Automatische Erstellung von Integrationslösungen

Markus Eppert

eppert@rhrk.uni-kl.de

1. Einleitung

Durch die weltweit ständig wachsenden Datenmengen wird das Verwalten der Informationen und die Integration aus verschiedenen Datenquellen immer schwieriger. Das manuelle Erstellen einer Integrationslösung ist sehr kostspielig, wodurch die automatische Erstellung von Integrationslösungen immer wichtiger wird. Ziel der Datenintegration ist es, einem Benutzer ein einheitliches Interface auf Daten aus verschiedenen Quellen für Queries bereitzustellen. Der Benutzer soll nicht selbst die einzelnen Quellen durchforsten müssen, sondern über ein integriertes Schema automatisch die Query über alle Quellen laufen lassen können.

Im Folgenden werden einige Ansätze zur automatischen Integration von Daten aus verschiedenen heterogenen Datenquellen vorgestellt. Genau diese Heterogenität ist die Hauptschwierigkeit bei der Datenintegration. Die forderndste Aufgabe bei der automatischen Integration besteht in der Lösung logischer Heterogenitätskonflikte. Zu diesen zählen die schematische, semantische und strukturelle Heterogenität. Die einzelnen Arten werden nun mit Hilfe eines Relation-Attribut-Schemas erläutert.

Semantische Heterogenitätsprobleme treten auf, wenn Attribute mit identischen Namen aus zwei verschiedenen Quellen importiert werden sollen, die aber eine andere Bedeutung haben (Homonyme) oder Attribute aus verschiedenen Datenquellen unterschiedliche Namen bei gleicher inhaltlicher Bedeutung haben (Synonyme). So soll beispielsweise ein Attribut „Straße" integriert werden, das Schema der bereits integrierten Daten enthält jedoch ein Attribut „Adresse", das aber dieselbe Bedeutung hat. Schwierigkeiten mit Homonymen treten z.B. auf, wenn ein Attribut „Adresse" eines Unternehmens in ein Schema integriert werden soll, in dem sich das Attribut „Adresse" auf Personendaten bezieht.

Schematische Heterogenitätsprobleme können wiederum in drei verschiedene Arten eingeteilt werden: Konflikte zwischen Relationen und Attributnamen, zwischen Attributnamen und Attributwerten oder zwischen Relationen und Attributwerten. Abb.1 zeigt ein Beispiel für einen Attributname-Attributwert-Konflikt: In der linken Tabelle werden „männlich" und „weiblich" als Attributnamen zu einer Person angegeben, das Geschlecht der einzelnen Person wird durch den Wert „true" markiert, das falsche Geschlecht mit „false". In der rechten Tabelle, die semantisch dasselbe aussagt, ist „männlich" bzw. „weiblich" der Attributwert zum Attributnamen „Geschlecht".

Name	Weiblich	männlich
Susi	true	false
Peter	false	true
Gabi	true	false

Name	Geschlecht
Susi	weiblich
Peter	männlich
Gabi	weiblich

Abb.1: Attributname-Attributwert-Konflikt

Strukturelle Heterogenitätsprobleme liegen vor, wenn Daten unterschiedlich strukturiert sind, wie z.B. bei Attributen, die bei verschiedenen Datenquellen verschiedenen Tabellen zugeordnet sind.

Im Folgenden werden die automatischen Integrationslösungen von AutoMed, SEMEX und Clio vorgestellt und es wird erläutert, wie diese Systeme u.a. die mannigfaltigen Probleme bei Datenintegration zu lösen versuchen.

2. AutoMed

Das AutoMed-Projekt [Auto06] ist ein domänenunabhängiger Ansatz zur automatischen Datenintegration. Das zugrundeliegende Konzept ist das Hypergraph Data Model, das als erstes erläutert wird. Im Gegensatz zu den gängigeren Integrationskonzepten global-as-view und local-as-view, die im Anschluss erläutert werden, verfolgt AutoMed ein weiteres Integrationskonzept namens both-as-view, das anschließend vorgestellt wird. Abschließend wird die Transformation Manipulation Language, die die neu entstandenen Transformationspfade optimiert und validiert, erklärt.

