $S1 \leftrightarrow PS \leftrightarrow S2$, wobei die Matchaufgabe entweder $S1 - PS$ oder $PS - S2$ lautet. Dies ist dann unvollständigen Mapping-Wiederverwendung vom Abschnitt 5.2.2 mit dem Pivotschema als Zwischenschema.
Da das Erzeugen von einem Mapping zwischen einem neuen Schema zu dem Pivotschema einer eigenen Matchaufgabe entspricht, kann es praktikabel sein, auch dafür die Wiederverwendung anzuwenden.
Es sei angenommen, dass die Matchaufgabe *S1-S2'* lautet und das Mapping $S1 \leftrightarrow PS$ schon existiert, während $S2' \leftrightarrow PS$ noch zu erstellen ist. Weiterhin sei das neue Schema *S2'* eine neuere Version von *S2*, für welches das Mapping $S2 \leftrightarrow PS$ zum Pivotschema existiert. So ist es durchaus vernünftig, das Matchproblem *S2'-PS* durch den unvollständigen Mappingpfad $PS \leftrightarrow S2 \leftrightarrow S2'$ mit dem Matchproblem *S2-S2'* zu ersetzen, da letzteres ein "leichtgewichtiges" Machtproblem darstellt. Insgesamt würde so statt dem unvollständigen Mappingpfad der Länge 2_1 $S1 \leftrightarrow PS \leftrightarrow S2'$ ein Pfad Länge 3_1 $S1 \leftrightarrow PS \leftrightarrow S2 \leftrightarrow S2'$ abgearbeitet werden.

Im Extremfall gibt es für beide Schemata kein Mapping zum Pivotschema. Es gibt dann zwei Matchaufgaben, $S1 - PS$ und $PS - S2$. Da jedoch keine Wiederverwendung stattfindet, wird der Fall für diese Arbeit außer Betracht gelassen.

Dem Pivotschema kommt natürlich eine sehr wichtige Rolle zu. Es muss einerseits alle wichtigen Elemente aller Schemata abbilden, denn sobald ein Attribut nicht vorhanden ist, kann dazu keine Korrespondenz erstellt werden. Andererseits sollte ein Pivotschema möglichst keine Redundanzen aufweisen. Dies würde sich beim Erstellen von Mappings zum Pivotschema negativ auswirken, da auch "redundante Korrespondenzen" erzeugt werden müssten. Dies entspricht 1:n Korrespondenzen, die jedoch nicht in jedem Matchprozess erzeugt werden. Es würden dann bei der Anwendung von MatchCompose Korrespondenzen nicht erzeugt werden und somit verloren gehen.
Für jedes neue Schema, dass importiert wird, müsste für optimale Ergebnisse das Pi-

votschema angepasst werden. Dies beinhaltet das Einfügen von Attributen und Strukturen, die bisher vom Pivotschema nicht abgebildet werden. Dabei stellt sich zuerst das Problem, wie Attribute und Strukturen denn als "neu" identifiziert werden. Daraufhin ist zu entscheiden, wo welche Änderungen im Pivotschema vorgenommen werden müssen. Je größer ein Pivotschema durch solche Erweiterungen wird, desto größer ist auch die Gefahr, bei einem Mapping von einem neuen Schema zu dem Pivotschema falsche Korrespondenzen zu berechnen oder gar richtige zu übersehen. Im Rahmen dieser Arbeit kann jedoch nicht genauer auf die auftauchenden Probleme mit der Verwendung eines Pivotschemas eingegangen werden.

5.2.6 Bildung von Wiederverwendungsstrategien

Die vorgestellten Möglichkeiten der Wiederverwendung lassen sich miteinander kombinieren und ergeben so vier verschiedene Wiederverwendungsstrategien:

- vollständige, unkombinierte Mapping-Wiederverwendung
- unvollständige, unkombinierte Mapping-Wiederverwendung
- vollständige, kombinierte Mapping-Wiederverwendung
- unvollständige, kombinierte Mapping-Wiederverwendung

In Abbildung 5.3 sind die Wiederverwendungsstrategien und ihr Zusammenhang abgebildet. Ausgehend von einem Matchproblem können vollständige und unvollständige Mappingpfade bestimmt werden. Für die unvollständigen Mappingpfade kann zusätzlich eine Selektion nach Ähnlichkeitskriterien erfolgen. Wird ein Pivotschema verwendet so entspricht dies der Filterung von den Mappingpfaden. Es erfolgt eine Beschränkung auf die Mappingpfade, die vom Ausgangsschema des Matchproblems direkt über das Pivotschema zum Zielschema führen.
Für die Berechnung eines Mappings aus einem gegebenen Mappingpfad wird die MatchCompose-Operation verwendet, wobei für unvollständige Mappingpfade zuerst die fehlenden Mappings berechnet werden müssen. Das Ergebnismapping kann dann direkt - unkombiniert - an den Nutzer zurückgeliefert werden. Werden mehrere solcher Ergebnismappings berechnet, so lassen sich diese durch eine Aggregationsstrategie zu einem einzigen Mapping kombinieren.

5.3 Integration in COMA++

In diesem Abschnitt wird beschrieben, wie die neu entwickelten Wiederverwendungsstrategien in COMA++ integriert sind. Es wird der Reusematcher vorgestellt, der die automatische Wiederverwendung ermöglicht. Außerdem wird die Verwendung des Pivotschemas erläutert. Eine schrittweise Wiederverwendung ist dialoggestützt möglich, wird jedoch im Zusammenhang mit der grafischen Benutzeroberfläche in Kapitel 7 beschrieben.

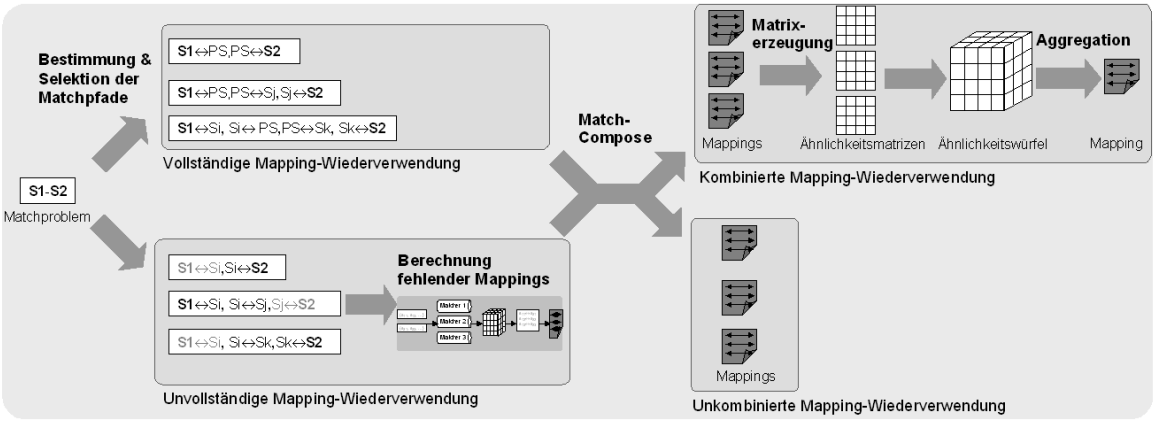


Abbildung 5.3: Die Wiederverwendungsstrategien

5.3.1 Reusematcher

Der Reusematcher kann für den automatischen Modus benutzt werden und entweder der einzeln ausgeführt oder auch mit anderen Matchern kombiniert werden. Wird der Matcher alleine ausgeführt, so erzeugt er ein Mapping für das gegebene Matchproblem. Soll das Ergebnis verschiedener Matcher miteinander kombiniert werden, erzeugt jeder Matcher - so auch der Reusematcher - eine Ähnlichkeitsmatrix.

Wie schon in Abschnitt 5.2.1 vermutet und in Kapitel 6 gezeigt wird, nimmt mit zunehmender Länge der Mappingpfade die Qualität des erzeugten Mappings ab. Daher erscheint es sinnvoll, sich zuerst auf die kürzen Pfade zu beschränken, wenn es sie gibt.
Außerdem wird durch das Kombinieren mehrerer Mappings eine höhere Qualität erreicht, wie ebenfalls in Kapitel 6 gezeigt wird. Gibt es also mehrere Mappingpfade einer Länge, werden diese miteinander kombiniert.

Der Matcher bestimmt zuerst alle vollständigen Mappingpfade der Länge 2. Wenn es diese gibt, berechnet er sie und vereinigt die nicht leeren Matchergebnisse der MatchCompose-Operation zu einem neuen Ergebnis. Diese liefert er dann zurück.
Gibt es keine vollständigen Mappingpfade der Länge 2 oder sind alle Matchergebnisse leer, dann bestimmt der Matcher alle unvollständigen Mappingpfade der Länge 3. Hier berechnet er wiederum mit MatchCompose die einzelnen Matchergebnisse, kombiniert diese und liefert das Ergebnis zurück.
Gibt es keine vollständigen Mappingpfade der Länge 2 oder sind alle Matchergebnisse leer, dann nimmt er sich die vollständigen Mappingpfade der Länge 4 vor und verfährt mit ihnen ebenso. Sollte es auch hier keine Pfade geben oder sollten alle Matchergebnisse der MatchCompose-Operation leer sein, so liefert der *Reusematcher* ein leeres Ergebnis.

5.3.2 Verwendung eines Pivotschemas

Um den Arbeitsrahmen einzuhalten, kann als Pivotschema nur ein vorhandenes Schema für die jeweilige Domäne verwendet werden. Es wird kein Pivotschema erstellt. Auch die komplexe Aufgabe der Anpassung des Pivotschemas an neue Schemata, das heißt die Erweiterung um noch nicht vorhandene Elemente und Strukturen ohne die Erzeugung von Redundanzen wird in diesem Rahmen nicht eingehender betrachtet.

Über die GUI ist es möglich ein bereits importiertes Schema als Pivotschema zu selektieren. Diese Auswahl kann jederzeit geändert werden, oder es kann auch ein Arbeiten ohne Pivotschema festgelegt werden.
Ist ein Pivotschema bestimmt worden, hat dies Einfluss auf jegliche Wiederverwendung von Mappings. So werden bei der dialoggestützten Wiederverwendung nur Mappingpfade angezeigt, die über das Pivotschema führen, und auch der Reusematcher beschränkt sich auf diese Pfade. Dementsprechend sollte das ausgewählte Pivotschema also kein Schema des aktuellen Matchproblems sein.

6 Evaluation

Die im Kapitel 4 vorgestellten Wiederverwendungsstrategien werden in diesem Teil der Arbeit evaluiert. Zunächst werden die Testszenarien, Maße zur Bewertung der Ergebnisse und die Testumgebung vorgestellt. Zuerst werden die Ergebnisse betrachtet, die mit den Matchstrategien AllContext und FilteredContext ohne Wiederverwendung erreicht werden. Darauffolgend werden die unterschiedlichen Wiederverwendungsstrategien mit den erzielten Ergebnissen vorgestellt. Es soll herausgearbeitet werden, welche Variante in Bezug auf Mappingpfadlänge, Matchstrategie bzw. Aggregationsstrategie für jede Wiederverwendungsstrategien die besten Ergebnisse erbringt. Außerdem erfolgt jeweils ein Vergleich zu den Matchstrategien, die keine Wiederverwendung benutzen. Dies alles wird in den Abschnitten 6.4 bis 6.7 behandelt. In Abschnitt 6.8 wird die Selektion von Matchaufgaben untersucht und näher betrachtet, welche der drei untersuchten Kriterien die gewünschte Qualitätsbesserung mit sich bringt und wie hoch der Aufwand zur Berechnung der Ähnlichkeit überhaupt ist. Den Effekt der Verwendung eines Pivotschemas wird schließlich in Abschnitt 6.9 für die unkombinierte Mapping-Wiederverwendung geprüft. Die Qualität und die Ausführungszeiten der untersuchten Strategien werden in Abschnitt 6.10 verglichen. Eine abschließende Diskussion der Ergebnisse erfolgt dann in Abschnitt 6.11.

6.1 Testschemata und Testaufgaben

Es wurden jeweils 15 Testszenarien durchgespielt. Die Ausgangs- und Zielschemata waren entweder CIDXPO, OpenTrans, Excel, Noris, Paragon oder Apertum, deren wichtigsten Kenngrößen in Tabelle 6.1 aufgelistet sind.

Es ergeben sich 15 Testszenarien, die in Tabelle 6.2 mit einer Testnummer von T1 bis T15 versehen sind. Die wichtigsten Charakteristika sind in obiger Tabelle aufgeführt.

Die Testszenarien wurden so angeordnet, dass die kleinste Testnummer auch der geringsten Komplexität entspricht, wobei diese hierfür als Pfadsumme der beiden Schemata berechnet wird. Die ersten zehn Testszenarien befinden sich zwischen zwei ungefähr gleich großen Schemata und haben eine Pfadsumme zwischen 82 und 213. Die Tests T11 bis T15 beinhalten ein kleines und ein großes Schema (OpenTrans) und die Pfadsummen liegen über 2500.
Für jedes Testmapping existiert jeweils bereits ein Mapping, welches manuell bestimmt wurde und nur korrekte Korrespondenzen enthält. Dieses Mapping ist in den folgenden Testaufgaben das *Zielmapping*, welches als Maßstab für ein berechnetes Mapping

Schema	#Knoten			# Pfade	Pfadlänge	
	alle	Inneren	Blätter		Max	Mittelwert
Apertum	74	22	52	136	5	3,6
CIDXPO	27	7	20	34	4	2,9
Excel	32	9	23	48	4	3,5
Noris	46	8	38	65	4	3,2
OpenTrans	195	85	110	2500	11	7
Paragon	59	11	48	77	6	3,6

Tabelle 6.1: Testschemata und ihre Kenngrößen

genommen wird. Im Idealfall wird von einem Matcher ein Mapping erzeugt, welches identisch mit dem gewünschten Zielmapping ist. Im ungünstigsten Fall enthält das erzeugte Mapping keine Korrespondenz des Zielmappings, sondern nur falsche.
Die Größe der Zielmappings liegt bei allen Testszenarien zwischen 26 und 94. Das ist damit zu erklären, dass die Anzahl der Korrespondenzen meist abhängig von der Anzahl der Pfade des kleineren Schemas ist, auch wenn n:m Mapping vorkommen können.
Der Dice-Koeffizient aus Abschnitt 3.4 liefert das Verhältnis der Elementanzahl, welche gemacht werden können, zur Gesamtanzahl der Mengenelemente. Dieser Koeffizient schwankt schon bei den ersten zehn Tests, jedoch nicht mit erkennbaren Zusammenhang zur Komplexität der Probleme. Bei den fünf komplexesten Tests jedoch ist der Dice-Wert sehr klein, da es verhältnismäßig wenig Korrespondenzen im Zielmapping gegenüber den vorhanden Elementen gibt.

6.2 Maße

Um die Qualität der erzeugten Mappings und damit der genutzten Operationen evaluieren zu können, benötigt man Maße. Hierfür sollen einige Maße aus [DR02], [DMR03] genutzt werden: Precision, Recall, Fmeasure und Overall. Diese basieren auf Matchmengen, die in Abbildung 6.1 dargestellt werden.

R $= A \cup B$ steht für die Anzahl der Korrespondenzen, die korrekt sind und gefunden werden sollen *(Real matches)*. $R \subseteq M$ und $|R| \in \mathbb{N}$.

P $= B \cup C$ steht für die Anzahl der Korrespondenzen, die tatsächlich von einer Operation erzeugt wurden *(Processed matches)*. $P \subseteq M$ und $|P| \in \mathbb{N}$.

A $= R \backslash B$ sind die Korrespondenzen, die korrekt sind, aber nicht von der Operation gefunden wurden *(False Negatives, Missed Matches)*. $A \subseteq R$ und $|A| \in \mathbb{N}$.

