

HUMBOLDT-UNIVERSITÄT ZU BERLIN



Studienarbeit

Visualisierung fusionierter relationaler Daten

eingereicht am: 27.04.2005

von Karsten Draba

Gutachter: Prof. Felix Naumann
Betreuer: Jens Bleiholder

Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik
Arbeitsgruppe Informationsintegration



Zusammenfassung

Im Rahmen dieser Arbeit werden die Informationen ermittelt, die notwendig sind um fusionierte Daten darzustellen. Es wird gezeigt, wie diese Informationen gewonnen und wie sie verarbeitet werden können. Es wird eine Benutzerschnittstelle vorgeschlagen, die es erlaubt diese Informationen darzustellen. Und es wird gezeigt, wie diese Benutzerschnittstelle den Analyseprozess unterstützt, indem sie an dessen Bedürfnisse angepasst werden kann. Abschließend wird auf Implementierungsdetails eingegangen.

Inhaltsverzeichnis

1	Einleitung	3
2	Datenfusion und ihre Metainformationen	3
2.1	Datenfusion	4
2.2	Informationen über die Herkunft von Daten	7
2.2.1	Arten von provenance	8
2.2.2	Bestimmung der provenance eines Tupels	10
2.3	Darzustellende Informationen	11
3	Definition der Benutzerschnittstelle	12
3.1	Anforderungen an die Benutzerschnittstelle	12
3.2	Aufbau und Interaktionsmöglichkeiten	14
3.2.1	Die Tabelle	14
3.2.2	Das Kontextmenü	17
3.2.3	Die Navigation	18
3.2.4	Der Statusbalken	21
3.3	Die Konzepte und ihre Umsetzung	21
4	Implementierung	22
4.1	Integration in das HumMer-Projekt	22
4.2	Klassen zur Realisierung von provenance	23
4.3	Klassen zur unmittelbaren Visualisierung	25
4.4	Klassen zur Integration in eine bestehende Benutzeroberfläche	26
4.5	Skalierbarkeit der Implementierung	27
5	Zusammenfassung und Ausblick	28

1 Einleitung

Integrierende Informationssysteme bieten eine einheitliche Sicht auf Daten aus heterogenen Informationsquellen. Doch bei der Integration kann eine Vielzahl an Konflikten sowohl auf Schema- als auch auf Datenebene auftreten.¹ Die Auflösung der Konflikte soll automatisch und für die Nutzer verdeckt erfolgen. Gleichzeitig sollen die Nutzer die Kontrolle über den Integrationsprozess behalten und die Qualität des Integrationsprozesses beurteilen können. Dies wird möglich, indem Ursprung und Entstehung der Daten den Nutzern bei Bedarf vermittelt werden. Wissen die Nutzer woher die Daten stammen und wie sie entstanden sind, dann können sie die Daten nicht nur besser verstehen, sondern sie erhalten die Möglichkeit ihre Qualität durch gezielte Anpassung der Quellen und des Prozesses zu verbessern.

Dies gilt auch für den Fusionsprozess. Im Fusionsprozess werden mehrere Repräsentationen eines Objektes der realen Welt zu einer einzigen konsistenten Repräsentation verschmolzen. Es ist die Aufgabe der vorliegenden Arbeit das Ergebnis dieses Prozesses den Nutzern visuell zu verdeutlichen. Es ist das Ziel, eine Benutzeroberfläche zu schaffen, die es den Nutzern erlaubt, sich den Fusionsprozess so transparent wie nötig zu machen. Ursprung und Entstehung sind nicht für alle Daten von gleicher Bedeutung. Deswegen muss eine Visualisierung die Nutzer darin unterstützen, die für sie interessantesten Daten schnell zu finden.

Um das gesteckte Ziel zu erreichen sind drei Schritte nötig. Erstens sollen die zu visualisierenden Informationen bestimmt werden und Wege aufgezeigt werden, wie man zu diesen Informationen gelangt. Zweitens soll eine Benutzeroberfläche inklusive ihrer Funktionalität entworfen werden, die in der Lage ist, diese Informationen dem Nutzer auf geeignete Weise zu vermitteln. Drittens ist die entworfene Oberfläche zu implementieren.

2 Datenfusion und ihre Metainformationen

Im folgenden Abschnitt wird der Prozess der Datenfusion beschrieben. Es wird darauf eingegangen, welche Informationen von Bedeutung sind, um das Ergebnis des Prozesses zu visualisieren und wie diese Informationen gewonnen und modelliert werden können.

¹Eine Aufzählung und Unterscheidung verschiedener Konfliktarten findet sich in [SCS03].

Title	Year	Director
Serenity	2005	Joss Whedon
Citizen Kane	1941	Orson Welles
Donnie Darko	2001	Richard Kelly
Metropolis	2001	⊥

Tabelle 1: Datenquelle Q1

Title	Year	Country
Serenity	2005	USA
Citizen Kane	⊥	USA
Metropolis	1927	Germany
Ying Xiong	2002	⊥

Tabelle 2: Datenquelle Q2

2.1 Datenfusion

Datenfusion ist der letzte Schritt im Integrationsprozess. Ihre Aufgabe ist es mehrere (möglicherweise inkonsistente) Repräsentationen eines Objektes zu einer einheitlichen und konsistenten Repräsentation zusammenzuführen. Sie setzt Informationen über Schemakorrespondenzen und Duplikaterkennung voraus. D.h. es ist bekannt, welche Attribute der verschiedenen Repräsentationen sich auf dieselben Eigenschaften des Objektes beziehen und es ist bekannt, welche Repräsentationen sich auf ein und dasselbe Objekt beziehen. Die Objektidentifikation kann durch einen global eindeutigen Objektidentifikator sichergestellt werden, mit dem jede Repräsentation eines Objektes im Duplikaterkennungsschritt versehen wird.

In den Tabellen 1 und 2 sind zwei Datenquellen gegeben. Beide Datenquellen beschreiben Filme, wobei angenommen wird, dass ein Film über seinen Titel eindeutig identifiziert wird. Das \perp Symbol kennzeichnet Null-Werte. Es ist leicht zu sehen, dass sich die beiden Tabellen auf Schemaebene ergänzen. Denn im Gegensatz zu Quelle Q2 besitzt Quelle Q1 Informationen zum Regisseur eines Films, während Quelle Q2 Informationen zum Produktionsland besitzt, die Quelle Q1 nicht besitzt. So wird beim Zusammenführen beider Datenquellen der Film `Citizen Kane` in Quelle Q2 um den Regisseur `Orson Welles` durch Quelle Q1 erweitert. Versucht man aber beide Datenquellen auf Datenebene zusammenzuführen, so ergibt sich dabei eine Reihe an Konflikten. Diese Konflikte behindern eine einheitliche und konsistente Sicht auf die Objekte. Denn wenn die Konflikte nicht aufgelöst werden, existieren mehrere sich widersprechende Repräsentationen ein und desselben Objektes. Es lassen sich zwei Konfliktfälle unterscheiden: Unsicherheit (*uncertainty*) und Widerspruch (*contradiction*). Eine Unsicherheit entsteht, wenn eine der Repräsentationen Informationen enthält die es in einer anderen Repräsentation nicht gibt (Null-Wert). Bei der Integration der beiden Quellen herrscht z.B. Unsicherheit über das Veröffentlichungsjahr von `Citizen`

Kane. Denn während Quelle Q1 1941 als Veröffentlichungsjahr angibt, ist das Veröffentlichungsjahr in Quelle Q2 unbekannt. Geben zwei Repräsentationen unterschiedliche Werte für eine Eigenschaft desselben Objektes an, so liegt ein Widerspruch vor. Dies ist für das Veröffentlichungsjahr von **Metropolis** der Fall, denn während Quelle Q1 als Veröffentlichungsjahr 2001 angibt, gibt Quelle Q2 1927 an.

Doch Quelle Q1 und Quelle Q2 stimmen in ihren Beschreibungen auch teilweise überein und erweitern sich um neue Objekte (Filme). Mehrere Repräsentationen eines Objektes besitzen ein exaktes Duplikat in einem Attribut, wenn sie in diesem Attribut identisch sind. Bei dem Veröffentlichungsjahr des Films **Serenity** handelt es sich um ein exaktes Duplikat, denn sowohl Quelle Q1 als auch Quelle Q2 geben als Veröffentlichungsjahr das Jahr 2005 an. Der Film **Donnie Darko** aus Quelle Q1 ist ein Unikat in dem Sinne, als dass er nur in Quelle Q1 beschrieben wird. Daraus, dass der Film **Donnie Darko** nur in Q1 zu finden ist, lässt sich schlussfolgern, dass bei der Integration von Quelle Q1 und Quelle Q2 bei dem Film **Donnie Darko** weder Duplikate, noch Unsicherheiten oder Widersprüche auftreten werden. Ist ein Tupel in dem oben genannten Sinne ein Unikat, dann sind auch alle seine Attribute Unikate in dem Sinne, dass jede Eigenschaft des Objektes, welches durch das Tupel repräsentiert wird, nur durch ein Attribut dieses Tupels beschrieben wird.