2.1 Hypergraph Data Model

Das Hypergraph Data Model, das von AutoMed als Grundlage für die Erstellung eines einheitlichen Schemas über viele verschiedene Datenquellen verwendet wird, ist ein beschrifteter und gerichteter Hypergraph bestehend aus Knoten und Kanten. In diesem Graphen können Kanten sowohl Knoten untereinander als auch andere Kanten verbinden. Dies wird von [McPo99] als *verschachtelt* (*nested*) bezeichnet. Ein weiterer Bestandteil des HDM sind die Constraints, die eine Menge von Queries mit booleschen Werten über einem Schema sind. Die Queries selbst sind nicht auf eine Sprache festgelegt, sie kann bei verschiedenen Implementierungen variiert werden. Hier wird das relationale Kalkül als Sprache betrachtet. Knoten können eindeutig über deren Namen identifiziert werden, Kanten und Constraints haben optional einen mit ihnen verbundenen Namen. Ein Schema ist ein Tripel über die Knoten, Kanten und Constraints.

Zwei Schemata gelten als äquivalent, wenn sie die gleichen Instanzen enthalten. Ein Schema S umschließt ein anderes Schema S´ bzgl. einer gegebenen Bedingung, falls eine beliebige Instanz in S´, die die Bedingung erfüllt, ebenfalls Teil des Schemas S ist. Wir schreiben hierfür $S' \leq S$. Schließlich ist S zu S´ bzgl. einer gegebenen Bedingung äquivalent, wenn $S' \leq S$ und $S \leq S'$.

Im HDM sind verschiedene primitive Transformationen definiert, die zur Erstellung eines einheitlichen Schemas zur Datenintegration verwendet werden:

1. *renameNode(fromName, toName)* benennt einen Knoten um, vorausgesetzt der Name in *toName* existiert noch nicht.
2. *renameEdge((fromName, c1, ... , cm), toName)* benennt dementsprechend eine Kante um, welche die Konstrukte *c1, ... , cm* verbindet.
3. *addConstraint(c)* fügt einen Constraint *c* mit der Voraussetzung hinzu, dass *c* als true ausgewertet wird, also dass der Constraint im Modell, in das er eingefügt wird, gültig ist.
4. *delConstraint(c)* löscht einen Constraint, sofern er existiert.
5. *addNode(name, q)* ergänzt einen Knoten mit dem Namen *name*. Die Query *q* beschreibt die in dem neuen Knoten vorhandenen Instanzen. Der Name darf wiederum noch nicht vorhanden sein.
6. *delNode(name, q)* löscht einen Knoten. Hier ist *q* eine Query, die angibt, wie die Instanzen, die im gelöschten Knoten vorhanden waren, aus den Instanzen der weiterhin vorhandenen Schemakonstrukte bzw. Knoten wiederhergestellt werden können. Der zu löschende Knoten muss existieren und darf mit keiner Kante verbunden sein.
7. *addEdge((name, c1, ... , cm), q)* ergänzt eine neue Kante mit dem (optionalen) Namen *name* in eine Sequenz von existierenden Schemakonstrukten *c1, ... , cm.* Die neuen Abhängigkeiten, die durch diese Kante entstehen, sind durch den Wert der Query *q* gegeben. Voraussetzungen hierbei sind, dass die Kante noch nicht vorhanden ist, *c1, ... , cm* existieren und *q* die bereits vorhandenen Constraints im Schema nicht verletzt.
8. *delEdge((name, c1, ... , cm), q)* löscht die Kante mit dem (optionalen) Namen *name* zwischen den Schemakonstrukten *c1, ... , cm. q* gibt an, wie die durch das Löschen der Kante entfernten Abhängigkeiten zwischen den Knoten, die durch die Kante verbunden wurden, aus den Instanzen der weiterhin vorhandenen Schemakonstrukte wiederhergestellt werden können. Die Kante muss wieder existieren und darf nicht Teil einer weiteren Kante sein, was durch die Verschachtelung des HDM möglich wäre.

Eine zusammengesetzte Transformation ist eine Sequenz von mehreren primitiven Transformationen. In der folgendenen Definition wird der Begriff eines nicht definierten Modells verwendet. Ein nicht definiertes Modell kann entstehen, wenn die bei den Transformationen angegebenen Voraussetzungen nicht erfüllt werden und die Transformation somit nicht erfolgreich ausgeführt werden kann. Jede Transformation angewendet auf einen undefinierten Wert liefert wiederum einen undefinierten Wert. Eine Transformation t heißt nun *instanzabhängig* bzgl. eines Schemas S, wenn sie nach ihrer Anwendung ein nicht definiertes Modell von S liefern kann, sonst ist t *schemaabhängig* bzgl. S.

Wenn ein Schema S in ein anderes Schema S´ und S´ in S mit Hilfe einer schemaabhängigen Transformation überführbar ist, sind diese Schemata äquivalent. Wenn S in S´ durch eine instanzabhängige Transformation mit einer gegebenen Bedingung überführt werden kann und umgekehrt, dann sind die beiden Schemata äquivalent bzgl. der Bedingung.