B $= R \backslash A = P \backslash C$ sind diejenigen Korrespondenzen, die korrekt identifiziert wurden *(True positives)*. $B = R \cap P$ und $|B| \in \mathbb{N}$.

Test	Schema *S1*	Schema *S2*	Korrespondenzen im Zielmapping	*s1*-Pfade überdeckt/alle	*s2*-Pfade überdeckt/alle
T1	CIDXPO	Excel	35	27/34	35/48
T2	CIDXPO	Noris	26	25/34	26/65
T3	CIDXPO	Paragon	35	28/34	35/77
T4	Excel	Noris	44	40/48	32/65
T5	Excel	Paragon	46	38/48	36/77
T6	Noris	Paragon	43	33/65	40/77
T7	CIDXPO	Apertum	48	28/34	48/136
T8	Excel	Apertum	67	42/48	55/136
T9	Noris	Apertum	79	50/65	79/136
T10	Paragon	Apertum	60	52/77	55/136
T11	CIDXPO	OpenTrans	36	28/34	36/2500
T12	OpenTrans	Excel	45	42/2500	40/48
T13	OpenTrans	Noris	47	43/2500	39/65
T14	OpenTrans	Paragon	53	51/2500	48/77
T15	OpenTrans	Apertum	94	68/2500	91/136

Tabelle 6.2: Testszenarien

C $= P \backslash B$ sind die Korrespondenzen, die zwar von der Operation gefunden wurden, aber nicht korrekt sind *(False Positives, False Matches)*. $C \subseteq P$ und $|C| \in \mathbb{N}$.

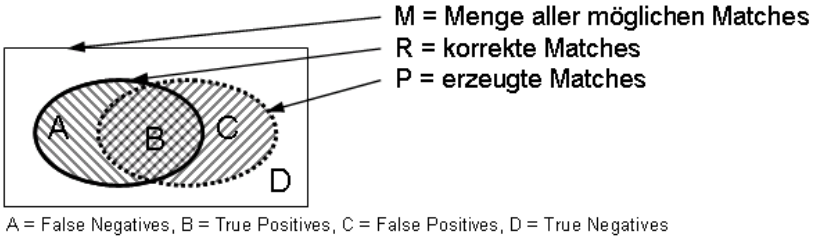


Abbildung 6.1: Übersicht zu Matchmengen

Precision steht$= \frac{|B|}{|P|} = \frac{|B|}{|B|+|C|}$ schätzt die Zuverlässigkeit des Mappingvorschlages ab. $Precision \in \Re$, wobei gilt $0 \leq Precision \leq 1$.

Recall $= \frac{|B|}{|R|} = \frac{|B|}{|A|+|B|}$ spezifiziert den Anteil der korrekten Korrespondenzen, die gefunden wurden. $Recall \in \Re$, wobei gilt $0 \leq Recall \leq 1$.

Im Idealfall enthält ein erzeugtes Mapping keine falsch erkannten Matches, $|C| = 0$, und beinhaltet alle korrekten Matches, $A = B$, so dass gilt $Precision = Recall = 1$. Die Maße Precision und Recall können jedoch jeweils nicht alleine die Matchqualität

bewerten. So lässt sich für Recall leicht ein hoher Wert nahe 1 erreichen, indem einfach (fast) alle möglichen Matches zurückgeliefert werden, was auf Kosten eines kleinen Precision-Wertes geht. Ein hoher Wert für Precision lässt sich erreichen, indem nur wenige korrekte Matches berechnet werden, weshalb sich ein kleiner Recall-Wert ergibt. Deshalb wurden Maße wie Fmeasure und Overall eingeführt, die eine Kombination von Precision und Recall darstellen und so zur Abschätzung der Matchqualität dienen.

Fmeasure $= 2 * \frac{Precision * Recall}{Precision + Recall}$ wird benutzt, um die Matchqualität zu bestimmen und hängt im gleichen Maße von Precision und Recall ab. $Fmeasure \in \Re$, wobei gilt $0 \leq Fmeasure \leq 1$.

Overall $= 1 - \frac{|A|+|C|}{|R|} = \frac{|B|-|C|}{|R|} = Recall * \left(2 - \frac{1}{Precision}\right)$ verkörpert ein kombiniertes Maß für die Matchqualität. $Overall \in \Re$, wobei gilt $Overall \leq 1$.

Das Fmeasure stammt aus dem Gebiet des Information Retrieval und stellt in dieser hier vorgestellten Version die harmonische Mitte zwischen Precision und Recall dar.
Das Maß Overall wurde als *Accuracy* in [MGMR02] eingeführt und direkt im Kontext des Schema-Matching entwickelt. Es basiert auf der Idee abzuschätzen, welcher Aufwand nach dem Erzeugen des Mappings entsteht, aus diesem das gewünschte, korrekte Mapping zu entwickeln. Diese Nachbearbeitung beinhaltet das Entfernen falscher Matches und das Hinzufügen fehlender Matches. Besitzt das Overall einen negativen Wert, so ist der Aufwand, aus dem erzeugten Mapping das gewünschte Mapping zu erzeugen, höher, als wenn man ohne solch ein Mapping diese Arbeit erledigt.
Wie in [DMR03] gezeigt wurde, ist Fmeasure im Verhalten optimistischer als Overall. Bei den gleichen Werten von Precision und Recall ist der Wert von Fmeasure größer als der von Overall, wobei beide maximal den Wert 1 annehmen, wenn sowohl $Precision = 1$ als auch $Recall = 1$. Im Gegensatz zu den anderen Maßen kann Overall jedoch negative Werte annehmen, genau dann, wenn es mehr falsche gefundene Matches gibt als richtig entdeckte und somit $Precision < 0.5$.
Dies ist auch in Abbildung 6.2 zu sehen.

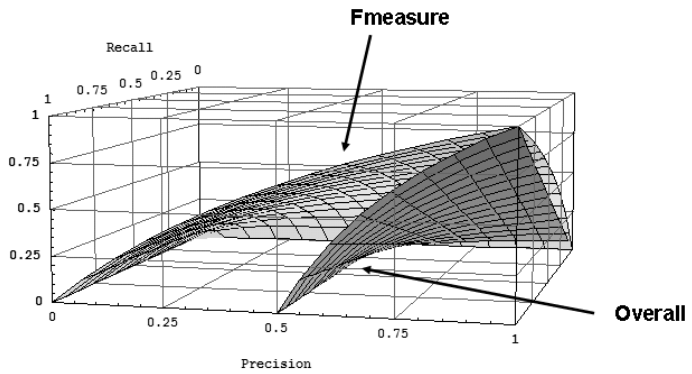


Abbildung 6.2: Fmeasure und Overall in Abhängigkeit von Recall und Precision

Es sei angenommen, dass es ein Mapping gibt, welches durch eine Matchoperation erzeugt wurde. Intuitiv erscheint es, dass sowohl die False Negatives als auch die False Positives die Mappingqualität reduzieren. Die Qualität wäre am höchsten, wenn jede korrekte Korrespondenz gefunden wurde ($R = P = B$) und keine falschen dabei sind ($A = 0$,$C = 0$). Dann wären sowohl Precision, Recall, Fmeasure als auch Overall 1. Würde das erzeugte Mapping jeden möglichen Match enthalten und somit der Menge M aus der Abbildung 6.1 entsprechen, so erhielte man einen sehr guten Recall-Wert ($= 1$), da Menge B in Menge M steckt. Da jedoch sehr viele falsche Matches damit im Mapping sind, würden die Werte für Precision und Fmeasure nahe 0 sein und Overall einen sehr großen negativen Wert besitzen. Allerdings sollte man auch nicht restriktiv sein. Dann wäre zwar der Wert für die Precision gut, da die erzeugten Matches fast alle richtig sind, aber dafür hätte man an sich wahrscheinlich kaum welche gefunden. Die Werte für Recall und Fmeasure wären klein, für Overall auch negativ. Am besten sollte man also versuchen, weder zu optimistisch noch zu restriktiv sein. Damit würden alle Maße gut bis sehr gut Werte besitzen, was angestrebt wird.

Für die in den folgenden Abschnitten ausgeführten Testszenarien werden für jede einzelne Matchaufgabe Precision, Recall, Fmeasure und Overall bestimmt. Über all diese Werte wird dann der Mittelwert ermittelt und außerdem Minimum bzw. Maximum, um die Qualitätsvariation der angewendeten Strategie abzuschätzen.

6.3 Matchen ohne Wiederverwendung

Abbildung 6.3 zeigt die Durchschnittswerte für die Mappings, die ohne Wiederverwendung durch Ausführung der Matchstrategien AllContext und FilteredContext erzeugt wurden. Diese Matchstrategien wurden mit den in Abschnitt 3.6 vorgestellten Standardeinstellungen verwendet. FilteredContext liefert das bessere Ergebnis mit einem durchschnittlichen Fmeasure-Wert von 0,73, wobei 85% aller gefundenen Korrespondenzen korrekt sind und immerhin 64% aller gesuchten Korrespondenzen im Ergebnis sind. Durch das Vorgehen von FilteredContext nur Kontexte von Knoten zu vergleichen, die ohnehin einander ähnlich sind, erfolgt eine Filterung. Diese sorgt dafür, dass weniger falsche Korrespondenzen berechnet werden. Allerdings werden so auch weniger korrekte Korrespondenzen gefunden als von AllContext, da diese Strategie alle Knoten inkl. der Kontexte miteinander vergleicht. Die Fehlerleisten zeigen das Minimum und Maximum jeder Strategie an, welches aus den Ergebnissen aller erzeugter Mappings ermittelt wurde. Insgesamt ist zu erkennen, dass FilteredContext stabiler in der Qualitätsvariation der Ergebnisse ist, während *AllContext* auch zu schlechten Ergebnissen führen kann, wenn es sehr viele falsche Korrespondenzen berechnet hat.

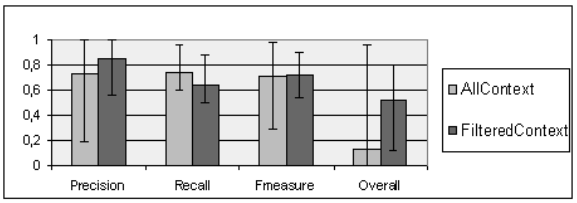


Abbildung 6.3: Ergebniswerte für die Matchstrategien AllContext und FilteredContext - ohne Wiederverwendung

6.4 Vollständige, unkombinierte Mapping-Wiederverwendung

Es wurden die 15 Testszenarien durchlaufen und für jedes Matchproblem alle möglichen vollständigen Mappingpfade der Länge 2, 3 und 4 berechnet, die existieren. Für die Länge 2 gibt es jeweils vier, für die Länge 3 gibt es zwölf und für Länge 4 gleich 24 Mappingfade. Aus den erwähnten einzelnen Möglichkeiten wurde der Durchschnitt ermittelt aus den Einzelwerten vom Recall, Precision und Fmeasure. Dabei ist der Fall, dass bei wenigen Werten, wie z. B. bei Länge 2 mit vier Werten jeder Wert stark den Durchschnitt beeinflusst - in negativer oder positiver Gewichtung.

Abbildung 6.4 zeigt, dass die vollständige, unkombinierte Mapping-Wiederverwendung

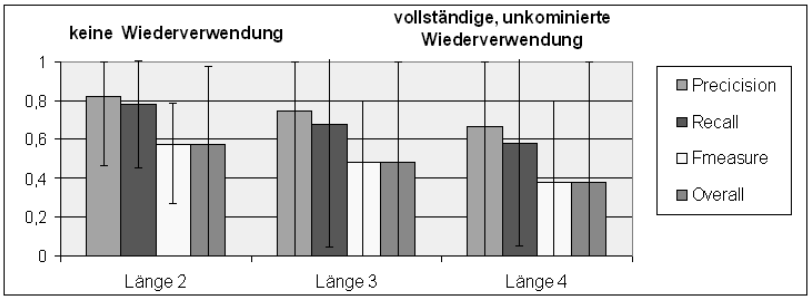


Abbildung 6.4: Ergebniswerte der vollständigen, unkombinierten Mapping-Wiederverwendung

mit Länge 2 am besten mit Fmeasure 0,78 und Overall 0,58 abschneidet. Außerdem werden bei Länge 2 75% aller gesuchten Korrespondenzen gefunden und 82% aller gefundenen Matches sind korrekt. Die Abbildung lässt darüber hinaus erkennen, dass die Annahme aus Abschnitt 5.2.1, dass mit zunehmender Länge der vollständigen Mappingpfade auch die Ergebnisqualität abnimmt, bestätigt wird. Wird MatchCompose also über mehrere Mappings ausgeführt, so werden mehr falsche und weniger korrekte Korrespondenzen erzeugt. Kürzere Mappingpfade sollten also für die Wiederverwendung bevorzugt werden.

Jedoch lässt sich in Abbildung 6.4 an den Fehlerleisten erkennen, die das Minimum

und Maximum der Werte aller erzeugten Mappings darstellen, dass die Ergebnisse stark schwanken können. Die Nullwerte bei Länge 3 und 4 ergeben sich daher, dass MatchCompose auf bestimmte Mappingpfade ein leeres Mapping - also ohne eine Korrespondenz - erzeugt hat. Dies kann der Fall sein, wenn die Mappings unterschiedliche Elemente des Schemata abdecken. Diese große Varianz der Ergebnisse ist negativ, da der Benutzer vorher damit nur schlecht einschätzen kann, welche Qualität das erzeugte Mapping haben wird.

6.5 Vollständige, kombinierte Mapping-Wiederverwendung

Bei der vollständigen, kombinierten Mapping-Wiederverwendung werden die einzelnen Mappings, die mit Hilfe von MatchCompose auf vollständigen Mappingpfaden berechnet werden, für jede Länge miteinander kombiniert. Wie bereits in Abschnitt 5.2.4 beschrieben, wurde für die Kombination nur der Aggregationsschritt ausgeführt. Drei Aggregationsstrategien wurden getestet: Min (Minimum), MW (Mittelwert) und Max (Maximum). Es wurden drei Mappingmengen zu jeweils einem Mapping kombiniert, nämlich die Mappings der vollständigen Mappingpfade einer Länge:

- der Länge 2 mit 4 vollständigen Mappingpfaden
- der Länge 3 mit 12 vollständigen Mappingpfaden
- der Länge 4 mit 24 vollständigen Mappingpfaden

Abbildung 6.5 zeigt die Werte für die unterschiedlichen Mappingmengen, die mit diesen Aggregationsstrategien zu einem Mapping kombiniert wurden. Dabei werden für jede untersuchte Längedie Werte aller drei Aggregationsstrategie dargestellt. Die besten Ergebnisse liefert die Länge 2 mit der Aggregationsstrategie *MW* - nämlich für Fmeasure 0,80 und für Overall 0,64. Es werden 74% aller gesuchten Korrespondenzen gefunden und 88% aller gefundenen Korrespondenzen sind korrekt. Da schon im vorherigen Abschnitt gezeigt wurde, dass die kürzeren Mappingpfade im Durchschnitt die besten Ergebnisse haben, überrascht es nicht, dass dies mit den kombinierten Mappings ebenso ist. Besitzen die Mappings, die kombiniert werden, eine hohe Qualität, so gilt dies ebenso für das kombinierte Ergebnismapping. Außerdem ist die Qualität der kombinierten Mappings stabiler und weist eine geringere Varianz auf als bei den unkombinierten Mappings.