Es lassen sich also vier Konfliktarten unterscheiden: Widerspruch, Unsicherheit, exaktes Duplikat und Unikat. Auch wenn es sich bei exakten Duplikaten und Unikaten nicht um Konflikte im strengen Sinne handelt ist eine Unterscheidung beider Arten dennoch interessant. Denn mit der Anzahl der exakten Duplikate steigt z.B. die Wahrscheinlichkeit dass es sich um einen korrekten Wert handelt.

Der Fusionsprozess lässt sich in zwei Phasen unterteilen. In der ersten Phase werden alle Datenquellen zu einer einzigen Tabelle kombiniert. Dies kann durch das Kreuzprodukt oder durch den Outer-Union Operator geschehen.² Dadurch wird eine einheitliche Sicht auf die Objekte geschaffen, die alle vorhandenen Attribute enthält. Die zweite Phase erzeugt eine konsistente und einheitliche Repräsentation der Objekte. Dazu werden Repräsentationen,

²Der Outer Union Operator vereinigt zwei Relationen miteinander. Dazu wird zuerst das Ausgabeschema bestimmt, dass sich aus allen Attributen der beiden Relationen zusammensetzt. Attribute mit gleichem Namen werden dabei miteinander identifiziert. Dann werden zu jedem Tupel die fehlende Attribute hinzugefügt und deren Werte mit Null-Werten belegt.

Title	Year	Director	Country
Serenity	2005	Joss Whedon	⊥
Serenity	2005	⊥	USA
Citizen Kane	1941	Orson Welles	⊥
Citizen Kane	⊥	⊥	USA
Donnie Darko	2001	Richard Kelly	⊥
Metropolis	2001	⊥	⊥
Metropolis	1927	⊥	Germany
Ying Xiong	2002	⊥	⊥

Title	Year	Director	Country
Serenity	2005	Joss Whedon	USA
Citizen Kane	1941	Orson Welles	USA
Donnie Darko	2001	Richard Kelly	⊥
Metropolis	1927	⊥	Germany
Ying Xiong	2002	⊥	⊥

Tabelle 3: Das Ergebnis des Outer-Union Operators angewandt auf Quelle Q1 und Quelle Q2 und das Ergebnis des Fusionsprozesses

welche dieselben Objekte repräsentieren, gruppiert und anschließend unter Berücksichtigung einer Konfliktlösungsstrategie miteinander verschmolzen. Danach wird auf die Spalten die in der Ausgabe erscheinen sollen projiziert.

Der Outer-Union angewendet auf die Datenquellen Q1 und Q2 ist in Tabelle 3 dargestellt, wobei die Daten zur besseren Übersicht bereits gruppiert wurden. Ein mögliches Ergebnis des Verschmelzungsprozesses ist ebenfalls in Tabelle 3 dargestellt.

Gibt es nur Unikate oder exakte Duplikate in der kombinierten Tabelle, so ist die Aufgabe eine einheitliche und konsistente Sicht zu Erstellen trivial und bereits mit dem DISTINCT Schlüsselwort der Anfragesprache SQL lösbar. Doch was soll im Konfliktfall geschehen? Wie sollen Konflikte gelöst werden? Für den Konfliktfall müssen Festlegungen getroffen werden, welche über die Ausdruckskraft von SQL hinausgehen. Es muss festgelegt werden, welche Spalten ein Objekt identifizieren³ und wie mit Konflikten in einer Spalte

³Das Problem der Objektidentifikation wurde zwar bereits im Duplikaterkennungsschritt gelöst, aber die Entscheidung welche Spalten benutzt werden sollen um die miteinander zu verschmelzenden Tupel zu identifizieren bleibt trotzdem den Nutzern überlassen.

```

SELECT Title, RESOLVE (Year, min), RESOLVE (Director), RESOLVE
(Country)
FUSE FROM Q1, Q2
FUSE BY (Title)

```

Abbildung 1: Beispiel einer Fuse-By Anfrage

umgegangen werden soll. Um diese Festlegungen in einer Datenbankabfrage ausdrücken zu können, wird in [BN04] eine Erweiterung der SQL Syntax vorgeschlagen.

Ein Beispiel für eine solche Anfrage ist in Abbildung 1 zu sehen. Die zur Objektidentifikation benötigten Spalten werden mittels des FUSE BY Schlüsselwortes festgelegt. Für jede Spalte kann in einer RESOLVE Klausel eine Konfliktlösungsfunktion angegeben werden, die verwendet wird um Konflikte in dieser Spalte zu lösen. Wird dies nicht getan, so wird die SQL Funktion COALESCE benutzt. Das Ergebnis der Anfrage in Abbildung 1 ist die in Tabelle 3 abgebildete Relation.

Einige für die Visualisierung interessante Informationen lassen sich also bereits durch ein einfaches Parsen der Fuse-By Anfrage gewinnen. So gibt die RESOLVE Klausel Auskunft darüber welche Konfliktlösungsfunktionen verwendet wurden und die FUSE BY Klausel verrät welche Spalten die Objektidentität bestimmen. Will man aber wissen, welche Tupel zu einer gemeinsamen Repräsentation eines Objektes verschmolzen wurden und welche Werte in diesen Tupeln in welchem Verhältnis zueinander standen, so reicht es nicht aus, sich mit der Anfrage zu beschäftigen, sondern man muss sich den Daten selbst zuwenden.

2.2 Informationen über die Herkunft von Daten

Fusionierte Daten sind das Ergebnis eines Prozesses. Gerade dieser Prozess ist es, welcher sie von anderen Daten unterscheidet. Die Zusammensetzung der Daten ist wesentlich durch die Eingabedaten, die Anfrage und die Semantik des Operators selbst bestimmt.⁴ Um die Funktionsweise des Operators und

Im Normalfall ist die Objektidentifikationsspalte die Spalte, welche im Duplikaterkennungsschritt mit einem globalen Objektidentifikator versehen wurde.

⁴Auch die Implementierung des Operators ist von Bedeutung. Denn die Implementierung kann alle die Räume ausfüllen welche die Semantik freigelassen hat. Dies kann z.B. die Reihenfolge der Tupel in der Ausgabe sein.

damit dessen Ergebnis transparent zu machen, ist es wichtig den Zusammenhang zwischen den Eingabedaten und dem Ergebnis zu verdeutlichen. Das bedeutet aber gerade den Fusionsprozess selbst zu verdeutlichen. Der Problemkreis, der sich mit der Herstellung dieses Zusammenhangs beschäftigt, wird im englischsprachigen Raum unter dem Begriff *provenance*⁵ zusammengefasst. In [BKT01] wird *provenance* als die Beschreibung des Ursprungs eines Datums und des Prozesses durch den dieses Datum in die Datenbank gelangt definiert.⁶

2.2.1 Arten von provenance

Nach [BKT01] sind zwei Arten von *provenance* zu unterscheiden: *why-provenance* und *where-provenance*. Im relationalen Modell beschreibt die *why-provenance* eines Tupels, welche Tupel in der Eingabe einer Transformation dafür verantwortlich sind, dass dieses Tupel in der Ausgabe der Transformation erscheint. Formal lässt sich die *why-provenance* folgendermaßen definieren:

Sei $T(I) = O$ eine Transformation T mit der Eingabe I und der Ausgabe O und $W \subseteq I$. Dann ist das Tupel (T, W) die *why-provenance* eines Tupels $t \in O$, wenn gilt $t \in T(W)$ und für alle $W^* \subseteq W$ gilt $t \notin T(W - W^*)$.⁷

Unter *where-provenance* wird in [BKT01, S. 2] eine Menge von Lokationen in Datenquellen verstanden, welche angeben woher ein Datum stammt. Es handelt sich hierbei um eine Menge von Lokationen, da ein Datum manchmal aus mehreren Lokationen stammen kann. Ein einfaches Beispiel für einen solchen Fall ist der SQL Befehl `DISTINCT`, angewendet auf eine Tabelle, welche nur exakte Duplikate enthält, die aus verschiedenen Datenquellen stammen. Jeder Attributwert in dem Ausgabebetupel könnte aus jedem der Duplikate stammen.

Die Bestimmung der *why-provenance* für den Outer Union und das Kreuzprodukt ist trivial. Und auch für die zweite Phase des Fuse-By Operators fällt deren Bestimmung leicht, denn die *why-provenance* eines Tupels in der Ausgabe des Fuse-By Operators wird durch alle Tupel der Eingabe gebildet, welche zu diesem Tupel verschmolzen wurden. Dies sind aber genau die Tupel, welche in den Werten der Objektidentifikationsspalten übereinstimmen. Ein Beispiel: Die Tupel `(Citizen Kane, 1941, Orson Welles, ⊥)` und

⁵Man findet in der englischsprachigen Literatur auch die Begriffe *pedigree* und *lineage*.

⁶„Data provenance [...] is the description of the origins of a piece of data and the process by which it arrived in a database.“ [BKT01, S. 1]

⁷Die Definition entspricht den in [BCTV04, S. 901] und [CW03, S. 44] gegebenen.

(Citizen Kane, \perp , \perp , USA) aus Tabelle 3 und die durch Abbildung 1 beschriebene Transformation bilden die *why-provenance* des Tupels (Citizen Kane, 1941, Orsen Welles, USA) aus Tabelle 3.