Durch die Wiederherstellungs-Query, die bei den Löschfunktionen mitgegeben wird, und die zusätzlichen Voraussetzungen, die von den primitiven Transformationen gefordert werden, ist eine Rücktransformation zum ursprünglichen Modell automatisch ableitbar. Es gibt also für jede primitive Transformation eine primitive Umkehrtransformation. Für *renameNode(from, to)* wäre dies *renameNode(to, from)*. Die Umkehrung von primitiven Transformationen schließt auch die mögliche Umkehrung zusammengesetzter Transformationen ein, indem jede einzelne Transformation in der Sequenz umgekehrt wird.

Auch wenn zwei Datenbanken erstellt werden, um dieselben Informationen zu verwalten, gibt es trotz allem höchstwahrscheinlich Schemaelemente in einem Schema, die keine semantische Entsprechung im anderen Schema haben. Zu diesem Zweck wurden in HDM für AutoMed vier weitere Transformationen definiert. Wenn ein Schema R ein Konstrukt enthält, das nicht aus einem anderen Schema S abgeleitet werden kann, dann wird das Schema S um das fehlende Konstrukt erweitert (extended), wobei der Rückgabewert *void* in der zur Transformation gehörenden Query angegeben wird. Die Umkehrfunktion hierzu ist das Löschen (contracting) von Konstrukten in R, um auf das Schema S zu kommen. Hierbei ist der Rückgabewert der Query wieder *void*, wodurch angedeutet wird, dass das gelöschte Objekt nicht aus S wiederhergestellt werden kann. Diese neuen Funktionen werden folgendermaßen definiert: *extendNode(n) = addNode(n, void); contractNode(n) = delNode(n, void); extendEdge(e) = addEdge(e, void); contractEdge(e) = delEdge(e, void).*

Wenn eine zusammengesetzte Transformation t mit R = t(S) eine Transformation der Art *contractNode(n)* oder *contractEdge(e)* enthält, dann kann R als Erweiterung von S bzgl. *n* oder *e* angesehen werden, in dem Sinne, dass R Informationen über *n* oder *e* hat, die in S nicht vorhanden sind. Analog, wenn eine zusammengesetzte Transformation t mit R = t(S) eine Transformation der Art *extendNode(n)* oder *extendEdge(e)* enthält, dann kann S als Teilmenge von R bzgl. *n* oder *e* angesehen werden.

Nehmen wir an, dass Transformationen vor der eigentlichen Integration auf Quellschemata angewendet wurden, so dass alle Konflikte zwischen diesen ausgeräumt wurden und die Schemata somit syntaktisch äquivalent sind. Durch Vereinigungen können diese nun integriert werden. Das resultierende vereinigte Schema hat die Eigenschaften, dass kein Low-Level-Konstrukt (also kein Knoten und keine Kante) existiert, das die ursprünglichen Schemata erweitert (Minimalität), und dass kein Konstrukt existiert, mit dem ein Quellschema das vereinigte Schema zusätzlich erweitern würde (Vollständigkeit).

2.2 GAV und LAV

Die Hauptaufgabe der Datenintegration ist das Erstellen eines globalen Schemas über eine Menge von Datenbanken mit jeweiligen lokalen Schemata. Es gibt verschiedene Möglichkeiten ein solches Schema aufzubauen. Eine besteht darin, ein Konstrukt des globalen Schemas als Sicht über die einzelnen lokalen Schemata zu definieren. Dieser Ansatz wird global-as-view (kurz: GAV) genannt. Die Bearbeitung von Anfragen an das globale Schema ist bei global-as-view sehr durch *view unfolding* möglich. Dabei werden Relationen des globalen Schemas direkt durch ihre Sichten auf die lokalen Schemata ersetzt. Mit local-as-view (LAV) werden die Konstrukte von lokalen Sche-

mata als Sicht über das globale Schema definiert. Das globale Schema bleibt hier beim Ändern, Entfernen und Hinzufügen von Quellen unverändert. Die Bearbeitung von Queries über das globale Schema ist bei LAV jedoch schwieriger als bei GAV.