Die drei untersuchten Aggregationsstrategien weisen unterschiedliche Tendenzen auf. Min hat die höchsten Precision-Werte jedoch die niedrigsten Recall-Werte. Dies liegt begründet in dem pessimistischen Vorgehen, das Minimum der Ähnlichkeitswerte für eine Korrespondenz zu nehmen. Es kommen somit nur die Korrespondenzen ins Endergebnis, die in jedem Mappingpfad gute Ähnlichkeitswerte haben und sehr wahrscheinlich korrekt sind. Auf der anderen Seite fallen jedoch Korrespondenzen raus,

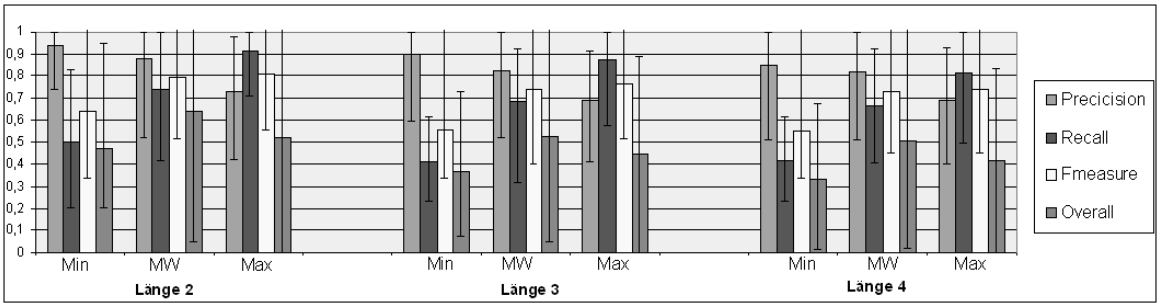


Abbildung 6.5: Ergebniswerte der vollständigen, kombinierten Mapping-Wiederverwendung

sobald sie in einem der Mappings einen schlechten Wert haben, obwohl diese nicht unbedingt falsch sein müssen. Mit Min werden also nicht so viele, aber (fast) nur korrekte Korrespondenzen gefunden.
Bei der optimistischen Strategie Max wird eine Korrespondenz mit dem höchsten Wert ins Endergebnis genommen, welche sie in einem der kombinierten Mappings hat. Dadurch werden alle korrekten Mappings beibehalten, woraus die hohen Recall-Werte resultieren. Aus demselben Grund werden jedoch auch falsche Korrespondenzen beibehalten, da bereits ein guter Ähnlichkeitswert ausreicht, um ins Endmapping übernommen zu werden. Dies drückt sich in den niedrigen Precision-Werten aus. Mit Max werden sehr viele korrekte aber auch etliche falsche Korrespondenzen berechnet.
MW geht den Mittelweg und berechnet den durchschnittlichen Ähnlichkeitswert. Das verhindert, dass Korrespondenzen wegen *eines* schlechten Ähnlichkeitswertes nicht bzw. wegen *eines* hohen Wertes ins Endmapping doch übernommen werden. Das durch die Anwendung dieser Aggregationsstrategie entstandene Mapping enthält viele korrekte und nur einige falsche Korrespondenzen.

6.6 Unvollständige, unkombinierte Mapping-Wiederverwendung

Bei der unvollständigen, unkombinierten Mapping-Wiederverwendung wurden die unvollständigen Mappingpfade der Länge 2_1, 3_1 und 3_2 getestet. Die fehlenden Mappings der Mappingpfade wurden mit den Matchstrategien AllContext und FilteredContext erzeugt, die keine Wiederverwendung beinhalten und auch schon in Abschnitt 6.3 verwendet wurden. Für jedes Testszenario gab es acht Mappingpfade der Länge 2_1, 24 der Länge 3_1 und zwölf der Länge 3_2. Zuerst wurden für den jeweiligen Mappingpfad die fehlenden Mappings erzeugt und mit Hilfe von MatchCompose aus diesen und den bereits vorhandenen Mappings ein neues Mapping erzeugt. Die Durchschnittswerte für die unterschiedlichen unvollständigen Mappingpfade sind in Abbildung 6.6 zu sehen. Das beste Ergebnis mit einem Fmeasure von 0,64 und einem

Overall von 0,41 liefern die unvollständigen Mappingpfade der Länge 2_1, wenn das fehlende Mapping mit FilteredContext erzeugt wird. Es sind 79% aller berechneten Korrespondenzen korrekt und es werden immer gut die Hälfte aller gesuchten Korrespondenzen gefunden. Die Kombination von den bereits vorhandenen und korrekten Mappings mit den automatisch erzeugten führt zu einem Qualitätsverlust bei unvollständigen Mappingpfaden. Der Qualitätsverlust von Länge 3_1 zu Länge 2_1 basiert allein auf dem weiteren Mapping, welches automatisch erzeugt wird. Da dieses auch eine geringere Qualität hat als ein bereits vorhandenes, so nimmt auch die Qualität des Ergebnismappings ab. Wie auch schon bei den vollständigen Mappingpfaden gilt also auch für die unvollständigen Mappingpfade, dass mit zunehmender Länge und mit zunehmender Anzahl der fehlenden Mappings die Qualität abnimmt. Außerdem lassen die Fehlerleisten eine hohe Varianz der Ergebnisqualität erkennen. Diese Strategie führt also zu sehr guten aber auch sehr schlechte Ergebnissen.

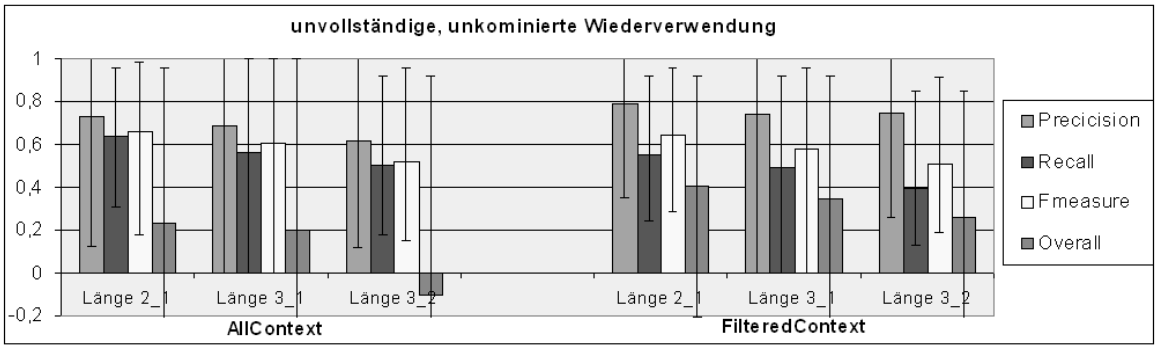


Abbildung 6.6: Ergebniswerte der unvollständigen, unkombinierten Mapping-Wiederverwendung für die Matchstrategien AllContext und FilteredContext

6.7 Unvollständige, kombinierte Mapping-Wiederverwendung

Bei der unvollständigen, kombinierten Mapping-Wiederverwendung werden die einzelnen Mappings, die für jeden unvollständigen Mappingpfad eines Matchproblems erzeugt werden, miteinander kombiniert. Wie auch schon bei der vollständigen, kombinierten Mapping-Wiederverwendung in Abschnitt 6.5 wurden drei verschiedene Aggregationsstrategien getestet: Min, MW und Max.
Es wurden drei Mappingmengen zu jeweils einem Mapping kombiniert, nämlich alle Mappings:

- der Länge 2_1 mit 8 unvollständigen Mappingpfaden
- der Länge 3_1 mit 24 unvollständigen Mappingpfaden

- der Länge 3_2 mit 12 unvollständigen Mappingpfaden

Die Ergebniswerte der Tests sind in Abbildung 6.7 für die unterschiedlichen Matchstrategien dargestellt, dabei werden für jede Mappingmenge die Ergebnisse aller drei Aggregationsstrategien angegeben. Die besten Werte bei der unvollständigen, kombinierten Mapping-Wiederverwendung werden mit der Länge 2_1 und der Aggregationsstrategie *MW* erreicht und zwar bei AllContext ein Fmeasure von 0,75 und ein Overall von 0,59, wogegen es bei FilteredContext mit 0,63 für Fmeasure und 0,47 für Overall etwas weniger ist.

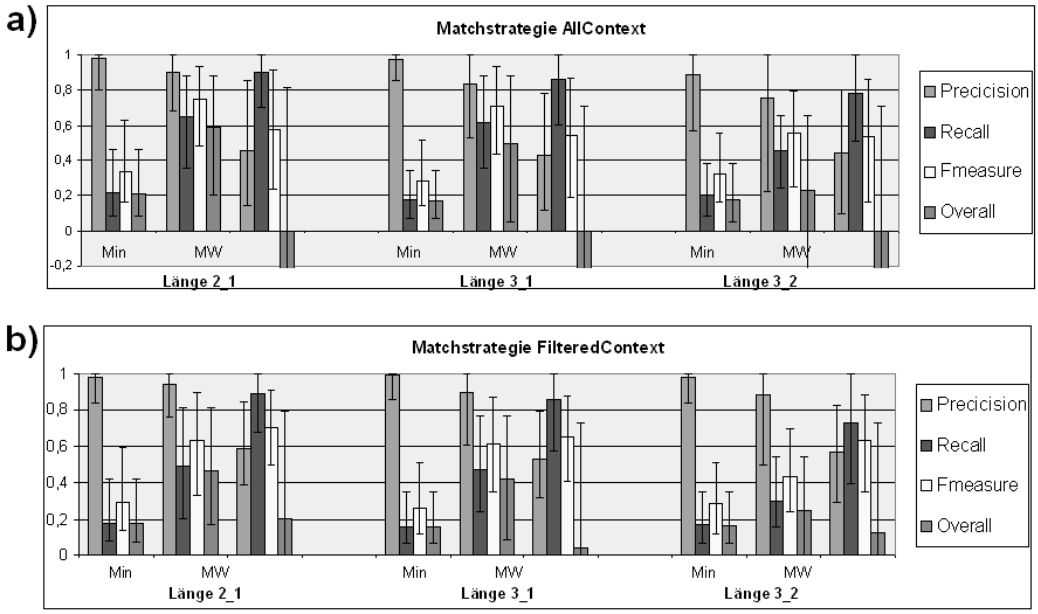


Abbildung 6.7: Die Werte der unvollständigen, kombinierten Mapping-Wiederverwendung für a) AllContext und b) FilteredContext

Es tauchen viele Effekte der vollständigen, kombinierten Mapping-Wiederverwendung auf. So ist die Varianz der Ergebnisse bei den kombinierten Ergebnissen deutlich geringer als bei den unkombinierten Ergebnissen. Durch das Aufnehmen nur der Korrespondenzen, die in jedem Teilmapping vorhanden sind, berechnet die Aggregationsstrategie Min zwar nur wenige Korrespondenzen, die jedoch sind zumeist korrekt. Mappings durch *Max* kombiniert beinhalten sehr viele korrekte jedoch auch viele falsche Korrespondenzen, da jede Korrespondenz nur mindestens in einem der Teilmappings vorhanden sein muss.

6.8 Selektion von Matchaufgaben

Während in den Abschnitten 6.6 und 6.7 alle unvollständigen Mappingpfade verwendet wurden, soll jetzt eine Selektion stattfinden, mit der die Qualität der Ergebnismap-

pings erhöht wird. Wie in Abschnitt 5.2.2 erläutert wurde, sollen nur noch Mappingpfade genutzt werden, deren Matchaufgaben "leichtgewichtig" sind. Dazu wird die Ähnlichkeit der Schemata unter der Annahme berechnet, dass Schemata, die sich ähnlicher sind, zu einem qualitativ höheren Mapping führen. Je besser die Qualität der einzelnen Mappings eines Mappingpfades ist, desto höher ist auch die Qualität des Endmappings.
Für diese Arbeit werden drei Möglichkeiten untersucht:

- Ähnlichkeit der Schemanamen
- Ähnlichkeit der Namensräume der Schemata
- Ähnlichkeit der Elementnamen

Im Folgenden werden die Auswirkungen der Selektion auf unvollständige Mappingpfade für die kombinierte Mapping-Wiederverwendung untersucht. Für die unkombinierte Mapping-Wiederverwendung würde sich direkt nichts ändern, da die Qualität der einzelnen Mappingpfade dieselbe bleibt. Es wird lediglich die Auswahl an möglichen Mappingpfaden kleiner. Um den Überblick zu verbessern, wird nur eine Aggregationsstrategie verwendet - und zwar MW, da diese bisher bei der kombinierten Mapping-Wiederverwendung die besten Ergebnisse erzielte.

6.8.1 Ähnlichkeit der Schemanamen

In der Tabelle 6.3 sind die Ähnlichkeitswerte für die Schemanamen aufgelistet. Viele der Schemanamen sind einander nicht oder nur sehr wenig ähnlich - ausgenommen der Fall, sie sind identisch. Aus diesem Grund wurde nur eine Selektion der Matchaufgaben auf die Namensähnlichkeit $\geq 0,05$ vorgenommen. Durch diese Selektion sind nur noch vier statt der 15 Matchaufgaben erlaubt. Dementsprechend fallen recht viele Mappingpfade heraus. Die Ergebnisse der unvollständigen, kombinierten Mapping-Wiederverwendung aller Mappingpfade bzw. nur mit ausgewählten Mappingpfaden ist in Abbildung 6.8 dargestellt.

	OpenTrans	CIDXPO	Apertum	Noris	Paragon	Excel
OpenTrans	1	0	0,095	0	0,10	0,11
CIDXPO	0	1	0,10	0	0	0
Apertum	0,10	0,10	1	0	0	0
Noris	0	0	0	1	0	0
Paragon	0,10	0	0	0	1	0
Excel	0,11	0	0	0	0	1

Tabelle 6.3: Ähnlichkeit der Schemanamen

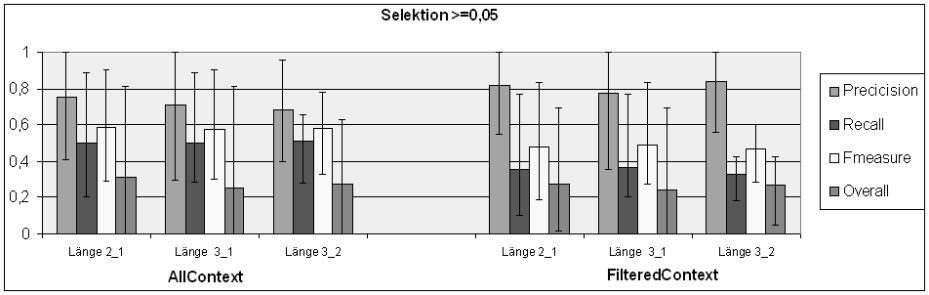


Abbildung 6.8: Ergebniswerte für die Selektion der unvollständigen Mappingpfade auf die Namensähnlichkeit

Für beide Matchstrategien verringert sich die durchschnittliche Qualität der Mappings deutlich, wenn man sich nur auf die selektierten Mappingpfade beschränkt. Eine Selektion auf der Basis der Ähnlichkeit der Schemanamen macht also für die gegebenen Testszenarien keinen Sinn.
Dies muss allerdings nicht bedeuten, dass es immer so ist. Angenommen es seien verschiedene Versionen von Schemata in den Tests vorhanden, die bis auf die Versionsnummer den gleichen Namen haben, also z.B. OpenTrans_v3.4 und OpenTrans_v4.0. In solchen Fällen würde eine Selektion über die Namensähnlichkeit auch zu ähnlichen Schemata führen. Generell jedoch ist die Reduzierung eines Schemas auf seinen Namen und damit auch der Ähnlichkeitstest von Schemata nur anhand dieser Information nur bedingt aussagefähig.