Im Falle des Outer Union Operators und des Kreuzproduktes ist auch die Bestimmung der *where-provenance* trivial. Die Bestimmung der *where-provenance* für die zweite Phase des Fusionsprozesses ist jedoch nicht trivial, da sie von der gewählten Konfliktlösungsfunktion abhängig ist. Auch hier ein Beispiel: Die *where-provenance* des Veröffentlichungsjahres 2005 in dem Tupel (Serenity, 2005, Joss Whedon, USA) in Tabelle 3 ist die Menge $\{(Q1, p_1), (Q2, p_2)\}$, da die Konfliktlösungsfunktion COALESCE verwendet wird. D.h. es wird der erste nicht Null-Wert in den zu verschmelzenden Tupeln gewählt. Da aber nicht festgelegt ist welches der beiden Tupel das erste Tupel ist, kann der Wert sowohl aus dem Tupel in Quelle Q1 als auch aus dem Tupel in Quelle Q2 stammen. Es steht lediglich fest, dass eines der beiden Tupel das Tupel mit dem ersten nicht Null-Wert war. Die Variablen p_1 und p_2 stehen für Pfade⁸, welche den Wert 2005 im Tupel (Serenity, 2005, Joss Whedon) in Quelle Q1 und im Tupel (Serenity, 2005, USA) in Quelle Q2 bezeichnen.

Die *where-provenance* der Werte eines Tupels wird durch die *why-provenance* des Tupels eingegrenzt. Denn die *where-provenance* eines Datums ist entweder leer (d.h. der Wert wurde von der Transformation neu generiert), oder sie setzt sich aus der *where-provenance* eines oder mehrerer Werte der Tupel in der *why-provenance* zusammen.

Der Unterschied beider *provenance* Arten sei noch einmal verdeutlicht. Während sich die *why-provenance* aus einer Transformation und einer Menge von Tupeln zusammensetzt, besteht die *where-provenance* aus einer Menge von Pfaden in Datenquellen.

Ein mögliches Datenmodell für die *why-provenance* und die *where-provenance* ist in der Abbildung 2 dargestellt.

Eine Trennung zwischen Datenquelle und Pfad wie sie von mir bei der *where-provenance* vorgenommen wurde ist nicht zwingend erforderlich. Denn die Datenquelle kann auch Teil des Pfades sein, genauso wie der Pfad Teil der Datenquelle sein kann. Eine Trennung ist jedoch von Vorteil, wenn man jede beliebige Beschreibung einer Datenquelle oder eines Pfades zulassen möchte und trotzdem zwischen Datenquelle und Pfad unterscheiden können muss.

⁸Repräsentationen eines Pfades können beispielsweise Tripel sein wie in [BCTV04] oder Baumstrukturen wie in [BKT01].

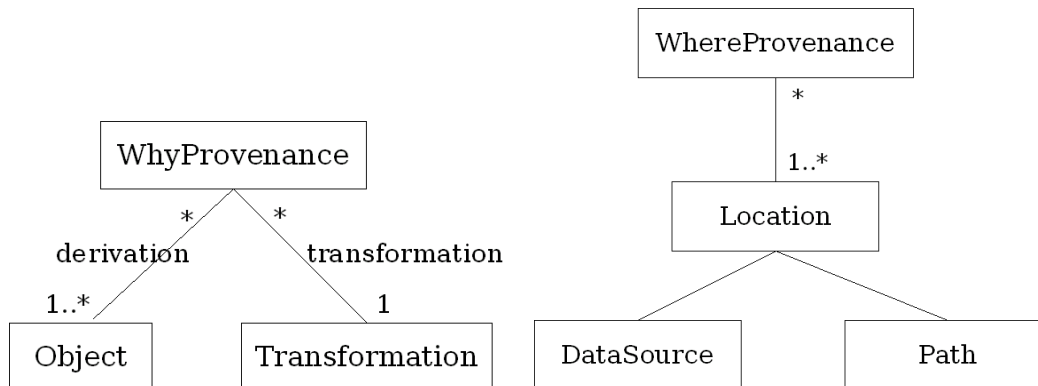


Abbildung 2: Modellierung der why-provenance und where-provenance

2.2.2 Bestimmung der provenance eines Tupels

In der Literatur finden sich zwei Ansätze zur Bestimmung der *provenance*. Während der eine Ansatz die Informationen zur *provenance* bereits während der Anfragebearbeitung erstellt, indem er Annotationen erzeugt und durch die einzelnen Transformationen propagiert, zielt der andere Ansatz darauf ab *provenance* Informationen durch das Zurückverfolgen der Daten durch die Transformationen im Nachhinein zu gewinnen. Ein Verfahren für den ersten Ansatz, welchen ich als Annotations-Ansatz bezeichnen will, findet sich in [BCTV04]. Der zweite Ansatz, im Folgenden als Rückverfolgungs-Ansatz bezeichnet, wird in [BCTV04] und [CW03] beschrieben. Beide Ansätze bieten sich für verschiedene Szenarien an. Der Annotations-Ansatz ist überall dort von Vorteil, wo Informationen zur *provenance* ständig einen Teil des gewünschten Anfrageergebnisses bilden. Denn der Annotations-Ansatz generiert die *provenance* Informationen bei jeder Anfrage, egal ob diese benötigt werden oder nicht. Dies führt bei jeder Anfrage zu einem Overhead an Speicherplatz und Laufzeit.

Ein Problem ist dabei die Tatsache, dass das Bedürfnis nach Informationen zur *provenance* häufig erst entsteht, nachdem das Anfrageergebnis vorliegt. Hinzu kommt noch, dass in den meisten Fällen nicht die *provenance* jedes Tupels von Interesse ist. Diesem Problem wird der Rückverfolgungs-Ansatz gerecht, indem er die *provenance* Informationen erst nach der Anfragebearbeitung auf Wunsch ermittelt. Dazu wird eine *lineage tracing procedure* genutzt die unter Berücksichtigung der Eingabedaten und der Ausgabedaten des Operators die Informationen zur *provenance* ermittelt. Doch das Problem

des Rückverfolgungs-Ansatzes ist, dass man einerseits für jede Transformation eine *lineage tracing procedure* benötigt und andererseits die Kosten zur Berechnung der *provenance* Informationen die Kosten zur Ausführung der Anfrage übersteigen können.⁹ In [CW03] wird jedoch gezeigt, dass es für die meisten Transformationen eine *lineage tracing procedure* gibt und auch wie deren Ausführungskosten gesenkt werden können.

Auf Vor- und Nachteile bei der Implementierung insbesondere für ein bestehendes System wird in Abschnitt 4.1 noch eingegangen werden.

2.3 Darzustellende Informationen

Die zu visualisierenden Informationen und ihre Herkunft sollen zusammengefasst werden.

Der Fuse-By Operator bzw. die Anfrage liefern die *Spaltennamen* der Ausgabe, die *Konfliktlösungsfunktion* einer Spalte, und die Information ob eine Spalte zur *Objektidentifikation* genutzt wurde oder nicht.

Die *why-provenance* gibt an, welche Tupel verschmolzen wurden und damit, *welche Werte in der Eingabe zu einem Wert in der Ausgabe verschmolzen wurden*. Aus diesen Werten lässt sich die entsprechende *Konfliktart* ableiten.

Durch die *where-provenance* ist bekannt, aus welcher *Datenquelle* ein Datum stammt.

Die im Folgenden vorgestellte Visualisierung sieht die zweite Phase des Fusionsprozesses als Schwerpunkt der Visualisierung des Fuse-By Prozesses an. Der Outer Union Operator bzw. das Kreuzprodukt werden nur indirekt durch die Möglichkeit der Rückverfolgung visualisiert. Die Implementierung wird es aber erlauben den Outer Union Operator bzw. das Kreuzprodukt stärker in den Fuse-By Prozess zu integrieren und dem Nutzer dadurch unsichtbar zu machen, ohne schwerwiegende Änderungen am Quellcode der Implementierung vornehmen zu müssen.

⁹Die *lineage tracing procedure* einer Transformation, für die gilt, dass sie für jedes Tupel in der Eingabe, unabhängig von den anderen Tupeln in der Eingabe, ein oder mehrere Tupel produziert (auch *Dispatcher* genannt), muss nicht nur einmal die Eingabedaten komplett einlesen, sondern darüber hinaus über jedem Tupel in der Eingabe die Anfrage ausführen (siehe dazu [CW03, S. 45]).

Ebene	Information
Tabelle	<ul style="list-style-type: none"> • der Operator
Spalte	<ul style="list-style-type: none"> • Spaltenname • Konfliktlösungsfunktion • Objektidentifikationspalte
Zeile	<ul style="list-style-type: none"> • why-provenance
Zelle	<ul style="list-style-type: none"> • Wert • where-provenance • Werte die Verschmolzen wurden • Konfliktart

Tabelle 4: Verteilung der Informationen des Fuse-By Operators in einer Tabelle

3 Definition der Benutzerschnittstelle

Nachdem die darzustellenden Informationen ermittelt wurden, soll eine Benutzeroberfläche vorgestellt werden, die den Nutzern diese Informationen in einer geeigneten Weise präsentiert. Dabei wird auf die einzelnen Komponenten der Benutzerschnittstelle eingegangen und deren Funktionalität erläutert.

3.1 Anforderungen an die Benutzerschnittstelle

Visualisierung fusionierter Daten bedeutet insbesondere die Visualisierung ihrer Entstehung. Da es sich beim Ergebnis der Fusion um relationale Daten handelt, ist für die Visualisierung eine tabellarische Darstellungsweise geeignet. Doch muss eine solche Darstellung erweitert werden, um die charakteristischen Eigenschaften fusionierter Daten zu verdeutlichen. Im Folgenden sollen vier Informationsebenen einer Tabelle unterschieden werden. Die gesamte Tabelle, Spalten, Zeilen und einzelne Zellen. Informationen, die sich auf die gesamte Tabelle beziehen, lassen sich im Tabellenrahmen darstellen. Informationen, die sich auf Spalten beziehen, werden in einer zusätzlichen Zeile (z.B. dem Tabellenkopf) dargestellt und Informationen, die sich auf Zeilen beziehen in einer zusätzlichen Spalte. Informationen, die sich auf eine Zelle beziehen werden in dieser Zelle selbst dargestellt.

Tabelle 4 zeigt eine mögliche Verteilung der im Abschnitt 2.3 aufgezählten Informationen auf eine Tabelle.

In [SSK00] werden einige Anforderungen an die Visualisierung fusionier-

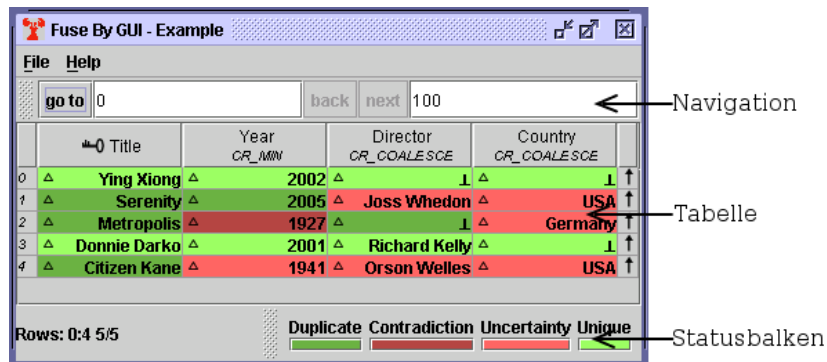


Abbildung 3: Das Hauptfenster

ter Daten gestellt, die auch bei der Visualisierung des Fuse-By Operators berücksichtigt werden sollen. Besonders bei großen Datenmengen ist es wichtig die Menge der visualisierten Daten bestimmen zu können. Dabei bedeutet eine Zunahme an Daten meistens ein Abnahme an Details. Eine Visualisierung sollte nur die Informationen enthalten, die zur Analyse benötigt werden. D.h. dass eine Visualisierung Möglichkeiten zur Verfügung stellen muss, nicht benötigte Informationen zu verstecken. Dies gilt auch für ganze Tupel, die für die Analyse uninteressant sind. Eine solche Einschränkung ist durch Sortierungs- und Filtermechanismen möglich. Wichtig ist es ebenfalls bei mehreren Visualisierungen die Verbindung zwischen den einzelnen Visualisierungen zu erhalten. Denn dadurch wird es ermöglicht Zusammenhänge zwischen den Daten der einzelnen Visualisierungen zu erkennen.

Die hier vorgeschlagene Visualisierung ähnelt der in [SCS03, S. 408] vorgestellten. Allerdings besitzt die hier vorgestellte Visualisierung keine Query-by-Example Funktionalität. Es geht vielmehr darum, das Verständnis des Fusionsprozesses und der Auswirkungen der Konfliktlösungsfunktionen zu fördern, indem den Nutzern Werkzeuge zur gezielten Auswahl und Erfassung wichtiger Informationen gegeben werden. Dazu gehört z.B. auch die Möglichkeit sich gezielt durch große Datenmengen bewegen zu können. Außerdem verfügt die hier vorgestellte Benutzeroberfläche über Konzepte zur Visualisierung der *why-provenance* und der *where-provenance*.

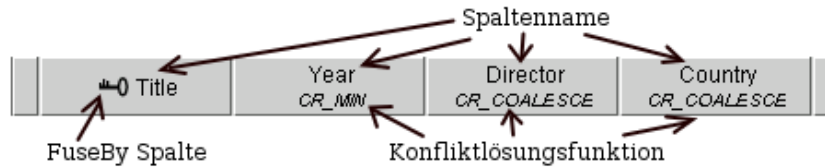


Abbildung 4: Die Kopfzeile der Tabelle

3.2 Aufbau und Interaktionsmöglichkeiten

In Abbildung 3 ist das Hauptfenster der Visualisierung dargestellt. Das Fenster lässt sich in drei Bereiche unterteilen: den Navigations-, den Tabellen- und den Statusbereich. Zusätzlich besitzt die Tabelle ein Kontextmenü.

3.2.1 Die Tabelle

Die Tabelle besitzt eine Kopfzeile (Abbildung 4) und für jedes Tupel eine Datenzeilen. Die Kopfzeile enthält den Spaltennamen und den Namen der Konfliktlösungsfunktion, die vom Fuse-By Operator verwendet wurde, um Konflikte zwischen den Werten dieser Spalte zu lösen. Darüberhinaus zeigt ein kleines Schlüsselsymbol vor dem Spaltennamen an, dass diese Spalte an der Objektidentifikation beteiligt war.

In den Datenzeilen sind die Werte eines Tupels als Zeichenketten repräsentiert. Die Farbe des Zellhintergrundes liefert weitere Informationen zu dem Wert einer Zelle. Ist im Kontextmenü die Ansicht für die *where-provenance* ausgewählt, dann gibt die Farbe die Datenquelle wieder aus der der Wert stammt.¹⁰ Jeder Datenquelle kann eine Farbe zugewiesen werden. Es ist aber nur die Datenquelle selbst und nicht der Pfad in ihr farblich kodiert.¹¹ Ist die Konflikt-Ansicht ausgewählt, dann spiegelt die Farbe die Konfliktart des beim Verschmelzen der Werte aufgetretenen Konfliktes wider. Jede der vier im Abschnitt 2.1 genannten Konfliktarten (Widerspruch, Unsicherheit,

¹⁰Wenn jeder Farbe genau eine Datenquelle zugeordnet wird, dann ist *eine* Farbe nicht notwendigerweise hinreichend um die *where-provenance* eines Wertes anzuzeigen. Denn die *where-provenance* eines Wertes kann mehrere verschiedene Datenquellen enthalten. Doch die Anzeige mehrerer Farben in einer Zelle, wäre unintuitiv und würde es erschweren einen allgemeinen Eindruck zu erhalten.

¹¹Aus diesem Grunde wurde in Abschnitt 2.2.1 zwischen Datenquelle und Pfad unterschieden. Dies bedeutet aber keine Einschränkung, da auch der Attributwert eines Tupel als Datenquelle verstanden werden kann.

2	△	Metropolis	△	1927	△	1	△	Germany	↑
3	△	Donnie Darko	△	2001	△	Richard Kelly	△	1	↑

2	▽	Metropolis	▽	1927	△	1	▽	Germany	↑
		Metropolis		2001				Germany	
		Metropolis		1927				Germany	
3	△	Donnie Darko	△	2001	▽	Richard Kelly	△	1	↑
						Richard Kelly			

Abbildung 5: Zeilen der Tabelle ohne und mit Konfliktwerten

Duplikat und Unikat) kann mit einer eigenen Farbe kodiert werden.

Abbildung 5 zeigt einen Ausschnitt der Tabelle in der Konflikt-Ansicht. Unikate sind in einem hellen Grün, exakte Duplikate in einem dunklen Grün, Unsicherheiten in einem hellen Rot und Widersprüche in einem dunklen Rot gekennzeichnet. Möchte man zu einem Wert die Werte sehen, die zu diesem Wert verschmolzen wurden so kann man sie sich durch einen Mausklick auf das kleine Dreieck in der entsprechenden Zelle anzeigen lassen. Eine so erweiterte Ansicht ist ebenfalls in Abbildung 5 zu sehen. In den Zellen in denen die Ansicht so erweitert wurde, ist das Dreieck nun um 180 Grad gedreht. Durch einen erneuten Mausklick auf das Symbol werden die Werte wieder versteckt.

Zusätzlich zu den Datenspalten, gibt es noch zwei weitere Spalten. Die eine Spalte (in Abbildung 3 ganz links) zeigt die Zeilennummer der Zeile bezogen auf die gesamte Relation an. Die andere Spalte (in Abbildung 3 ganz rechts) enthält ein Symbol. Durch einen Mausklick auf dieses Symbol kann man die Visualisierung der *why-provenance* des in dieser Zeile dargestellten Tupels öffnen oder schließen. Wenn die Visualisierung geschlossen ist, dann wird dies durch einen nach oben gerichteten Pfeil symbolisiert. Ist die Visualisierung geöffnet, so erscheint ein nach rechts gerichteter Pfeil. Da sich die *why-provenance* auf Tupel bezieht, wird das Symbol in allen Visualisierungen die dieses Tupel enthalten entsprechend angepasst.