6.8.2 Ähnlichkeit der Namensräume

In Tabelle 6.4 sind die Schemata und ihre Namensräume[8] aufgeführt. Die Ähnlichkeitswerte zwischen den Namensräumen sind in Tabelle 6.5 angegeben.
Da die Ähnlichkeit der Namensräume durchaus unterschiedlich ist, wurden zwei Selektionen auf die Mappingpfade durchgeführt, nämlich $\geq 0,03$ und $\geq 0,3$. Bei der Selektion $\geq 0,03$ sind noch 13 bzw. bei $\geq 0,3$ nur sechs der 15 Matchaufgaben erlaubt. Dementsprechend fallen wenige bzw. viele Mappingpfade heraus.
In Abbildung 6.9 sind die Effekte der Selektionen zu sehen, die in diesem Fall zu Qualitätsverbesserungen führen. Die höchsten Werte für Fmeasure mit 0,72 und Overall mit 0,52 erreicht AllContext mit der Selektion $\geq 0,03$ auf Mappingpfade der Länge 2_1. Die Selektion auf $\geq 0,03$ führt zu einer Verbesserung besonders bei den Mappingpfaden der Länge 2_1. Dagegen bringt eine stärkere Selektion für Länge 2_1 nur eine ebenso gute Qualität, verbessert aber deutlich die Qualität für die längeren Pfade bzw. mit zwei Matchaufgaben. Allerdings gibt es auch negative Effekte. Während es bei der Selektion $\geq 0,03$ für jedes Testszenario Mappingpfade gibt, bleiben bei $\geq 0,3$

[8]Da CIDXPO, Apertum, Noris, Paragon und Excel aus einer XDR-Datei importiert wurden und dort keine Namensräume angegeben waren, wurden die dort vermerkten Bezeichnungen für das Schema stellvertretend als Namensräume verwendet.

Schema	Namensraum
OpenTrans	http://www.opentrans.org/XMLSchema/1.0
CIDXPO	CIDXPOSCHEMA.xml
Apertum	APERTUM_PO_1
Noris	NORIS_PurchaseOrder
Paragon	ParagonOrder
Excel	PurchaseOrder.biz

Tabelle 6.4: Die Namensräume der Testschemata

	OpenTrans	CIDXPO	Apertum	Noris	Paragon	Excel
OpenTrans	1	0	0,06	0	0,06	0,06
CIDXPO	0	1	0,28	0,30	0,28	0,30
Apertum	0,06	0,28	1	0,36	0,33	0,36
Noris	0	0,30	0,36	1	0,36	0,40
Paragon	0,06	0,28	0,33	0,36	1	0,36
Excel	0,06	0,30	0,36	0,4	0,36	1

Tabelle 6.5: Ähnlichkeit der Namensräume

für drei der 15 Testszenarien keine Mappingpfade für die Wiederverwendung übrig.

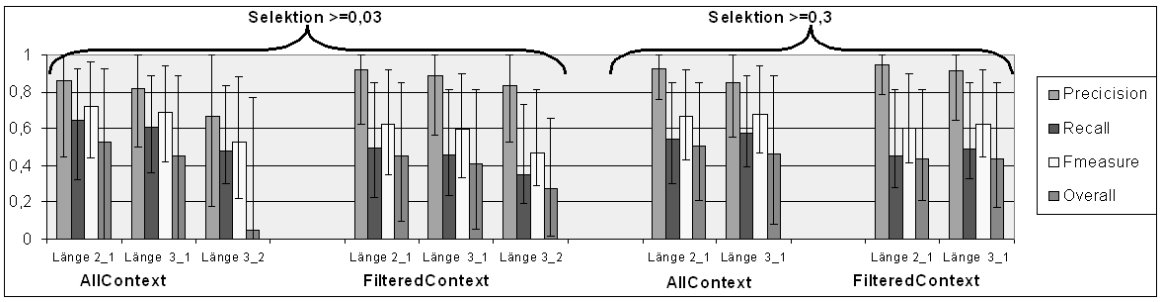


Abbildung 6.9: Ergebniswerte für die Selektion der unvollständigen Mappingpfade auf die Ähnlichkeit der Namensräume

Für diese Testszenarien bringt eine Selektion auf Namensräume einen Erfolg, da die Namensräume im allgemeinen zur Abgrenzung und Zuordnung zu Gültigkeitsbereichen genutzt werden. Sie enthalten also wichtige Informationen. Allerdings sollte man aufpassen, da ein Namensraum nur eine gewisse Aussagekraft über das Schema besitzt.

6.8.3 Ähnlichkeit der Elementnamen

Als letzte Variante zur Selektion in dieser Arbeit wurden die Elementnamen, die in den Schemata vorkommen, genutzt. Es wurde ein Ähnlichkeitswert berechnet, der die Ähnlichkeitswerte für die einzelnen Elementnamen der Schemata kombiniert. Die berechneten Werte für die Schemata sind in Tabelle 6.6 aufgeführt.

	OpenTrans	CIDXPO	Apertum	Noris	Paragon	Excel
OpenTrans	1	0,46	0,58	0,51	0,56	0,48
CIDXPO	0,46	1	0,61	0,67	0,64	0,79
Apertum	0,58	0,61	1	0,73	0,67	0,68
Noris	0,51	0,67	0,73	1	0,64	0,70
Paragon	0,56	0,64	0,67	0,64	1	0,66
Excel	0,48	0,79	0,68	0,70	0,66	1

Tabelle 6.6: Ähnlichkeit der Elementnamen

Es wurden zwei Selektionen auf $\geq 0,6$ und $\geq 0,7$ getestet, bei der noch zehn bzw. nur noch drei der 15 Matchaufgaben erlaubt waren.
Abbildung 6.10 zeigt die erzielten Ergebnisse für alle bzw. die jeweils ausgewählten Mappingpfade.

Abbildung 6.10: Ergebniswerte für die Selektion der unvollständigen Mappingpfade auf die Ähnlichkeit der Elementnamen

Die höchsten Werte - Fmeasure 0,81 und Overall 0,65 - erreicht die Aggregationsstrategie MW für die Matchstrategie AllContext und der Selektion $\geq 0,7$ auf Mappingpfade der Länge 3_2. Letzteres ist überraschend, da für diese Mappingpfade zwei Matchaufgaben gelöst werden müssen. Vermutlich war es den Matchstrategien möglich für die selektierten Mappingaufgaben sehr guten Mappings zu erzeugen, so dass ein qualitativ hochwertiges Mapping von MatchCompose erzeugt werden konnte.
Darüber hinaus ist zu erkennen, dass mit einer weiteren Selektion auch die durchschnittliche Qualität der Mappingpfade steigt. Dabei kommt es zu dem bereits erwähnten Effekt, dass die Mappingmengen der Länge 3_2 durchschnittlich bessere

Ergebnisse erzielen. Auch ist die Varianz der Ergebnisse, die mit den Fehlerbalken dargestellt ist, am geringsten für die Länge 3_2. Wie schon bei der Selektion auf Namensraumähnlichkeit bringt die stärkste Selektion zwar den Vorteil der höchsten Qualität, allerdings auch den Nachteil, dass evtl. nicht mehr alle Tests überhaupt ein Ergebnis erzielen. In diesem Fall gab es bei der Selektion $\geq 0,7$ für einen Test keinen einzigen unvollständigen Mappingpfad, der diesem Kriterium entsprach, und somit konnte auch kein Mapping berechnet werden.

6.8.4 Ausführungszeiten der Ähnlichkeitsberechnung

Bei der Selektion der Mappingpfade sollte sich natürlich der Aufwand, die Ähnlichkeit der Schemata zu berechnen und damit die Komplexität einer Matchaufgabe abzuschätzen, in Grenzen halten. Ist solch eine Berechnung zu aufwendig, kann vielleicht lieber gleich die Matchaufgabe gelöst werden.
Die Berechnung der Ähnlichkeiten von Schemanamen und Namensräumen ist nicht sehr aufwendig, da jeweils nur zwei Zeichenketten verglichen werden. Die Berechnung kann bei der Bestimmung der unvollständigen Mappingpfade nebenbei erfolgen. In der Ausführung werden nur einigen Millisekunden benötigt, weshalb für diese Varianten keine nähere Untersuchung erfolgte.
Im Gegensatz dazu dauert die Ähnlichkeitsberechnung der Elementnamen deutlich länger, da es viel mehr Zeichenketten gibt, die miteinander verglichen werden müssen. Dies entspricht einer Matchaufgabe, die jedoch einfach ist, da keinerlei Struktur, Pfade oder Datentypen betrachtet werden. Zur Eingrenzung des Aufwandes wird beim Import eines Schemas die Ähnlichkeitsberechnung zu jedem anderen bereits vorhandenen Schema ausgeführt und der sich ergebende Wert intern im Repository gespeichert. Bei der Bestimmung der unvollständigen Mappingpfade kann auf diesen Wert zurückgegriffen werden.
Abbildung 6.11 zeigt die Ausführungszeit für die Ähnlichkeitsberechnung von fünf bzw. sechs Schemata. Die Berechnung dauert nur wenige Sekunden und benötigt ungefähr genauso viel Zeit wie das Importieren der Schemata in das Repository. Zu beachten ist, dass die Anzahl der zu berechnenden Ähnlichkeiten quadratisch zu der Anzahl der Schemata wächst. Bei fünf Schemata mussten zehn und bei sechs Schemata schon 15 Ähnlichkeiten berechnet werden. Bei n Schemata sind es nämlich $\frac{(n-1)^2-n-1}{2}$ Werte und ist n=20 so werden schon 190 Werte benötigt. Diese Berechnung würde demzufolge bereits viele Minuten in Anspruch nehmen.

6.9 Verwendung eines Pivotschemas

Da der Rahmen der Arbeit nicht ausreicht um ein Pivotschema durch Schemaintegration zu erzeugen, wird eines der bereits importierten Schemata stellvertretend dafür verwendet. Dabei werden exemplarisch drei Schemata untersucht:

- OpenTrans - das größte Schema

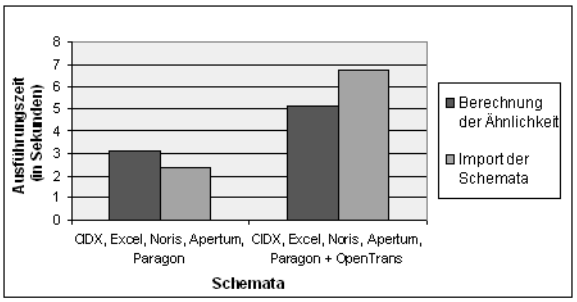


Abbildung 6.11: Benötigte Ausführungszeit für die Berechnung der Ähnlichkeit der Elementnamen bei fünf kleinen Schemata bzw. fünf kleinen + einem großen

- Apertum - ein mittelgroßes Schema
- CIDXPO - das kleinste Schema

Da das Pivotschema jeweils eines der Testschemata ist, werden nur jeweils die zehn Testszenarien gelöst, bei denen die gegebene Matchaufgabe nicht aus dem Pivotschema besteht. In den anderen Fällen kann direkt das Mapping *Schema* ↔ *Pivotschema* geladen werden, da es die gesuchte Lösung darstellt.

Zuerst soll die Verwendung eines Pivotschemas mit vollständigen Mappingpfaden untersucht werden. Dabei entspricht, wie in Abschnitt 5.2.5 erläutert, der Mappingpfad Länge 2 dem Matchen direkt über das Pivotschema. Es werden also nur Mappings verwendet, die zwischen einem Schema und dem Pivotschema vorhanden sind. Hierfür gab es jeweils immer nur einen direkten Weg. Abbildung 6.12 zeigt die durchschnittlichen Ergebnisse für die vollständige Wiederverwendung der Länge 2 sowohl ohne und mit Pivotschema. Durch die Verwendung eines Pivotschemas findet eine Filterung der

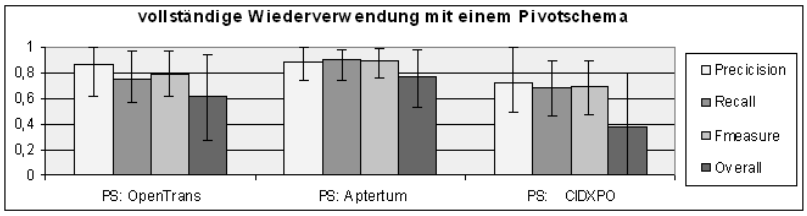


Abbildung 6.12: Mapping-Wiederverwendung über verschiedene Pivotschemata mit vollständigen Mappingpfaden der Länge 2

vollständigen Mappingpfade der Länge 2 statt. Es wird nur noch der eine Pfad berechnet, der über das jeweilige Pivotschema führt. Das Pivotschema Apertum erzielt mit Fmeasure von 0,89 und einem Overall von 0,77 das beste Ergebnis. Mit dieser Strategie werden durchschnittlich 90% aller gesuchten Korrespondenzen gefunden und

88% aller erzeugten Korrespondenzen sind korrekt. Dies liegt daran, dass das Schema Apertum groß genug ist, um die wichtigen Elemente abzudecken, jedoch nur wenige Angriffspunkte bietet um falsche Korrespondenzen zu berechne. Mit OpenTrans als Pivotschema wurden zwar ungefähr ebenso viele gesuchte Korrespondenzen gefunden wie bei Apertum, jedoch auch mehr falsche. Das Schema OpenTrans ist sehr groß und somit sind auch mehr Möglichkeiten gegeben, falsche Korrespondenzen zu erzeugen. CIDXPO wiederum ist das kleinste Schema und deckt weniger Elemente ab. Deshalb werden auch weniger der gesuchten Korrespondenzen abgebildet und die Gesamtqualität der erzeugten Mappings darüber ist geringer.
Außerdem lässt sich an den Fehlerleisten feststellen, dass durch die Verwendung eines großen bzw. mittelgroßen Pivotschemas die Varianz der Ergebnisse kleiner ist. Dies bedeutet, dass die Verwendung des Pivotschemas für alle getesteten Szenarien (sehr) gute Ergebnisse mit sich bringt. Für ein kleines Pivotschema gilt dies jedoch nicht.

Nun wird die unvollständige Mapping-Wiederverwendung über ein Pivotschema betrachtet. Für die Länge 2_1 gab es für jedes Testszenario zwei Mappingpfade unter der Annahme, dass jeweils eines der beiden Mappings zum Pivotschema bereits existiere und das andere noch erzeugt werden muss. Die fehlenden Mappings wurden - wie schon in den vorherigen Abschnitten - mit den Matchstrategien AllContext und FilteredContext erzeugt.
In Abbildung 6.13 sind die Durchschnittswerte, das Minimum und das Maximum aller erreichten Ergebnisse dargestellt. Zu erkennen ist, dass wiederum Apertum als Pivotschema die höchsten Ergebnisse mit einem Fmeasure von 0,78 und einem Overall von 0,56 für AllContext erreicht hat. Die Varianz der Ergebnisse mit der Verwendung eines Pivotschemas ist geringer als dies bei der unvollständigen, unkombinierten Mapping-Wiederverwendung der Fall ist. Insgesamt eignet sich also Apertum als mittelgroßes Schema am besten für alle Testszenarien, da mit diesem Schema für alle Testszenarien gute bis sehr gute Ergebnisse erzielt werden. Außerdem werden für AllContext fast ähnlich hohe Ergebnisse erzielt wie für FilteredContext, während ohne Wiederverwendung dort ein deutlicher Unterschied ist.