Abbildung 6 zeigt die Visualisierung der *why-provenance* des Tupels in der dritten Zeile der darunter liegenden Tabelle. Der Name des Fensters in dem die *why-provenance* dargestellt ist, wird aus dem Namen der Transformation und einer Zeichenkettenrepräsentation des Tupel dessen *why-provenance* dargestellt ist gebildet. Sollten die in der *why-provenance* dargestellten Tupel ebenfalls Informationen zur *why-provenance* oder zur *where-provenance*

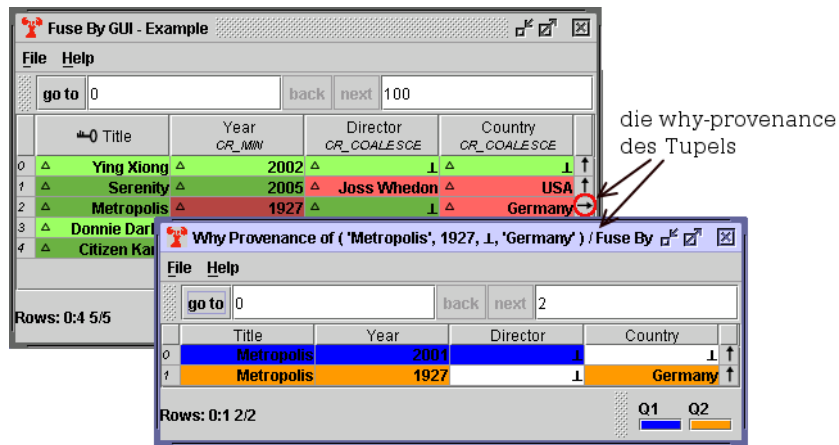


Abbildung 6: Ein Tupel und die Visualisierung seiner *why-provenance*

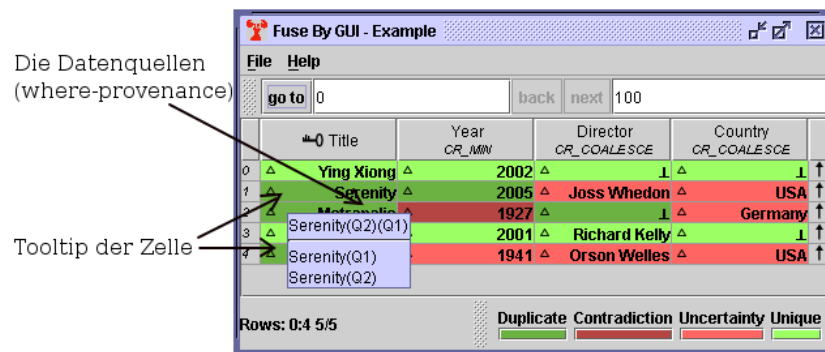


Abbildung 7: Der Tooltip einer Zelle

besitzen, so werden diese Informationen in der Visualisierung der *why-provenance* angezeigt. So lässt sich die *why-provenance*, solange sie gegeben ist, beliebig zurückverfolgen.

Befindet sich der Mauszeiger über einer Zelle, so erscheint ein Tooltip (Abbildung 7), der nicht nur den Ergebniswert und die Konfliktwerte anzeigt, sondern auch die *where-provenance* der Werte. Auch hier wird lediglich die Datenquelle angezeigt, aus der der Wert stammt und nicht der Pfad. Es werden aber alle Datenquellen angezeigt, weswegen der Tooltip mehr Informationen zur *where-provenance* enthält als eine Zelle in der Tabelle. Der Tooltip erlaubt es alle wichtigen Informationen zu einer Zelle zu erhalten, ohne diese expandieren zu müssen. Ein Tooltip hat den Vorteil, dass er in

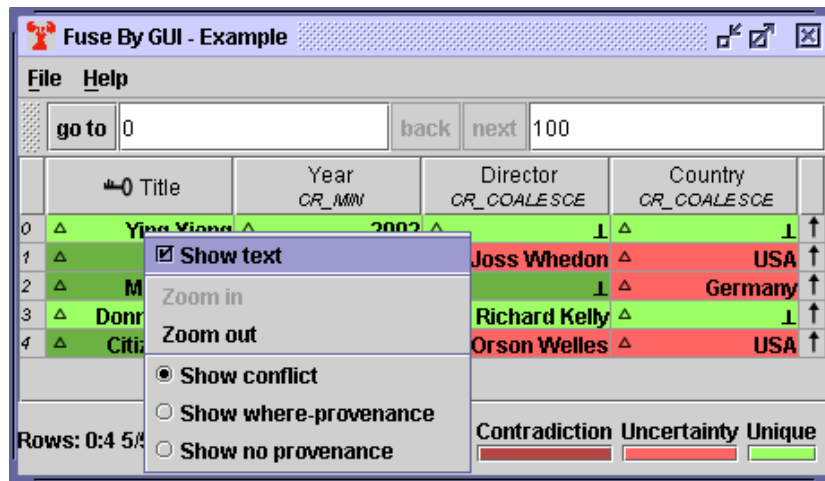


Abbildung 8: Das Kontextmenü

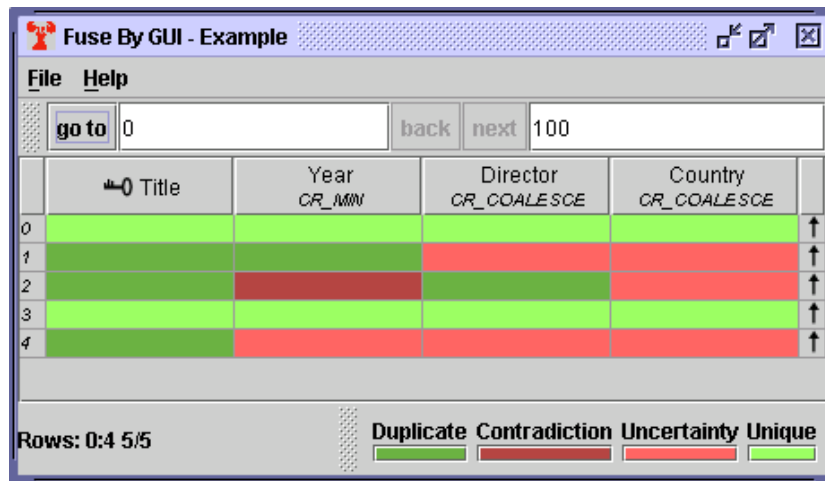
der Regel mehr Platz zur Anzeige der Werte bietet als die Zelle einer Tabelle. Insbesondere bei Tabellen mit vielen Spalten oder langen Zeichenkettenrepräsentationen der Werte ist dies von Vorteil.

3.2.2 Das Kontextmenü

Das Kontextmenü (Abbildung 8) erlaubt es die Visualisierung an die sich aus dem Analyseziel ergebenden Bedürfnisse anzupassen. Interessieren die konkreten Werte nicht, so kann man ihre Darstellung über den ersten Menüpunkt abschalten. Abbildung 9 zeigt die Tabelle aus Abbildung 3 ohne Werte. Die Textrepräsentation der Werte abzuschalten ist z.B. sinnvoll, wenn man nur einen Eindruck über die Verteilung der Konflikte oder der Datenquellen erhalten möchte.

Eine Zoom-Funktion erlaubt es auf Kosten von Details mehr Daten im Blick zu haben. Beim Herauszoomen, wird automatisch die Textrepräsentation der Werte abgeschaltet. Herauszoomen ist daher nur dann sinnvoll, wenn die Konflikte oder die Datenquellen angezeigt werden. Dann erleichtert es der Zoom den Überblick über die Verteilung der Konflikte bzw. Datenquellen zu gewinnen, da man mehr Zeilen übersieht. Ein Beispiel für einen Zoom zeigt Abbildung 10.

Schließlich kann man noch festlegen, welche Information in der Hintergrundfarbe der Zellen kodiert werden soll. Man kann sich die Konfliktklasse



	Title	Year CR_MIN	Director CR_COALESCE	Country CR_COALESCE
0				
1				
2				
3				
4				

Rows: 0:4 5/5

Duplicate Contradiction Uncertainty Unique

Abbildung 9: Tabelle ohne Werte

(Abbildung 3) oder die Datenquelle aus der der Wert stammt (Abbildung 11) anzeigen lassen. Alternativ kann man die Hintergrundfarbe auch ausschalten (Abbildung 12), wenn für die Analyse nur die Werte von Bedeutung sind.

3.2.3 Die Navigation

Der Navigationsbereich (Abbildung 13) erlaubt es gezielt die Menge der angezeigten Daten einzuschränken. Er besteht aus zwei Eingabefeldern und drei Knöpfen. In das erste Eingabefeld trägt man die Zeilennummer des Tupels ein, das als Erstes in der Ausgabe erscheinen soll. Die Zeilennummer bezieht sich auf die Zeile des Tupels in der gesamten Relation. In das zweite Eingabefenster trägt man die Anzahl der Zeilen ein, die in der Tabelle angezeigt werden sollen. Sollen alle Tupel der Relation angezeigt werden, so trägt man hier eine null ein. Drückt man nun den ersten Knopf, so erscheint das gewählte Tupel als Erstes in der Tabelle und es werden maximal so viele Tupel angezeigt, wie im zweiten Eingabefeld angegeben. Die Knöpfe zwei und drei erlauben es sich mit den gesetzten Einstellungen durch die gesamte Relation zu bewegen. Angenommen die Anzahl der Tupel in der Tabelle wurde auf 50 begrenzt. Wird Knopf zwei gedrückt, so werden die 50 Tupel vor dem bisherigen ersten Tupel angezeigt. Wird Knopf drei gedrückt, so werden die 50 Tupel angezeigt, die auf das letzte Tupel folgen.

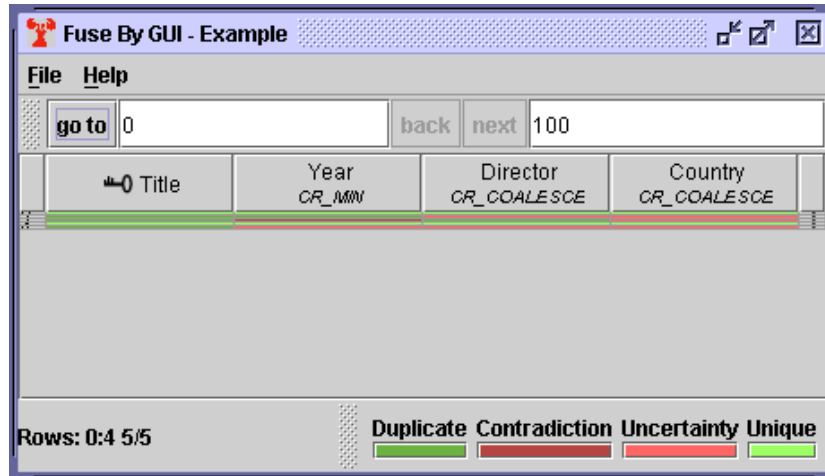


Abbildung 10: Herausgezoomte Tabelle

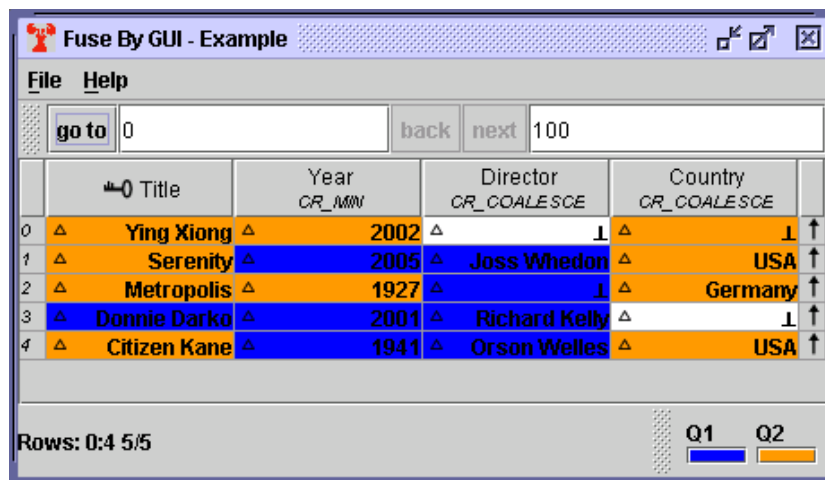


Abbildung 11: Tabelle in der die Hintergrundfarbe die Datenquelle repräsentiert

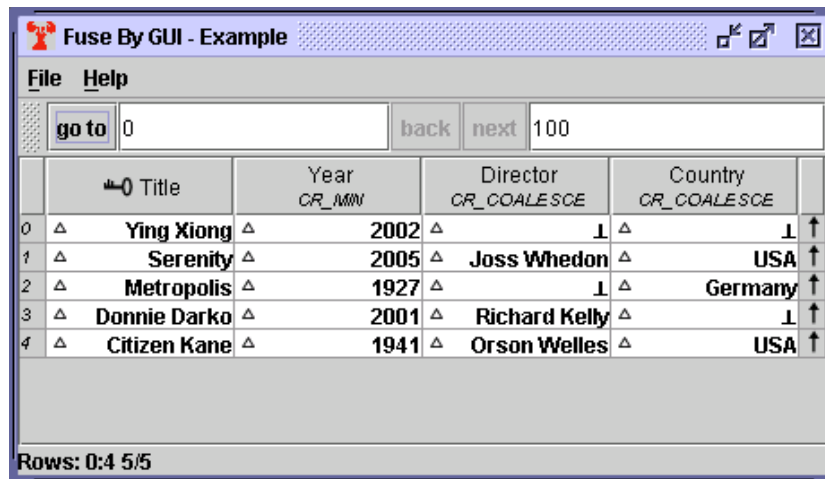


Abbildung 12: Tabelle in der die Hintergrundfarbe deaktiviert ist

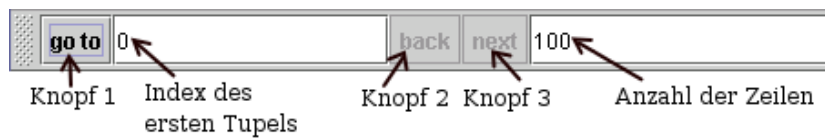


Abbildung 13: Der Navigationsbereich

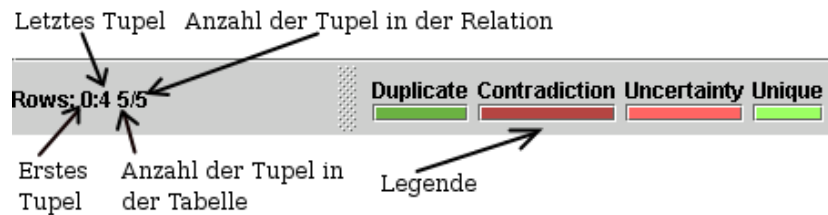


Abbildung 14: Der Statusbalken

3.2.4 Der Statusbalken

Der Statusbalken (Abbildung 14) hilft bei der Orientierung in der Relation und bei der Interpretation der Daten. Er enthält einerseits Informationen darüber, wo man sich gerade in der gesamten Relation befindet und bietet andererseits eine Legende. Die erste Zahl im Statusbalken ist die Zeilennummer die das erste Tupel der Ansicht in der gesamten Relation hat. Die zweite Zahl ist die Zeilennummer, die das letzte Tupel der Ansicht in der gesamten Relation hat. Darauf folgt die Anzahl der in der Tabelle dargestellten Tupel und die Anzahl der Tupel der gesamten Relation.

Die Legende erlaubt es die Hintergrundfarben der Zellen zu interpretieren und sie zu verändern. Eine Änderung der Farbe in einer Visualisierung wirkt sich auf alle anderen Visualisierungen aus. Farbänderungen sind nicht nur wichtig um die Visualisierung der individuellen Farbwahrnehmung anpassen zu können, sondern auch um dadurch Informationen hervorzuheben. Angenommen man interessiert sich nur für die Datenfelder, bei deren Verschmelzung ein Widerspruch zu lösen war. Dann hat man die Möglichkeit den Konfliktklassen Unikat, exaktes Duplikat und Unsicherheit ein und die selbe unauffällige Farbe zuzuweisen (z.B. Weiß) und der Konfliktklasse Widerspruch eine auffällige Farbe (z.B. Rot).

3.3 Die Konzepte und ihre Umsetzung

Die vorgestellte Oberfläche stellt die in Abschnitt 2.3 aufgezählten Informationen dar. Durch Zoomfunktion und Navigation ist die Menge der angezeigten Daten kontrollierbar. Zellen die von besonderem Interesse sind können expandiert werden, um Informationen über die verschmolzenen Werte zu erhalten. Die farbliche Kodierung des Hintergrundes lenkt die Aufmerksamkeit auf die interessanten Daten. Welche Daten dies sind kann festgelegt werden,

indem man wichtigen Datengruppen eine auffällige Farbe zuweist. Sind die Werte oder der Zellhintergrund nicht von Bedeutung kann die Anzeige deaktiviert werden. Ist die *why-provenance* eines Tupels bekannt, so kann sie Transformation für Transformation zurückverfolgt werden und macht so den Entstehungsprozess des Tupels deutlich. Wird in einer Visualisierung die Farbe einer Datenquelle oder Konfliktklasse verändert oder die Visualisierung der *why-provenance* eines Tupels geöffnet, so wird diese Änderung auch auf alle anderen Visualisierungen bezogen. Dadurch ist eine Verbindung zwischen den verschiedenen Visualisierungen hergestellt. Die Statusleiste hilft bei der Orientierung in der Gesamtrelation und die Legende erleichtert die Interpretation der Zellhintergrundfarbe.

4 Implementierung

Die vorgestellte Benutzerschnittstelle wurde in der Programmiersprache Java implementiert. Die wichtigsten der dabei entstandenen Klassen sollen im nächsten Abschnitt vorgestellt und ihre Zusammenhänge erläutert werden.