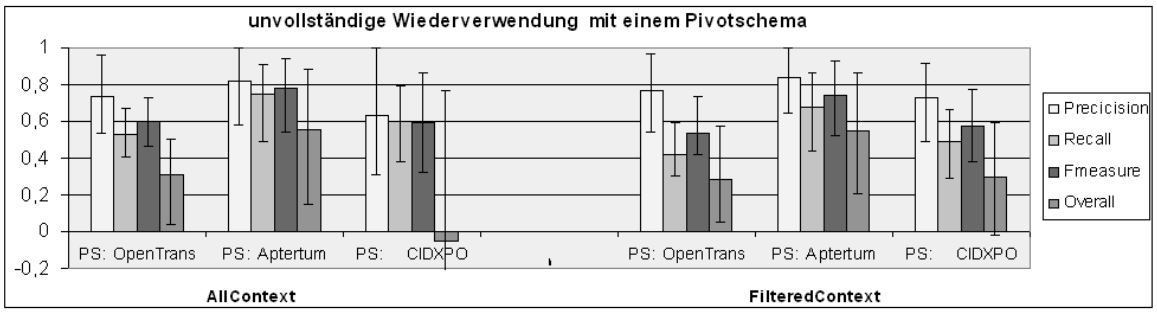


Abbildung 6.13: Mapping-Wiederverwendung über verschiedene Pivotschemata mit unvollständigen Mappingpfaden der Länge 2_1 und der Matchstrategie AllContext bzw. FilteredContext

6.10 Qualitäts- und Zeitvergleich

Um die Übersichtlichkeit zu wahren, wird für den Qualitätsvergleich für jede Wiederverwendungsvariante nur der beste Fall genommen. Die Ergebnisse des kombinierten Wertes Fmeasure sind in Abbildung 6.14 dargestellt. Bis auf zwei Fälle - *unvollständige, unkombinierte Mapping-Wiederverwendung* und deren Selektion auf Schemanamensähnlichkeit - erzielt die Mapping-Wiederverwendung gleich gute und sogar deutlich bessere Ergebnisse als das Matchen ohne Wiederverwendung. Eine Stärke der Wiederverwendung ist dabei, dass Elemente aufgrund der Transitivität als ähnlich gelten - unabhängig davon, ob sie beispielsweise ähnliche Namen oder Datentypen aufweisen. Die angewendeten Matchstrategien finden jedoch Korrespondenzen nicht, wenn die Element zu unterschiedlich in den untersuchten Kriterien sind.
Es ist zu erkennen, dass die kombinierte Wiederverwendung sowohl für unvollständige als auch vollständige Mappingpfade einen höheren Fmeasure erzielt und auch eine geringere Varianz der Ergebnisse beinhaltet. Dies ist darauf zurückzuführen, dass die falschen Korrespondenzen herausgefiltert werden. So kann in der Kombinationsmenge durchaus ein Mapping von geringer Qualität sein, ohne dass das Gesamtergebnis deswegen eine geringe Qualität erhält. Die höchste Qualität und die geringste Variation erreicht das Pivotschema Apertum, wenn für die Matchaufgaben die Mappings bereits vorhanden sind. Da beide Mappings in diesem Fall vom Nutzer bestätigt sind, besitzen sie eine hohe Qualität und durch MatchCompose wird ein Mapping erzeugt, was nur wenige falsche Korrespondenzen und die meisten korrekten enthält.

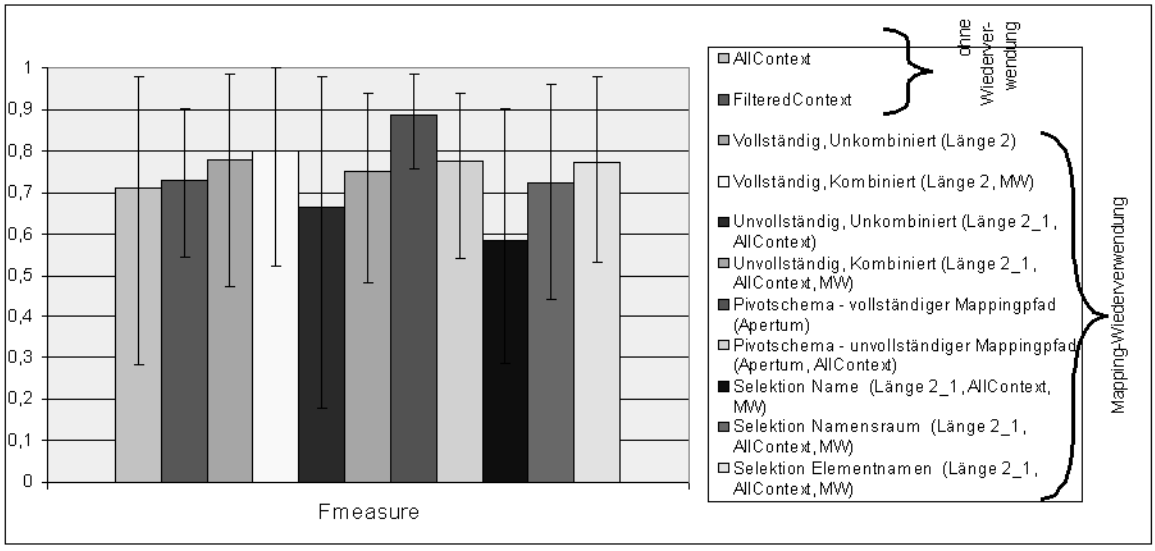


Abbildung 6.14: Die Ausführungszeiten für das Matchen ohne Wiederverwendung und die unterschiedlichen Varianten der Mapping-Wiederverwendung

Nachdem die Qualität der erzeugten Mappings angeschaut wurde, wird im Folgen-

den die Ausführungszeit der unterschiedlichen Wiederverwendungsstrategien[9] betrachtet. Diese ist für den Anwender insofern interessant, als dass dieser Ergebnisse in einer angemessenen Zeit erwartet oder zumindestens vor der Ausführung wissen will, wie lange die Berechnung dauern wird.
Abbildung 6.15 zeigt die durchschnittliche Ausführungszeit der zuvor untersuchten Strategien. Am schnellsten ist die vollständige Mapping-Wiederverwendung für einen einzelnen Mappingpfad mit nur 0,003 Millisekunden. Dabei wird jedoch angenommen, dass alle benötigten Mappings bereits im Speicher vorliegen und auf diesen MatchCompose ausgeführt wird. Die unvollständige Mapping-Wiederverwendung benötigt länger, da neben der MatchCompose-Operation zuvor noch das eine oder die beiden fehlenden Mappings berechnet werden müssen. Dieser Vorgang dauert weniger als drei Sekunden und dann liegt auch hier das Ergebnis vor. Am längsten dauert die unvollständige, kombinierte Mapping-Wiederverwendung, da hier alle vorhandenen Mappingpfade zuerst berechnet werden müssen und dabei ganz unterschiedliche Matchaufgaben gelöst werden müssen. Die Aggregation der Mappings dauert wiederum nur wenige Millisekunden. Durch eine Selektion der Mappingpfade wird die benötigte Zeit im Durchschnitt halbiert, da nur circa die Hälfte der Pfade ausgewählt und verwendet werden.
Die Verwendung eine Pivotschemas benötigt generell wenig Zeit, da für eine Matchaufgabe genau ein Matchpfad berechnet werden muss. Je nachdem ob die benötigten Mappings vorliegen oder nicht, ist dies ein vollständiger oder unvollständiger Mappingpfad.

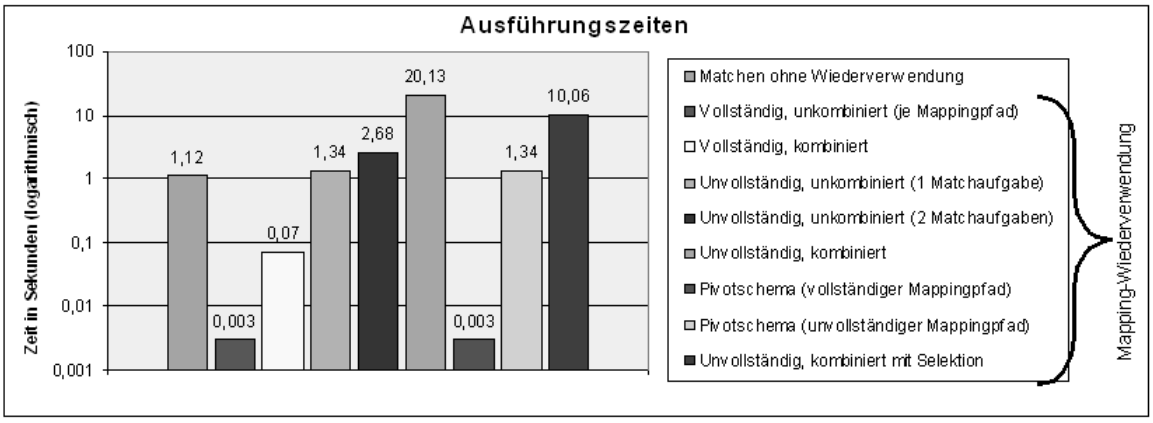


Abbildung 6.15: Die Ausführungszeiten für das Matchen ohne Wiederverwendung und die unterschiedlichen Varianten der Mapping-Wiederverwendung

[9]Implementation erfolgte in Java 1.4.2 (http://www.sun.com/download/index.jsp) und als Datenbank wurde MySQL 4.0.18 (http://dev.mysql.com/downloads/mysql/4.0.html) verwendet.

6.11 Diskussion der Ergebnisse

Die Evaluation erfolgte auf 15 Testszenarien und für unterschiedliche Varianten der Mapping-Wiederverwendung. Bei der Ausführung der Tests wurden bestimmte Datenbankinhalte bzw. die Matchprobleme simuliert.
In den untersuchten Testszenarien erzielte die vollständige Mapping-Wiederwendung bessere Ergebnisse als das Matchen ohne Wiederverwendung und benötigt dafür nur wenige Millisekunden. Die kürzesten Mappingpfade führen dabei zu Mappings mit höherer Qualität als dies lange Pfade tun.
Außerdem konnte gezeigt werden, das es bei der unvollständige Mapping-Wiederverwendung durchaus Sinn macht ein gegebenes Matchproblem durch ein neues - in einem unvollständigen Mappingpfad - zu ersetzen. Die erzielten Resultate waren zwar nicht so hoch wie bei der vollständigen Mapping-Wiederverwendung, aber immer noch gut. In der Anwendung ist diese Strategie besonders sinnvoll, wenn mit unterschiedlichen Schemaversionen umgegangen werden muss und nicht jedes Mal die Mappings ganz neu erstellt werden sollen.
Bessere Ergebnisse lassen sich bei der unvollständigen Mapping-Wiederverwendung erzielen, wenn nur Mappingpfade verwendet werden, bei denen die Schemata eine gewisse Ähnlichkeit besitzen. Dabei ist es entscheidend, woran diese Ähnlichkeit gemessen wird und welchen Zeitaufwand man dafür investieren will. In den untersuchten Testszenarien brachte die Selektion auf ähnliche Schemanamen keinen Vorteil. Dies mag in Anwendungen eventuell anders aussehen. Die Selektion auf möglichst ähnliche Elementnamen hat eine starke Qualitätsverbesserung mit sich gebracht, aber bedeutete auch viele - wenn auch einmalige - Berechnungen. Die Ähnlichkeit der Namensräumen ist schnell berechnet und die Selektion der Mappingpfade auf dieses Kriterium führte auch zu einer erhöhten Qualität. Nachteil ist bei einer Selektion, dass eventuell kein Mappingpfad den gewünschten Ähnlichkeitswert besitzt. In diesem Fall muss der Schwellwert verringert werden, um dann die nächstbesten zu erhalten.
Bei der Mapping-Wiederverwendung werden falsche Korrespondenzen berechnet, wenn die Voraussetzung der Transitivität von Ähnlichkeit nicht durchgängig gilt. Durch die kombinierte Mapping-Wiederverwendung kann dieser Effekt jedoch abgemildert werden, da die falsche Korrespondenzen im Gegensatz zu den korrekten herausgefiltert werden. Je nach Kombinationsstrategie können unterschiedliche Schwerpunkte gesetzt werden. Durch die Auswahl des Ähnlichkeitsminimums werden zwar nur wenige Korrespondenzen berechnet, aber diese sind fast alle korrekt. Wird das Maximum ausgewählt, so werden sehr viele von den gesuchten Korrespondenzen geliefert. Nachteil sind die relativ vielen falschen Korrespondenzen, die wiederum erstmal herausgefunden und entfernt werden müssen. Der Mittelwert der Ähnlichkeitswert gleicht die Vor- und Nachteile dieser beiden Strategien aus. Es werden nicht so restriktiv Korrespondenzen verworden, aber auch nicht so optimistisch jede gleich aufgenommen. Im allgemeinen ist es einfacher, aus der vorgeschlagenen Korrespondenzenmenge die falschen herauszufinden, als aus allen möglichen Korrespondenzen die fehlenden Korrespondenzen zu ermitteln. Daher empfiehlt es sich, die Aggregationsstrategien Mittelwert bzw. Maximum zu verwenden.

Eine Variante der Mapping-Wiederverwendung ist das Nutzen eines Pivotschemas. Kurzfristig bedeutet es zwar zusätzlichen Aufwand, da Mapping zu diesem zentralen Schema erstellt werden müssen, obwohl dies nicht Teil der gegebenen Matchaufgaben ist. Mittelfristig bringt es Einsparungen bei der Berechnung von Mappings.Bei diesem Konzept ist natürlich die Beschaffenheit des Pivotschemas sehr wichtig. Während für diese Arbeit aus Aufwandsgründen vorhandene Schemata als Pivotschema verwendet wurden, sollte in der Anwendung ein Pivotschema für die jeweilige Domäne erstellt werden durch z.B. die Integration der Schemata der Matchaufgaben. Dabei sollten alle Elemente aller Schemata abgedeckt sein, da für jedes fehlende Element auch keine Korrespondenz erstellt werden kann. Anderseits sollten auch möglichst keine Redundanzen vorhanden sein, da dies zu fehlerhaften Korrespondenzen führen kann und auf jeden Fall die Erstellung der Mappings zu dem Pivotschema erschwert. Aus diesen Gründen wurden mit dem mittelgroßen Pivotschema auch die qualitativ hochwertigsten Mappings erzeugt. Dieses führteneben der Verringerung des Arbeitsaufwand sogar noch zu einer Steigerung der Qualität der erzeugten Mappings.
Generell sei noch erwähnt, dass weniger die Anzahl der vorhandenen Mappings entscheidend ist, obwohl mit steigender Anzahl mehr Mappingpfade berechnet werden können, sondern die Qualität. Die Mappings, die mit MatchCompose aus Mappingpfaden erzeugt werden, hängen nämlich von der Qualität der Teilmappings ab. Besitzen die Ausgangsmappings eine geringe Qualität, dann wird das Ergebnis auch keine hohe Qualität besitzen. Dabei reicht schon ein Mapping im Mappingpfad mit geringer Qualität aus, um das Ergebnis stark an Qualität verlieren zu lassen. Daher sollte man bewusst eine Auswahl aller möglichen Mappingpfade treffen, um ein besseres Ergebnis zu erzielen.

7 Grafische Benutzeroberfläche

7.1 Überblick

In dem Prototypen COMA++ wurde eine Grafische Benutzeroberfläche (GUI, Graphical UserInterface) implementiert, mit der die Funktionalitäten des Prototyps benutzt werden können. Die Programmierung erfolgte in Java Swing. Als Programmierumgebung wurde die Eclipse Platform 3.0.1[10] der Apache Software Foundation[11] genutzt. Alle Komponenten von COMA++ - Schema Pool, Match Customizer, Mapping Pool, Execution Engine und Repository - werden verwaltet und mit Hilfe der GUI angesprochen. Der GUI kommt in der Gesamtarchitektur, siehe Abbildung 3.1, eine besondere Rolle zu: Sie visualisiert die Schemata und Mappings und ermöglicht so dem Nutzer eine interaktive Benutzung, wie z.B. das Erstellen von Mappings.