4.1 Integration in das HumMer-Projekt

Die Implementierung erfolgte im Rahmen des HumMer-Projektes.¹² Dadurch konnten einige bereits bestehende Komponenten genutzt werden. Das Interface *Relation*, das sich in dem Paket `de.hu.mac.hummer.connectivity.relational` befindet und dessen Implementierungen liefern die Eingabedaten für die Visualisierung. Und das Metadaten-Repository bietet die Möglichkeit Datenquellen zu bezeichnen. Im Rahmen des HumMer-Projektes wurde auch der in [BN04] vorgestellte Fuse-By Operator implementiert. Dazu wurde das in [dBBD⁺01] vorgeschlagene XXL Framework benutzt. Dieses Framework besitzt allerdings keine Funktionalität um die *provenance* eines Tupels oder Wertes zu bestimmen. Diese Funktionalität musste erst bereitgestellt werden, wozu jeder der beiden in Abschnitt 2.2.2 vorgestellten Ansätze verwen-

¹²Der Humboldt-Merger (HumMer) ist ein Projekt der Arbeitsgruppe Informationsintegration am Institut für Informatik der Humboldt Universität zu Berlin mit dem Ziel ein Integrationssystem zu schaffen, „welches heterogene Informationen effizient sammelt, qualitativ bewertet, korrekt kombiniert, gegebenenfalls annotiert und schließlich dem Informationssuchenden präzise und verständlich darstellt“ [<http://www.informatik.hu-berlin.de/mac/hummer/>].

det werden konnte. Sowohl für den Fuse-By Operator als auch für den Outer Union wurden die *why-provenance* und die *where-provenance* nach dem Annotations-Ansatz implementiert. Allerdings wurde die *where-provenance* des Fuse-By Operators nicht für alle bisher existierenden Konfliktlösungsfunktionen implementiert.

Tabelle 5 listet die wichtigsten Klassen der Implementierung auf. Sie lassen sich in drei Gruppen aufteilen.

1. Klassen zur Realisierung von *provenance*
2. Klassen die unmittelbar der Visualisierung dienen (Erweiterungen von Swing Komponenten)
3. Klassen die der Integration in eine bestehende Benutzeroberfläche dienen

4.2 Klassen zur Realisierung von provenance

Die Klassen im Paket `de.hu.mac.hummer.provenance` modellieren die *why-provenance* und die *where-provenance*. Die Klassen `DataItemLocation` und `WhyProvenance` sind möglichst allgemein gehalten um so auch jenseits der relationalen Welt und des Fuse-By Operators verwendbar zu sein.

Die Klasse `DataItemLocation` ist eine Implementierung der Klasse `Location` des allgemeinen Modells in Abbildung 2. Sie verwaltet eine Datenquelle und einen Pfad in dieser Datenquelle. Zur Beschreibung der Datenquelle bietet sich ein Objekt der Klasse `MetadataObject` aus dem Paket `de.hu.mac.hummer.metadata` an. Die *where-provenance* eines Datums besteht aus einer Menge von `DataItemLocation` Objekten.

Entsprechend implementiert die Klasse `WhyProvenance` die gleichnamige Klasse aus Abbildung 2. Ein Objekt dieser Klasse fasst eine Transformation und Eingabedaten dieser Transformation zusammen. Wird die Transformation über den Eingabedaten ausgeführt, so muss in der Ausgabe der Transformation das Objekt enthalten sein, dessen *why-provenance* ein Objekt dieser Klasse repräsentiert.

Zur Berechnung der Konfliktklasse einer Spalte in der *why-provenance* wurde eine statische Methode der Klasse `FuseByRelationalWhyProvenance` implementiert.

Die zur *why-provenance* gehörende Transformation wird durch das Interface `Transformation` beschrieben. Eine Transformation stellt das Schema ihrer

<i>de.hu.mac.hammer.provenance</i>
DataItemLocation WhyProvenance Transformation FuseByRelationalWhyProvenance
<i>de.hu.mac.hammer.connectivity.relational</i>
TraceableRTuple
<i>de.hu.mac.hammer.connectivity.relational.swinggui</i>
HummerTable
RelationTable RelationTableModel RelationTableCellRenderer RowNumberRenderer
TraceableRelationTable TraceableRelationTableModel TraceableRelationTableButtonCell TraceableRelationTableButtonCellRenderer TraceableRelationTableCellRenderer
FuseByTable FuseByTableModel FuseByTableCell FuseByTableCellRenderer FuseByTableMouseListener
TableWithLegend ColorLegend
ComponentCreator ComponentCreatorListener DefaultComponentCreator
RelationTableModelWrapper PartRelationModel
RelationPanel
<i>de.hu.mac.hammer.examples.relational.swinggui</i>
FuseByGUIExamplePerson

Tabelle 5: die wichtigsten Klassen

Eingangs- und Ausgangsdaten bereit und ein Mapping zwischen diesen. Dadurch können Eingabe- und Ausgabedaten einer Transformation aufeinander bezogen werden.

Bei der Klasse `TraceableRTuple` handelt es sich schließlich um die Implementierung eines Tupels, welches sowohl die Informationen seiner *where-provenance* als auch die Informationen seiner *why-provenance* verwaltet. Die Klasse befindet sich im Paket `de.hu.mac.hummer.connectivity.relational` und erweitert die Klasse `RTuple`, welche sich im selben Paket befindet, um die *provenance* Funktionalität. Ein `TraceableRTuple` besitzt ein `WhyProvenance` Objekt, das die *why-provenance* des Tupels repräsentiert und eine Menge von `DataItemLocation` Objekten zu jedem Attribut zur Repräsentation der *where-provenance*.

4.3 Klassen zur unmittelbaren Visualisierung

Die Klasse `RelationPanel` stellt die Navigation zur Verfügung. Sie fasst die einzelnen Komponenten (Navigation, Tabelle, Status, Legende) zu einer Einheit zusammen und übernimmt ihre Anordnung.

Zur Darstellung der Daten in einer Tabelle wird die Swing Klasse `JTable` schrittweise um Funktionalität erweitert. Dazu muss auch das jeweilige Datenmodell der Tabelle erweitert werden. Wenn in einer Zelle nicht nur eine einfache Zeichenkette dargestellt werden soll, ist darüber hinaus auch noch die Implementation eines neuen Zellenrenderers erforderlich.

Die erste Erweiterung der Klasse `JTable` ist die Klasse `HummerTable`. Die Klasse `HummerTable` bietet eine Möglichkeit auf einfache Weise die maximal benötigte Zeilenhöhe einer Tabellenzeile zu ermitteln. Diese Fähigkeit wurde zum Expandieren und Kollabieren der Zellen benötigt.

Als nächstes erweitert die Klasse `RelationTable` die Klasse `HummerTable`. In einer Tabelle der Klasse `RelationTable` können Objekte der Klasse `Relation` aus dem Paket `de.hu.mac.hummer.connectivity.relational` visualisiert werden. Dazu wird der `Relation` eine weitere Spalte mit der Zeilennummer des Tupels hinzugefügt und ein Zellenrenderer benutzt, der eine spezielle Darstellung von Null-Werten bietet (`RelationTableCellRenderer`).

Die Klasse `TraceableRelationTable` erweitert die Klasse `RelationTable` um die Funktionalität zur Darstellung der *provenance* eines Tupels. Um die *where-provenance* eines Tupels darzustellen muss die Hintergrundfarbe einer Zelle in Abhängigkeit des Wertes angepasst werden. Deswegen ist ein neuer Zellenrenderer nötig (`TraceableRelationTableCellRenderer`). Für die *why-prove-*

nance wird eine weitere Spalte dem Datenmodell der Tabelle hinzugefügt. In der Spalte findet das Icon Platz, über das man die *why-provenance* einer Spalte öffnen und schliessen kann.

Als letztes erweitert die Klasse `FuseByTable` die Klasse `TraceableRelationTable` um die Möglichkeit einzelne Zellen zu expandieren und so die Werte anzeigen zu können, die miteinander in Konflikt stehen.

Bei Verwendung des Model-View-Controller Konzeptes, wie es die Klasse `JTable` benutzt, kann es sinnvoll sein mehrere Modelle miteinander zu kombinieren. Man kann Modelle kombinieren, indem man sie ineinander verschachtelt. Die Klasse `RelationTableModelWrapper` hilft einem dabei ein Objekt der Klasse `RelationTableModel` um ein anderes Objekt der Klasse `RelationTableModel` zu hüllen. Dies macht sich die Klasse `PartRelationModel` zu nutzen, indem sie sich von der Klasse `RelationTableModelWrapper` ableitet. Die Klasse `PartRelationModel` stellt ein neues Datenmodell zur Verfügung, dass nur einen Bereich der Daten des eingebetteten Datenmodells umfasst. Anwendung findet diese Klasse in der Navigation. Da jeder der Klassen `RTable`, `TraceableRTable` und `FuseByTable` ein Datenmodell der Klasse `RelationTableModel` zugrunde liegt, kann deren Datenmodell beliebig durch Ableitungen der Klasse `RelationTableModelWrapper` erweitert werden.

4.4 Klassen zur Integration in eine bestehende Benutzeroberfläche

Ein Ziel der Visualisierung war es, sie leicht in andere Benutzeroberflächen einbinden zu können. Deswegen wurde als oberster Container ein `JPanel` anstelle eines `JFrame` gewählt. Dadurch kann die Benutzerschnittstelle zu beliebigen anderen Containern hinzugefügt werden. Es ergibt sich aber das Problem, dass bei der Interaktion mit der Oberfläche neue Visualisierungskomponenten entstehen wie z.B. die Visualisierung der *why-provenance*. Damit auch diese Komponenten in andere Benutzeroberflächen integriert werden können, wurde das Interface Paar `ComponentCreator` und `ComponentCreatorListener` nach dem Beobachter Entwurfsmuster geschaffen. Objekte der Klasse `ComponentCreatorListener` können sich bei Objekten der Klasse `ComponentCreator` registrieren um benachrichtigt zu werden, wenn diese eine neue Komponente erschafft. Die neu entstandene Komponente wird bei der Benachrichtigung übergeben, so dass die benachrichtigte Komponente in der Lage ist sie auf eine geeignete Art darzustellen. Eine Beispielimplementation des Interfaces

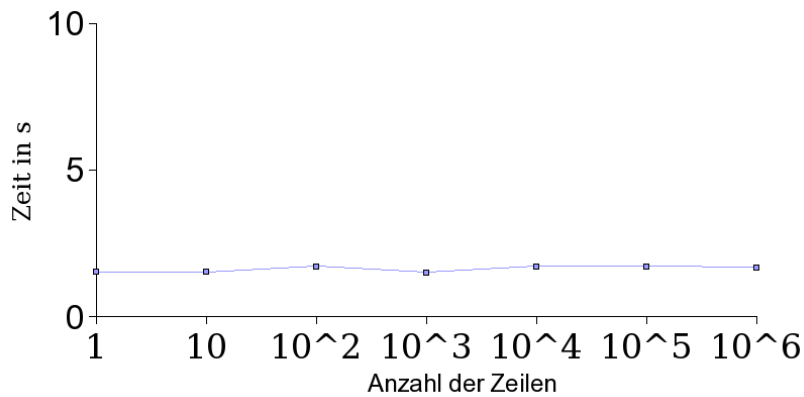


Abbildung 15: benötigte Zeit zur Anzeige der Tabelle in Abhängigkeit der Anzahl der Tupel

`ComponentCreatorListener` ist die Klasse `DefaultComponentCreatorListener`, die alle neuen Komponenten in einem separaten Fenster darstellt. Die Klasse `TraceableRelationTable` implementiert das `ComponentCreatorListener` Interface.

4.5 Skalierbarkeit der Implementierung

In Abbildung 15 ist in Abhängigkeit von der Zeilenanzahl die Initialisierungszeit der Visualisierung dargestellt. D.h. die Zeit, die von der Initialisierung des Objektes vergeht, bis die Visualisierung auf dem Bildschirm erscheint. Es ist deutlich zu sehen, dass die Anzahl der Zeilen nur einen sehr geringen bis gar keinen Einfluss auf die Initialisierungszeit hat. Selbst wenn die Anzahl der Zeilen um den Faktor 10^6 steigt, liegt der Unterschied in der Initialisierungszeit unter einer Sekunde.

Der Grund dafür liegt einerseits in der Implementierung der Klasse `JTable`, die für die Visualisierung nur erweitert wurde. Die Klasse `JTable` fordert immer nur die Daten an, die in der Ausgabe zu sehen sind. Daher steigt zwar mit der Anzahl der sichtbaren Zeilen und Spalten in der Tabelle die Zeit, die für das initiale Zeichnen der Tabelle benötigt wird, aber die Anzahl der Zeilen im Datenmodell hat kaum eine Auswirkung. Besonders beim Scrollen ist diese Vorgehensweise von Vorteil, denn es muss immer nur der Inhalt der Zellen berechnet werden, die durch das Scrollen angezeigt werden.

Andererseits ist es aber auch die einfache Tatsache, dass die Visualisierung die Daten nie vollständig scannen muss. Würden die Daten z.B. vor

dem Anzeigen sortiert, so würde auch mit der Anzahl der Zeilen die Initialisierungszeit steigen.

5 Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurden die Informationen ermittelt, die notwendig sind um fusionierte Daten darzustellen. Es wurde gezeigt, wie diese Informationen gewonnen und wie sie verarbeitet werden können. Dann wurde eine Benutzerschnittstelle vorgeschlagen, die es erlaubt diese Informationen darzustellen. Es wurde gezeigt, dass diese Benutzerschnittstelle den Analyseprozess unterstützt, indem sie an dessen Bedürfnisse angepasst werden kann. Schließlich wurde die Benutzerschnittstelle im Rahmen des HumMer-Projektes implementiert.

Für die Zukunft sind noch viele Erweiterungen der Benutzerschnittstelle möglich. So ist es noch wünschenswert die Daten sowohl nach Werten als auch nach Konfliktarten sortieren zu können. Für einen besseren Überblick würde eine Statistik sorgen, der man z.B. entnehmen kann wieviele Widersprüche in einer Spalte aufgetreten sind oder wie viele Werte einer Spalte aus einer bestimmten Datenquelle stammen. Der Zusammenhang zwischen Konflikten und dem Ursprung der Werte kann durch eine Visualisierung hergestellt werden, die *where-provenance*-Ansicht und Konflikt-Ansicht einander gegenüberstellt. Eine weitere Möglichkeit wäre, die Konfliktklassen als Filter für die *where-provenance*-Ansicht zu benutzen. D.h. eine Möglichkeit sich nur die *where-provenance* der Zellen anzeigen zu lassen, die einer bestimmten Konfliktklasse zugehören. Angenommen, man würde sich nur die *where-provenance* der Zellen anzeigen lassen, in denen ein Widerspruch aufgetreten ist. Dadurch würde man sehr schnell einen Überblick bekommen, welche Datenquelle sich im Falle eines Widerspruchs in welchem Maße durchgesetzt hat. Insbesondere aber bedarf auch die Ansicht der *where-provenance* selbst einer Erweiterung, die es erlaubt die verschiedenen Datenquellen aus denen ein Wert stammen kann optisch zu verdeutlichen, da wie oben bemerkt wurde eine Farbe zu diesem Zweck nicht hinreichend ist.

Auch sollte der Zusammenhang zwischen den einzelnen Fenstern (z.B. dem Hauptfenster und den *why-provenance* Fenstern) optisch stärker hervorgehoben werden. Erreichbar ist dies beispielsweise, indem man in dem Hauptfenster ein Tupel optisch markiert, wenn dessen *why-provenance* gerade das aktive Fenster ist.

Die Visualisierung kann durch Caching-Strategien noch beschleunigt werden. Es ist z.B. möglich die Konfliktarten für einzelne Zellen zu Speichern. Caching-Strategien spielen insbesondere dann eine Rolle, wenn die Verbindung zur Datenquelle sehr langsam ist und wenn die Daten sortiert und gefiltert werden sollen.

Es sollten auch noch andere Visualisierungsmethoden als eine Tabelle in Betracht gezogen werden. So können z.B. bei sehr großen Datenmengen pixelbasierte Darstellungen einen noch besseren Überblick verschaffen, als dies mit der bisherigen Zoomfunktion möglich ist, da sie noch mehr Daten auf einmal darstellen können.

Literatur

- [BCTV04] BHAGWAT, DEEPAVALI, LAURA CHITICARIU, WANG CHIEW TAN und GAURAV VIJAYVARGIYA: *An Annotation Management System for Relational Databases*. In: NASCIMENTO, MARIO A., M. TAMER ÖZSU, DONALD KOSSMANN, RENÉE J. MILLER, JOSÉ A. BLAKELEY und K. BERNHARD SCHIEFER (Herausgeber): *VLDB*, Seiten 900–911. Morgan Kaufmann, 2004.
- [BKT01] BUNEMAN, PETER, SANJEEV KHANNA und WANG CHIEW TAN: *Why and Where: A Characterization of Data Provenance*. In: *Proceedings of the 8th International Conference on Database Theory*, Seiten 316–330. Springer-Verlag, 2001.
- [BN04] BLEIHOLDER, JENS und FELIX NAUMANN: *FUSE BY: Syntax und Semantik zur Informationsfusion in SQL*. In: *GI Jahrestagung (1)*, Seiten 331–335, Ulm, September 2004.
- [CW03] CUI, Y. und J. WIDOM: *Lineage tracing for general data warehouse transformations*. *The VLDB Journal*, 12(1):41–58, 2003.
- [dBBD⁺01] BERCKEN, JOCHEN VAN DEN, BJÖRN BLOHSFELD, JENS-PETER DITTRICH, JÜRGEN KRÄMER, TOBIAS SCHÄFER, MARTIN SCHNEIDER und BERNHARD SEEGER: *XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries*. In: *Proceedings of the 27th International Conference on Very Large Data Bases*, Seiten 39–48. Morgan Kaufmann Publishers Inc., 2001.
- [SCS03] SATTLER, KAI-UWE, STEFAN CONRAD und GUNTER SAAKE: *Interactive example-driven integration and reconciliation for accessing database federations*. *Inf. Syst.*, 28(5):393–414, 2003.
- [SSK00] SAAKE, GUNTER, KAI-UWE SATTLER und DANIEL KEIM: *Datenbank- und Visualisierungstechnologie in der Informationsfusion*. 2000.