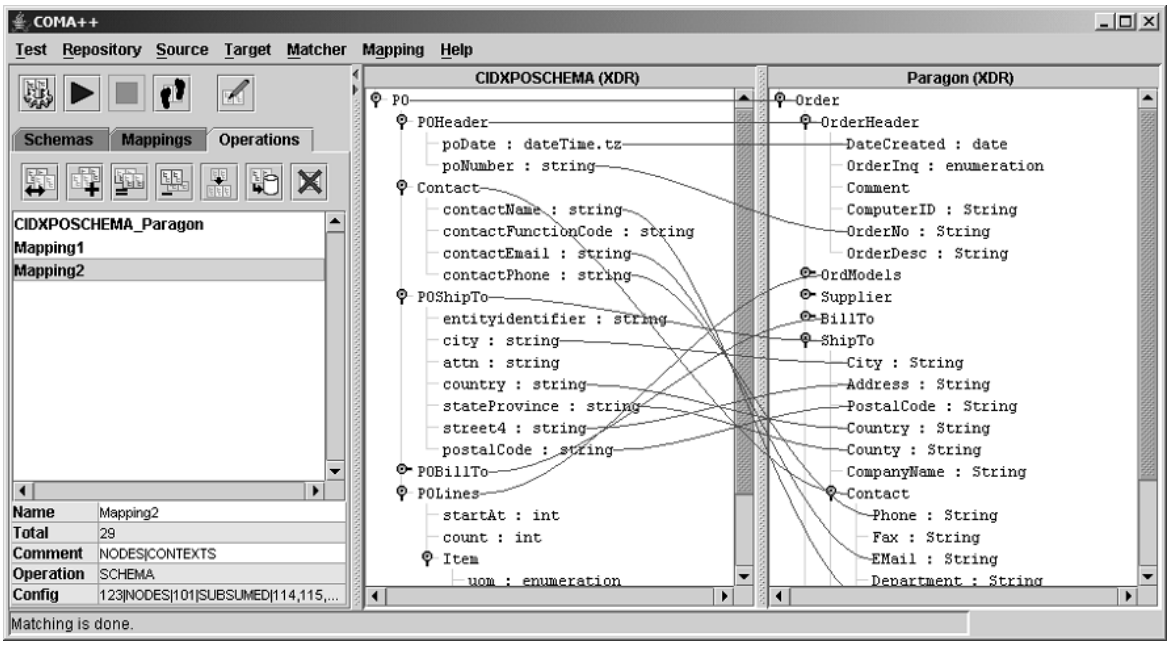


Abbildung 7.1: Die GUI des Prototyps COMA++

Die Funktionalität von COMA++ kann über Menüs, Button und Dialoge angespro-

[10]Eclipse Platform http://www.eclipse.org/platform, Release Build 3.0.1 vom 16. September 2004
[11]Apache Software Foundation http://www.apache.org/

chen werden. Die wichtigsten werden im Anhang 8 näher erläutert. Nachfolgend sollen einige Nutzungsmöglichkeiten der GUI diskutiert werden. Es wird gezeigt, wie man mit Hilfe der GUI Matcher und die Matchstrategie konfiguriert, wie das Matchen erfolgt, wie mit Hilfe der Wiederverwendung Mappings erzeugt werden könne und welche Möglichkeiten zum Bearbeiten der Mappings bestehen.

7.2 Matcherkonfiguration

Die Konfiguration jedes Matchers aus der Matcherbibliothek, siehe Abschnitt 3.5, kann jederzeit geändert werden. Dadurch ist das Verwenden unterschiedliche Einstellungen möglich, um eine optimale Konfiguration für ein Matchproblem zu finden. Es ist auch möglich, mit Hilfe der Auflösungs-, Matcher- und Kombinationsbibliothek neue Matcher zu definieren und diese der Matcherbibliothek hinzuzufügen.

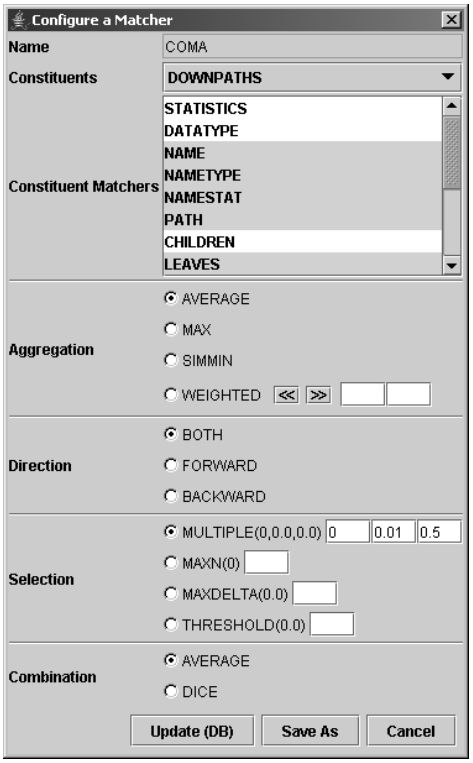


Abbildung 7.2: Der Konfigurationsdialog für Matcher - hier mit den Standardeinstellungen des Matchers COMA

Jeder Matcher besitzt einen Namen, der ganz oben im Konfigurationsdialog, siehe Abbildung 7.2, angegeben ist. Wird ein bestehender Matcher konfiguriert, ist dieser Name unveränderlich, bei einem neu erstellten Matcher kann dieser vom Benutzer frei gewählt werden.

In Abschnitt 3.3 wurde die Auflösungsbibliothek vorgestellt, mit der die Bestandteile (Constituents) zum Matchen bestimmt wurden. Für einen Matcher muss eine Auflösungsstrategie ausgewählt werden, z.B. SelfPath bei NamePath.
Darüber hinaus müssen die Teilmatcher (Constituent matchers, Abschnitt 3.4) festgelegt werden, mit denen die Ähnlichkeit der Bestandteile berechnet werden soll. Der Benutzer bestimmt selber, ob ein Teilmatcher ausreichend ist oder ob mehrere Teilmatcher ausgeführt werden sollen.
Schließlich muss noch festgelegt werden, durch welche Aggregationsstrategie die Ähnlichkeitswerte kombiniert sollen. Durch die Festlegung der Richtung (Direction) und der Selektionsstrategie (Selection) werden die Matchkandidaten in eine Rangfolge gebracht und eine Auswahl wird getroffen. Die Kombinationsstrategie (Combination) wird für die Berechnung eines kombinierten Wertes verwendet, wobei dieser Schritt nicht für jeden Matcher ausgeführt wird. Sind die Einstellungen eines existierenden Matchers geändert worden, so kann die neue Konfiguration in das Repository übernommen werden oder ein neuer Matcher mit diesen Einstellungen gespeichert werden.

7.3 Konfiguration der Matchstrategie

Der Dialog in Abbildung 7.3 dient zur Konfiguration der Matchstrategie, die auf ein Matchproblem angewendet werden soll. Zum einen muss eine der drei Matchstrategien ausgewählt werden. AllContext und FilteredContext wurden bereits in Abschnitt 3.6 vorgestellt. Als dritte Matchstrategie wurde Fragment implementiert, die das Matchen auf Schemafragmenten ermöglicht.

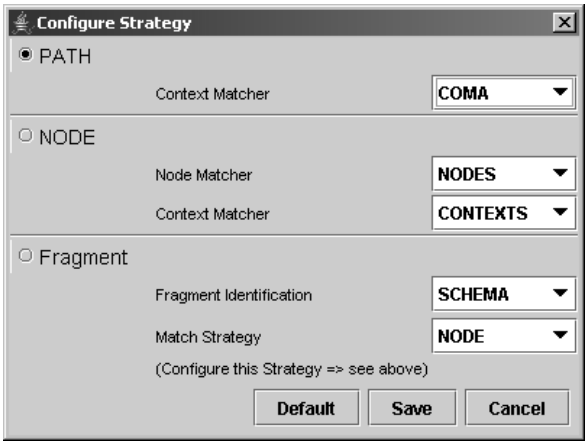


Abbildung 7.3: Der Dialog zur Konfiguration der Matchstrategie

Zum anderen muss für jede Matchstrategie eine Konfiguration erfolgen. Für AllContext ist eine Matcherkombination zu spezifizieren, mit der die Elemente und ihre Kontexte zweier Schemata gematcht werden. Die Standardeinstellung wurde unter dem

Namen COMA abgelegt. Für FilteredContext müssen zwei Matcherkombinationen bestimmt werden, die eine zum Matchen der Elemente und die andere für die Kontexte. Hierfür sind die Standardeinstellungen unter den Namen Nodes und Contexts abgelegt. Fragment bestimmt zuerst die Schemafragment, für die Korrespondenzen berechnet werden sollen. Dazu muss die Strategie zur Fragmentidentifikation bestimmt werden. Diese Fragmente werden mit der ausgewählten Matchstrategie gematcht. Möglich sind AllContext oder FilteredContext, deren Konfiguration der Einstellung im oberen Teil des Dialoges entspricht.

7.4 Matchen

Um für ein Matchproblem ein Mapping zu erzeugen, müssen zuerst die Schemata ins Repository importiert werden. Dann müssen die entsprechenden Schemata aus dem Repository in die GUI als Ausgangs- und Zielschema geladen werden. Dort werden sie in Form eine Baumes angezeigt. Als nächstes können die Matchstrategie und deren Konfiguration verändert werden. Gegebenenfalls können auch Matcher konfiguriert oder erstellt werden, um sie für diese Matchaufgabe zu nutzen.
Durch die GUI lässt sich dann der Matchprozess starten. Die Ausführung kann automatisch gesamt ablaufen. Nach Ende der Berechnung wird das erzeugte Mapping in die GUI geladen und die Korrespondenzen angezeigt. Außerdem ist ein schrittweise Ablauf möglich, bei dem der Benutzer interaktiv in den Matchprozess eingreifen kann. Das Ergebnis des Schema-Matchings für die Schemata *Excel* und *Noris* und die Matchstrategie AllContext mit der Matcherkombination COMA ist in Abbildung 7.4 zu sehen. Es werden alle Korrespondenzen angezeigt, die erzeugt wurden. Es ist möglich, für einen einzelnen Knoten alle vorhandenen Korrespondenzen mit den Ähnlichkeitswerten anzeigen zu lassen, indem man den Knoten im Graph selektiert.

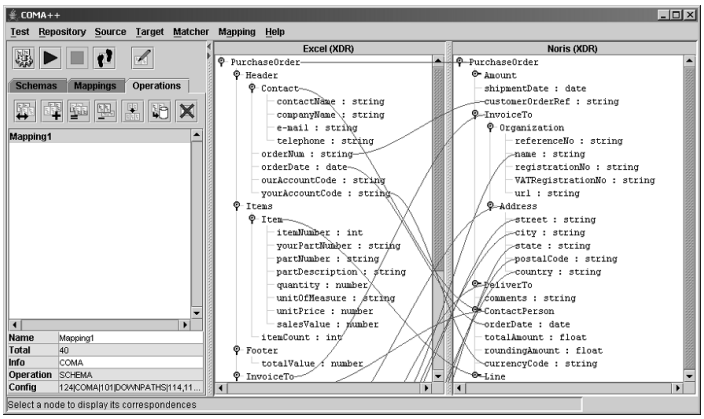


Abbildung 7.4: Ein mit der Matchstrategie *AllContext* erzeugtes Mapping *Excel* ↔ *Noris*

7.5 Dialoggeführte Wiederverwendung

Die GUI ermöglicht eine dialoggeführte Wiederverwendung. Dabei wird dem Benutzer aufeinander folgend - soweit vorhanden - zur Auswahl angeboten:

- direkte Mapping-Wiederverwendung
- unvollständige Mappingpfade Länge 2, 3 und 4
- unvollständigen Mappingpfade Länge 2_1, 3_1 und 3_2

Mit direkter Mapping-Wiederverwendung sind alle Mappings gemeint, die zwischen den Schemata des gegebenen Matchproblems $S1 - S2$ im Repository existieren. Dazu werden alle Mappings M im Repository gesucht, wobei $M : S1 \leftrightarrow S2$ oder $M : S2 \leftrightarrow S1$ gilt. In Abbildung 7.5 ist ein Beispieldialog zu sehen. Falls der Benutzer ein Mapping auswählt, wird dieses geladen, und er kann es in der GUI ansehen und bearbeiten. Er kann auch zum nächsten Schritt übergehen, wenn er z. B. diese *direkten Mappings* selber erzeugt hat und eine andere Methode ausprobieren will.

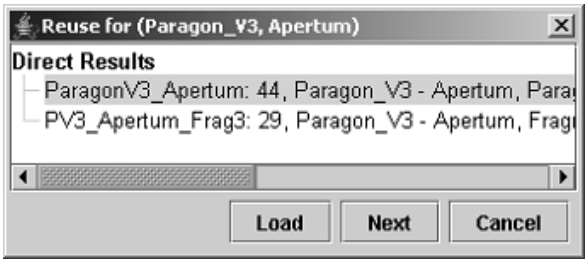


Abbildung 7.5: Screenshot eines Dialogs, in dem für das Matchproblem *Paragon-Apertum* zwei direkte Mappings angezeigt werden

Im zweiten Schritt werden alle berechneten vollständigen Mappingpfade der Länge 2 bis 4 dem Benutzer angezeigt, wofür in Abbildung 7.6 ein Beispiel zu sehen ist. Wählt der Benutzer einen Mappingpfad aus, wird die vollständige, unkombinierte Mapping-Wiederverwendung angewendet und das erzeugte Mapping in die GUI geladen, in der er es bearbeiten kann. Wählt er mehrere Pfade aus, kann der Benutzer entscheiden, ob er die vollständige, unkombinierte Mapping-Wiederverwendung oder die vollständige, kombinierte Mapping-Wiederverwendung ausführen will. In allen Fällen werden die Ergebnismappings in die GUI geladen.

Der Benutzer kann auch diesen Schritt übergehen, wenn er z. B. diese Mappingpfade bereits berechnet hat.
Im dritten Schritt werden die unvollständigen Mappingpfade der Länge 2_1, 3_1 und 3_2 berechnet und dem Benutzer angezeigt. In Abbildung 7.7 ist ein Beispieldialog abgebildet.

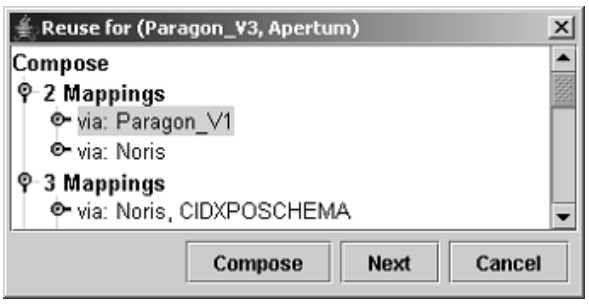


Abbildung 7.6: Screenshot eines Dialogs, in dem für das Matchproblem *Paragon-Apertum* vollständige Mappingpfade angezeigt werden

Wählt der Benutzer einen Mappingpfad aus, so wird die unvollständige, unkombinierte Mapping-Wiederverwendung ausgeführt und das Ergebnismapping in die GUI geladen. Sind mehrere Pfade ausgewählt worden, so kann der Benutzer sich für die unvollständige, unkombinierte Mapping-Wiederverwendung entscheiden, bei der alle erzeugten Mappings an die GUI weitergereicht werden. Er kann jedoch auch die unvollständige, kombinierte Mapping-Wiederverwendung anwenden und erhält so nur ein Mapping.

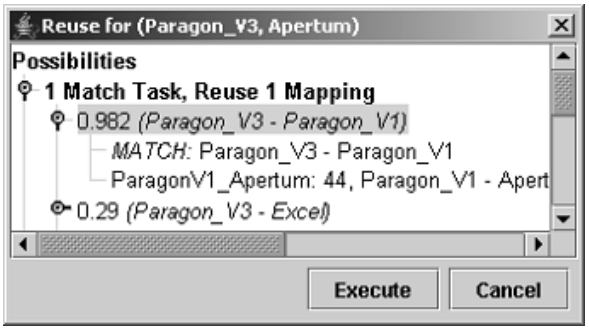


Abbildung 7.7: Screenshot eines Dialogs, in dem für das Matchproblem *Paragon-Apertum* unvollständige Mappingpfade angezeigt werden

7.6 Mappingmanipulation

7.6.1 Editieren

Ein Mapping, welches sich in der GUI befindet, kann durch den Benutzer editiert werden. Dabei spielt es keine Rolle, ob das Mapping aus dem Repository bzw. einer Datei geladen oder durch einen Matchprozess erzeugt wurde.
Zuerst muss der Editiermodus aktiviert werden, in welchem die Änderungen von Korrespondenzen möglich ist. Ist der Editiermodus aktiv, dann zeigt der Hintergrund des MappingSplitpane eine hellblaue statt weiße Farbe. Soll eine Korrespondenz hinzugefügt werden, die es noch nicht gibt, so muss erst der eine Knoten in dem einen

Schema und dann der zweite Knoten in dem anderen Schema angeklickt werden. Die Korrespondenz wird dann automatisch hinzugefügt und besitzt den Ähnlichkeitswert 1.0, was der höchstmögliche Wert ist. Existiert zwischen 2 Elementen eine Korrespondenz, die entfernt werden soll, muss ebenfalls zuerst auf den einen Knoten und dann auf den anderen geklickt werden. Daraufhin wird die Korrespondenz aus dem Mapping entfernt und die Anzeige aktualisiert.
Sind alle gewünschten Änderungen gemacht, so muss der Editiermodus ausgeschaltet werden, damit durch das zufällige Umherklicken im Graphen keine falschen Korrespondenzen entstehen oder korrekte gelöscht werden. Der Hintergrund der MappingSplitpane wird wieder weiß, sobald der Modus deaktiviert wurde.

In Abbildung 7.8 wurde die Korrespondenz $Footer.totalValue \cong totalAmount$ zu dem Mapping aus Abbildung 7.4 hinzugefügt. Der Ähnlichkeitswert der neu erzeugten Korrespondenz ist 1.0.

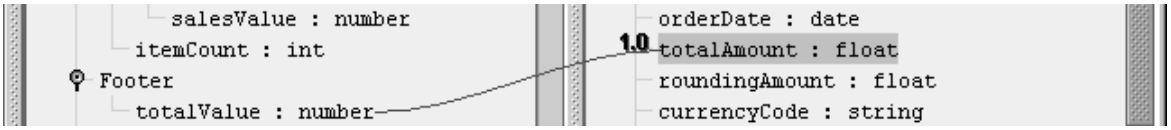


Abbildung 7.8: Zu dem Mapping aus Abbildung 7.4 wurde eine Korrespondenz hinzugefügt

7.6.2 Operationen

Befinden sich mehrere Mappings in der GUI, ob durch einen Matchprozess erzeugt oder aus dem Repository bzw. Dateien geladen, lassen sich verschiedene Operationen auf diesen ausführen. Abbildung 7.9 zeigt die Buttons, über welche die Mappingoperationen ausgeführt werden.

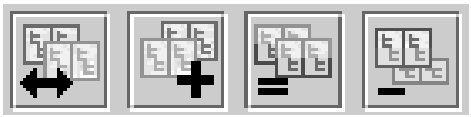


Abbildung 7.9: Mappingoperationen (von links nach rechts): Compare, Merge, Intersect, Diff

- **Compare**: Mappings können miteinander verglichen werden, indem man ein Mapping als Zielmapping bestimmt und ein weiteres Mapping, was mit diesem verglichen werden soll. Die Ergebniswerte sind dann Maße, wie z. B. Precision und Overall, welche in Abschnitt 6.2 erklärt wurden. In Abbildung 7.10 ist ein Vergleich von dem Mapping *Mapping1* zum Zielmapping *Excel_ Noris* zu sehen.

- **Merge**: Es ist möglich, mehrere Mappings zu bestimmen, die dann zu einem neuen Mapping vereinigt werden. Das Ergebnismapping wird berechnet und automatisch angezeigt.
- **Intersect**: Durch die Funktion *Intersect* werden Korrespondenzen bestimmt, die sich in jedem Mapping einer ausgewählten Mappingmenge befinden. Eine Korrespondenz ist damit nicht im Ergebnismapping, sobald sie in einem Mapping nicht vorhanden ist.
- **Diff**: Es kann die Differenz von Mappings gebildet werden, das heißt, dass die Korrespondenzen vom einem Mapping entfernt werden, wenn sie in dem anderen Mapping vorhanden sind. Das Mapping mit den übrig gebliebenen Korrespondenzen wird wieder automatisch in der GUI angezeigt.

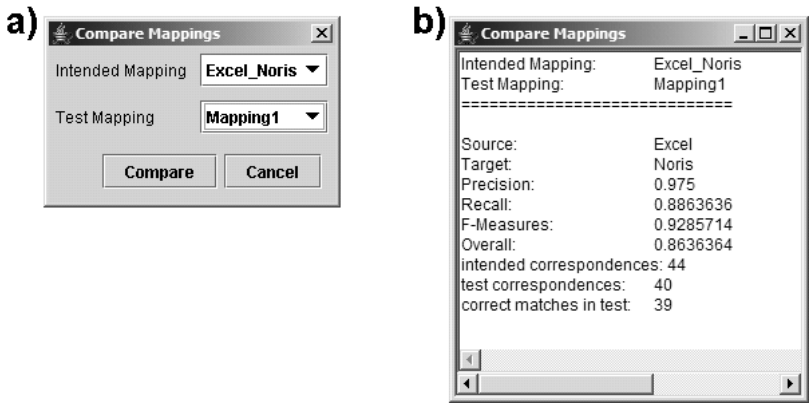


Abbildung 7.10: a)Dialog zum Mappingvergleich b) Ergebniswerte des Vergleichs *Mapping1* und *Excel_Noris*

8 Zusammenfassung

Diese Arbeit hat einen Überblick über bisherige Ansätze der Wiederverwendung gegeben. Da die Mapping-Wiederverwendung ein lohnenswerter Ansatz ist und dort noch viel Potential identifiziert wurde, wurden auf der Basis bereits implementierter Ansätze im Prototyp COMA weitere entwickelt. Zum einen wurden Mappingpfade berechnet, die mehr als zwei Mappings beinhalten. Zum anderen wurden unvollständige Mappingpfade betrachtet, um auch Mappings wiederzuverwenden, wenn diese keine vollständigen Mappingpfade bilden. Die fehlenden Mappings werden durch das Ausführen ein Matchprozesses erzeugt. Durch eine Selektion der Mappingpfade wird die Qualität der Ergebnisse erhöht. Sowohl für vollständige als auch unvollständige Mappingpfade wurde die Kombination von Mappings beschrieben, die zu einer Qualitätsverbesserung führt. Zusätzlich wurde die Verwendung eines Pivotschemas dargelegt, was zu Einsparungen bei den Berechnungen führt.Ein Testen verschiedener Strategien und darüber hinaus Nutzerinteraktionen werden durch die implementierte GUI ermöglicht. Der Anwender kann damit sowohl flexibel die Matchstrategie und die Matcher handhaben, als auch erzeugte Mappings weiter bearbeiten.

Durch Tests konnte gezeigt werden, dass die Algorithmen korrekt arbeiten und für das Lösen von Matchaufgaben geeignet sind. Die Evaluation der implementierten Strategien zur Mapping-Wiederverwendung zeigt, dass die Wiederverwendung Vorteile gegenüber dem wiederverwendungslosen Schema-Matching bietet. Einerseits wird mit der Mapping-Wiederverwendung besonders für vollständige Mappingpfade und die kombinierten Mappings eine hohe Qualität erreicht, die von Matchern ohne Wiederverwendung nicht erzielt wird. Andererseits ist die Ausführungszeit der MatchCompose sehr klein und somit werden schnell Ergebnisse erzeugt. Die Ausführungszeit für die unvollständige Mapping-Wiederverwendung liegt da höher, da noch ein oder mehrere Matchaufgaben zu lösen sind In der Praxis hängt die Art der Mapping-Wiederverwendung davon ab, welche Mappings bereits in der Datenbank vorhanden sind und ob mit diesen vollständige Mappingpfade gebildet werden können. Die Strategie der Verwendung eines (mittelgroßen) Pivotschemas lohnt sich nach kurzer Zeit, da sowohl Mappings mit sehr hoher Qualität erzeugt werden können und dabei jedoch der Aufwand gering ist. Dabei ist die Größe des Pivotschemas zu beachten, denn es sollte möglich viele Elemente abdecken ohne selber zu komplex zu werden.

Die Weiterentwicklung der Strategien zur Mapping-Wiederverwendung kann in mehrere Richtungen erfolgen. Zum einen kann die Selektion von Matchaufgaben der unvollständigen Mappingpfade durch weitere Kriterien ermittelt werden. Zum anderen

bietet sich eine weiterführende Untersuchung zur Verwendung von Pivotschemata an. Interessant wären die Ergebnisse, wenn das Pivotschema durch Schemaintegration erstellt und an neue Matchaufgaben angepasst wird. Auch sollte der Prototyp COMA++ mit den gewonnenen Erkenntnissen weiterentwickelt werden, um so auch Benutzern die Möglichkeit zur Anwendung zu geben.

Literaturverzeichnis

[AB01] S. Alagic and P.A. Bernstein. A model theory for generic schema management. *Proc. DBPL*, 2001.

[ADMR05] D. Aumueller, H.H. Do, S. Massmann, and E. Rahm. Schema and Ontology Matching with COMA++. *Akzeptiert als Demo Proposal: ACM SIGMOD/PODS Conference*, 2005.

[AH00] A.Bauer and H.Guenzel. *Data Warehouse System - Architektur, Entwicklung, Anwendung.* dpunkt, 2000.

[Bax03] A. D. Baxevanis. The molecular biology database collection: 2003 update. *Nucl Acids Res. (NAR)*, 31:1–12, 2003.

[BC86] J. Biskup and B. Convent. A formal view integration method. *ACM SIGMOD Record 15 (2)*, pages 398 – 407, 1986.

[BLN86a] C. Batini, M. Lenzerini, and S. B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Comput. Surv.*, 18(4):323–364, 1986.

[BLN86b] C. Batini, M. Lenzerini, and S.B. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys 18(4)*, pages 323–364, 1986.

[BM01] J. Berlin and A. Motro. Autoplex: Automated discovery of contents for virtual databases. *Proceedings of COOPIS-01*, pages 108–122, 2001.

[BMW01] Regina M. M. Braga, Marta Mattoso, and Cláudia M. L. Werner. The use of mediation and ontology technologies for software component information retrieval. In *SSR '01: Proceedings of the 2001 symposium on Software reusability*, pages 19–28. ACM Press, 2001.

[BR00] P.A. Bernstein and E. Rahm. Data warehouse scenarios for model management. *Proc 10th Int Conf On Entity-Relationship Modeling*, 1920:1–15, 2000.

[BS01] Glenn B. Bell and Anil Sethi. Matching records in a national medical patient index. *Commun. ACM*, 44(9):83–88, 2001.

[CAFP98] S. Castano, V. De Antonellis, M.G. Fugini, and B. Pernici. Conceptual schema analysis: Techniques and applications. *ACM Trans. Database System*, 23:3:286–333, 1998.

[DDH01] A.H. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. *SIGMOD Conference*, 2001.

[DDL00] A.H. Doan, P. Domingos, and A. Levy. Learning source descriptions for data integration. *Proc WebDB Workshop*, pages 81–92, 2000.

[Die00] R. Diestel. *Graphentheorie*. Springer-Verlag, 2000.

[DMDH02] A.H. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. *WWW*, 2002.

[DMR03] H.-H. Do, S. Melnik, and E. Rahm. Comparison of schema matching evaluations. *GI-Workshop Web and Databases, LNCS*, 2593:221–237, 2003.

[DR02] H.-H. Do and E. Rahm. COMA - A system for flexible combination of match algorithms. *VLDB*, 2002.

[Hal01] Alon Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[HD80] P. Hall and G. Dowling. Approximate string matching. *Computing Survey*, 12:381–402, 1980.

[Inm92] William H. Inmon. *Building the Data Warehouse*. John Wiley & Sons, Inc., 1992.

[LAFP66] V. I. Levenshtein, V. De Antonellis, M.G. Fugini, and B. Pernici. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–7103, 1966.

[LC00] W. Li and C. Clifton. SemInt: a tool for identifying attribute correspondences in heterogeneous databases using neural network. *Data Knowl Eng 33*, pages 49–84, 2000.

[LYHY02] M. Li Lee, L. Huai Yang, W. Hsu, and X. Yang. XML schemas: integration and translation: Xclust: clustering xml schemas for effective integration. *11th CIKM*, 2002.

[MBC+03] J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based schema matching. *Information Integration Workshop at IJCAI*, 2003.

[MBDH02] J. Madhavan, P. A. Bernstein, P. Domingos, and A.Y. Halevy. Representing and reasoning about mappings between domain models. *18th National Conference on Artificial Intelligence (AAAI)*, 2002.

[MBHR04] S. Melnik, P. A. Bernstein, A.Y. Halevy, and E. Rahm. A semantics for model management operators. *Microsoft Technical Report*, 2004.

[MBR01] J. Madhavan, P.A. Bernstein, and E. Rahm. Generic schema matching with Cupid. *Proceedings of the 27th VLDB Conference*, 2001.

[MGMR02] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. *ICDE*, 2002.

[MHH00] R.J. Miller, L.M. Haas, and M. A. Hernández. Schema mapping as query discovery. *26th VLDB*, pages 77–88, 2000.

[MZ98] T. Milo and S. Zohar. Using schema matching to simplify heterogeneous data translation. *ACM SIGMOD Record 30 (1)*, pages 78–83, 1998.

[PVM+02] L. Popa, Y. Velegrakis, R.J. Miller, M. Hernandez, and R. Fagin. Translating web data. *28th VLDB*, 2002.

[RB01] E. Rahm and P.A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10, April 2001.

[RDM04] E. Rahm, H.H. Do, and S. Massmann. Matching large XML schemas. *SIGMOD Record 33(4)*, pages 26–31, 2004.

[SvKJ04] M Smiljanic, M. van Keulen, and W. Jonker. Defining the XML schema matching problem for a personal schema based query answering system. *CTIT Technical Reports*, 2004.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A: Komponenten der GUI

Die GUI ist in Abbildung A.1 zu sehen und beinhaltet eine *Menüleiste*, den *Hauptteil* und unten die *Statusleiste*. Der Hauptteil besteht aus 3 Komponenten - links eine *Buttonleiste* und darunter den *Tabs* , in der Mitte die Darstellung des Ausgangsschemas (Source) und rechts des Zielschemas (Target). Die letzten beiden Komponenten bilden zusammen den *MappingSplitpane*.

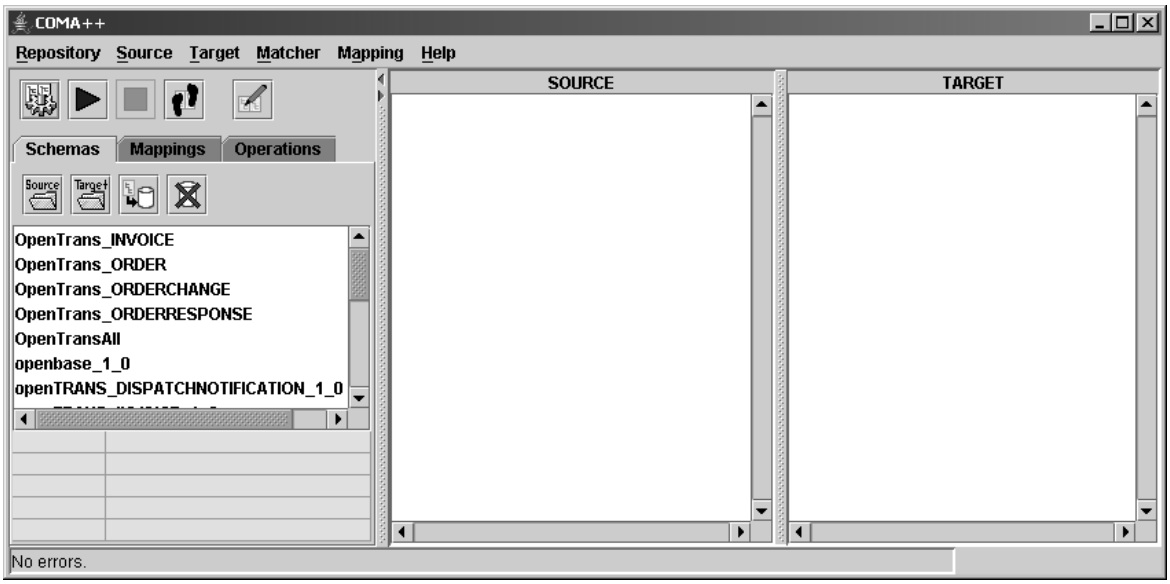


Abbildung A.1: Die GUI des Prototyps COMA++

Menüleiste

In Abbildung A.2 ist die Menüleiste zu sehen, die den Zugriff auf 6 Menüs - Repository, Source, Target, Matcher, Mapping und Help - bietet. Diese sind in Abbildung A.3 ausgeklappt dargestellt.

Die Menüs bieten in Unterpunkten jeweils eine Vielzahl von Funktionen an, die kurz in Tabelle A.1 aufgezeigt werden. Diese können sowohl mit der Tastatur als auch mit der Maus erreicht und gestartet werden.

Abbildung A.2: Die Menüleiste der GUI

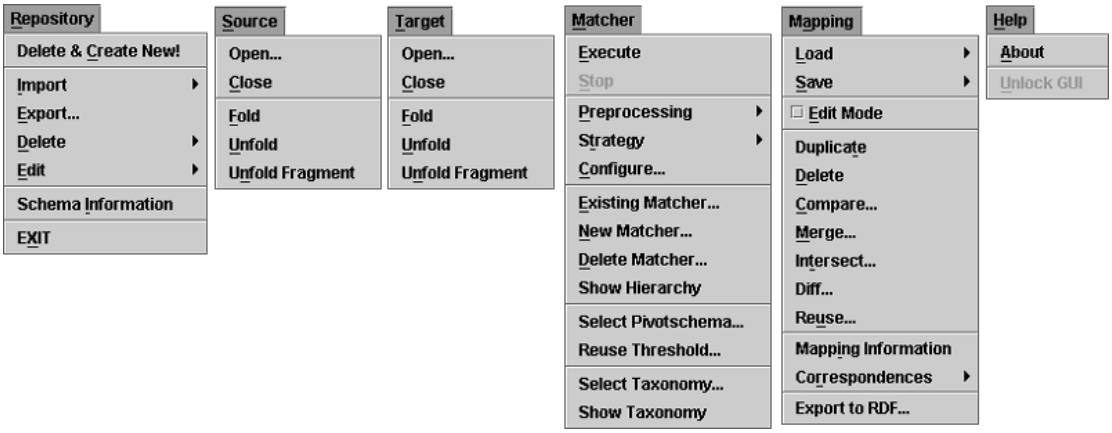


Abbildung A.3: Die aufgeklappten Menüs der Menüleiste

Repository	
Delete & Create New!	Das derzeitige Repository wird gelöscht und ein neues mit Standardmatchern und Synonymen bzw. Abkürzungen erzeugt. *Warnung:* Dabei gehen alle importierten Schemata, gespeicherten Mappings, neu erstellten Matcher und hinzugefügten Synonyme bzw. Abkürzungen verloren!
Import	Für den Import gibt es mehrere Möglichkeiten: **Default Schemas** ermöglicht den Import von Standardschemata, **Default Mappings** von Standardmappings, wobei zuvor die Standardschemata importiert sein müssen), **Schema...** von einem beliebigen Schemata, **Ontology...** einer beliebigen Ontologie.
Export...	Ein beliebiges Schema aus dem Repository wird in eine Datei als ASCII-Baum gespeichert.
Delete	Mit **Schema...** kann ein importiertes Schema aus dem Repository gelöscht werden, mit **Mapping...** wird ein Mapping gelöscht.
Edit	**Synonyms** ermöglicht das Hinzufügen oder Entfernen von Synonymen, **Abbreviations** von Abkürzungen.
Schema Information	Es wird ein Fenster geöffnet, in dem Informationen, wie z. B. die Anzahl der Blätter und die Anzahl aller Pfade, über die aktuelle in der GUI geladenen Schemata angezeigt werden.
EXIT	Beendet die Applikation.
Source/Target	

Fortsetzung auf der nächsten Seite

Tabelle A.1: Menüunterpunkte und ihre Funktion

Fortsetzung der vorherigen Seite	
Open...	Ein beliebiges Schema aus dem Repository wird geöffnet und in die GUI in den Source- bzw. Targetteil des MappingSplitpane geladen.
Close	Schliesst das Schema im Source bzw. Targetteil des MappingSplitpane.
Fold	Der Graph des Source- bzw. Targetschemas wird so gefaltet, dass nur die Wurzel zu sehen ist.
Unfold	Alle Ebenen des Graphes vom Source- bzw. Targetschema werden ausgeklappt.
Unfold Fragment	Das selektierte Fragment im Source- bzw. Targetgraph wird so zusammengefaltet, dass nur noch die Fragmentwurzel zu sehen ist.
Matcher	
Execute	Führt den Matchprozess mit der eingestellten Konfiguration aus.
Stop	Bricht den aktuellen Matchprozess ab. *Warnung*: Die Berechnung kann nicht weitergeführt werden!
Preprocessing	Dient zur Einstellung des Zustandes der Schemata und bestimmt damit die Vorverarbeitungschritte, die auf ihnen ausgeführt werden. Möglich sind **Loaded**, **Resolved**, **Reduced** und **Simplified**.
Strategy	Ermöglicht die Einstellung der Matchstrategie, nämlich **AllContext**, **FilteredContext** und **Fragment**
Configure...	Öffnet einen Dialog, in dem die Matchstrategie und ihre Konfiguration festgelegt werden kann.
Existing Matcher...	Öffnet einen Dialog, in dem alle bereits existierenden Matcher und ihre Konfigurationen angezeigt werden. Diese können konfiguriert oder zum Erstellen neuer Matcher genutzt werden.
New Matcher...	Gibt die Möglichkeit, in einem Dialog einen neuen Matcher mit Hilfe der Auflösung-, Matcher- und Kombinationsbibliotheken zu erstellen.
Delete Matcher...	Matcher, zu denen keine Abhängigkeiten bestehen, können mit dieser Funktion gelöscht werden.
Show Hierarchy	Zeigt in einem neuen Fenster die Hierarchie der Matcher.
Select Pivotschema...	Dient zur Selektion eines Schemas aus den bereits importierten Schemata, welches bei der Mapping-Wiederverwendung als Pivotschema verwendet werden soll (siehe Abschnitt 5.2.5).
Reuse Threshold...	Es kann ein Schwellenwert festgelegt werden, den die Matchprobleme der unvollständigen Mappingpfaden mit der ihnen zugewiesenen Ähnlichkeit mindestens erreichen müssen (siehe Abschnitt 5.2.2).
Select Taxonomy...	Dient zur Selektion einer Taxonomie oder Ontologie, auf die der Taxonomymatcher zurückgreift.
Show Taxonomy	In einem Extrafenster wird die für den *Taxonomymatcher* ausgewählte Taxonomy dargestellt.
	Fortsetzung auf der nächsten Seite

Fortsetzung der vorherigen Seite	
Mapping	
Load	Ein Mapping kann entweder mit **DB...** aus dem Repository oder mit **File...** aus einer Datei geladen werden.
Save	Bietet die Möglichkeit, Mappings entweder mit **DB...** in das Repository oder mit **File...** in eine Datei zu speichern.
Edit Mode	Aktiviert oder deaktiviert den Editiermodus, in dem Korrespondenzen zu Mappings hinzugefügt oder entfernt werden können.
Duplicate	Das in der GUI aktivierte Mapping wird dupliziert.
Delete	Das in der GUI aktivierte Mapping wird gelöscht
Compare...	In einem Dialog können Zielmapping und Testmapping bestimmt werden, die dann miteinander verglichen werden. Das Ergebnis erscheint in einem Extrafenster.
Merge...	In einem Dialog kann man Mappings aus dem Mapping Pool auswählen, die dann mit dem in der GUI aktivierten Mapping zu einem neuen Mapping zusammengefügt werden.
Intersect...	In einem Dialog kann man Mappings aus dem Mapping Pool auswählen. Es wird dann von diesen Mappings und dem in der GUI aktivierten Mapping alle gemeinsamen Korrespondenzen berechnet und daraus ein neues Mapping erzeugt.
Diff...	In einem Dialog kann man Mappings aus dem Mapping Pool auswählen. Es wird dann ein neues Mapping erzeugt mit den Korrespondenzen, die in dem in der GUI aktivierten Mapping vorhanden sind, aber nicht in den selektierten Mappings.
Reuse...	Die dialoggeführte Wiederverwendung, siehe Abschnitt 7.5, wird gestartet.
Mapping Information	In einem Extrafenster werden Metainformationen und alle Korrespondenzen des aktuellen Mappings angezeigt.
Correspondences	Mit **Global (red)** kann die Darstellung von globalen und mit **Local (blue)** von lokalen Korrespondenzen aktiviert und deaktiviert werden. In der Standardeinstellung sind beide Optionen aktiviert.
Export to RDF...	Das in der GUI aktivierte Mapping wird in einem Extrafenster in dem RDF-Format dargestellt.
Help	
About	Öffnet einen Dialog, der einige Informationen über den Prototypen beinhaltet.
Unlock GUI	Wenn einige Funktionen von der GUI gesperrt sind, aber die Operation längst abgeschlossen oder abgebrochen wurde, so können sie damit wieder aktiviert werden.

Globale Buttonleiste

Die globale Buttonleiste ist oberhalb der Registerkarten und enthält 5 Buttons.

Tabs

Bei den *Tabs* handelt es sich um Registerkarten, die jeweils Funktionen und Informationen zu bestimmten Objekten anbieten.Es gibt das *Schemastab, Mappingstab* und

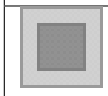
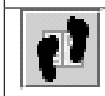
Button	Name	Funktion
	**Configure Strategy**	Öffnet einen Dialog, in dem die Matchstrategie und ihre Konfiguration festgelegt werden kann.
	**Execute**	Führt den Matchprozess mit der eingestellten Konfiguration aus.
	**Stop**	Bricht den aktuellen Matchprozess ab.
	**Step By Step**	Startet den fragmentbasierten Matchprozess, der schrittweise abläuft.
	**Edit Mode**	Aktiviert oder deaktiviert den Editiermodus, in dem Korrespondenzen zu Mappings hinzugefügt oder entfernt werden können.

Tabelle A.2: Die Funktionen der globale Buttonleiste

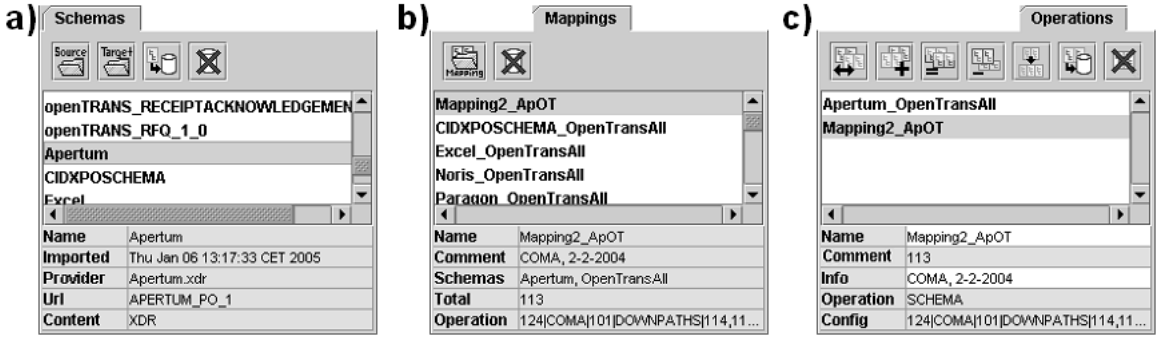


Abbildung A.4: a) Schemastab, b) Mappingstab und c) Operationstab

Operationstab, die in Abbildung A.4 zu sehen. Die Tabs bestehen wiederum aus 3 Komponenten: einer Buttonleiste, einer Objektliste und einem Eigenschaftenfenster.

Tabelle A.3: Die Tabs und ihre Funktion

Schemastab		
Buttonleiste		Schneller Zugriff auf einige Menüpunkte, siehe Tabelle A.1: **Open...** (Source), **Open...** (Target), **Import - Schema...** (Repository), **Delete - Schema...** (Repository)
Objektliste		Liste von allen Schemata, die bereits ins Repository importiert wurden
Eigenschafts-fenster	**Name**	Name des Schemas
	Imported	Zeitpunkt des Importes
	Provider	Quelle bzw. Anbieter des Schemas
	Url	Url der Quelle
	Content	Art der Quelle, z.B. XSD oder RDF
Mappingstab		
Buttonleiste		Schneller Zugriff auf einige Menüpunkte, siehe Tabelle A.1: **Load** (Mapping), **Delete - Mapping...** (Repository)
Objektliste		Liste aller Mappings, die im Repository gespeichert sind
Eigenschafts-fenster	**Name**	Name des Mappings
	Comment	Kommentar
	Schemas	Beteiligte Schemata
	Total	Anzahl der Korrespondenzen
	Operation	Die Operation bzw. Konfiguration, mit der das Mapping erzeugt wurde
Operationstab		
Buttonleiste		Schneller Zugriff auf einige Menüpunkte, siehe Tabelle A.1: **Compare** (Mapping), **Merge** (Mapping), **Intersect** (Mapping), **Diff** (Mapping), **Reuse** (Mapping), **Save** (Mapping), **Delete** (Mapping)
Objektliste		Liste aller Mappings, die zum aktuellen Ausgangs- und Zielschema erzeugt oder aus dem Repository geladen wurden
Eigenschafts-fenster	**Name**	Name des Mappings
	Total	Anzahl der Korrespondenzen
	Comment	Kommentar
	Operation	Operation, mit der das Mapping erzeugt wurde
	Config	Konfiguration des verwendeten Matchers

Statusleiste

Die Statusleiste in Abbildung A.5 wird genutzt, um Informationen und Meldungen an den Benutzer weiterzugeben. Eine Anzeige erfolgt beispielsweise, wenn Schema fertig geladen, ein Dialog geschlossen oder ein Speicherprozess erfolgreich beendet wurde. Eine neue Mitteilung erscheint dabei die ersten 3 Sekunden in fetter Schrift, um die Aktualität anzudeuten. Danach verändert sich die Schriftart und wird dünn.

Abbildung A.5: Die Statusleiste der GUI

MappingSplitpane

Die MappingSplitpane dient zur Darstellung des Ausgangsschemas (links) und des Zielschemas (rechts) und ist in Abbildung A.6 zu sehen. Wird ein Mapping geladen oder erzeugt, so werden dessen Korrespondenzen durch Linien dargestellt.

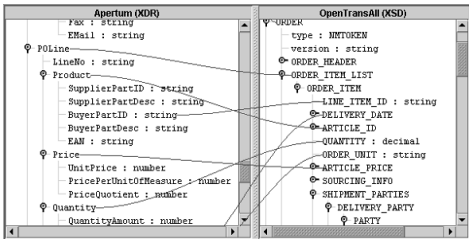


Abbildung A.6: Die MappingSplitpane der GUI

Ehrenwörtliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Ort und Datum

Unterschrift