In [JTM+03] werden einzelne Fälle genannt, mit denen GAV und LAV Probleme haben. GAV kann ein Konstrukt aus einem Quellschema nicht darstellen, wenn z.B. *Geld* in einem Quellschema, das integriert werden soll, ein Attribut ist, das den gleichen Wert wie die Summe der Attribute *Scheine* und *Münzen* aus dem globalen Schema hat. Dann kann GAV weder *Scheine* noch *Münzen* als Sicht über das Quellschema legen, da nur ein Zugriff auf das Attribut *Geld*, das sich eben nicht aus verschiedenen Komponenten zusammensetzt, möglich ist. Also kann keine Query nach der Summe beantwortet werden, auch wenn die korrekte Antwort im Quellschema unter *Geld* zu finden wäre. LAV kann das Attribut *Geld* zwar durch eine Sicht als Summe von den Attributen *Scheine* und *Münzen* aus dem globalen Schema darstellen, eine Frage nach *Scheine* und *Münzen* auf dem globalen Schema könnte aber auch LAV nicht beantworten. Wenn nun umgekehrt *Scheine* und *Münzen* Attribute im lokalen Schema und *Geld* ein Attribut im globalen Schema sind, kann dieser Fall mit LAV nicht dargestellt werden. Dieser umgekehrte Fall könnte dagegen problemlos durch GAV dargestellt werden.

2.3 BAV in AutoMed

AutoMed verfolgt als Alternative zu den zuvor vorgestellten Verfahren den both-as-view-Ansatz (BAV) zur Datenintegration. Hierbei ist das Mapping zwischen verschiedenen Schemata als ein Pfad von primitiven Transformationsschritten, wie in 2.1 beschrieben, aufgebaut. In Abb.2 ist beispielhaft eine Integration von n lokalen Schemata, die bereits in Form des HDM vorliegen, in ein globales Schema dargestellt. Zunächst wird jedes lokale Schema in ein Äquivalenzschema durch entsprechende Transformationsschritte übersetzt. Die Äquivalenzschemata sind syntaktisch äquivalent, so dass eine Vereinigung dieser möglich ist. Zwischen zwei solcher Schemata erzeugt AutoMed automatisch eine weitere Transformation *id*, die die syntaktische Äquivalenz jedes Schemakonstrukts aus den Äquivalenzschemata sicherstellt. Schließlich kann ein beliebiges Äquivalenzschema ausgewählt werden, über das ein globales Schema erstellt wird. Hier können nun Instanzen, die ursprünglich aus verschiedenen lokalen Schemata stammen, durch eine Vereinigung der Äquivalenzschemata kombiniert werden. Alle durch Transformationen entstandenen Zwischenschemata sowie die Quellschemata und globalen Schemata als auch die Transformationspfade zwischen diesen Schemata werden im Metadata Repository von AutoMed gespeichert. Wie AutoMed die unter den lokalen Schemata liegenden Datenquellen behandelt, ob nun z.B. ein Wrapper um die Datenquellen gelegt wird und eine Transformation in eine HDM-Darstellung stattfindet, wurde aus keiner untersuchten Quelle ersichtlich.

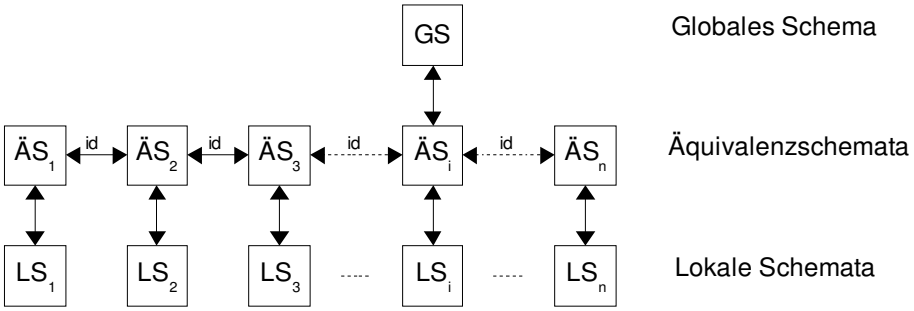

Abb.2: Eine mögliche AutoMed-Integration

2.4 Erzeugung von GAV- und LAV-Sichten aus BAV

Die BAV-Sichtdefinition ist gegenüber LAV und GAV mächtiger, da durch die begleitende Query bei Löschvorgängen eine Transformation in beide Richtungen vorgenommen werden kann. Jedes Auftreten des gelöschten Konstrukts wird durch diese Query ersetzt. Beim Umbenennen von Konstrukten werden die Referenzen auf das alte Konstrukt in der aktuellen Sicht durch Referenzen auf das neue Konstrukt ersetzt. Durch diese zusätzlichen Maßnahmen kann BAV leicht in GAV oder LAV übertragen werden:

Um eine GAV-Sicht für das globale Schema zu erhalten, werden die Pfade von diesem globalen Schema zu jedem lokalen Schema aus dem Metadata Repository von AutoMed entnommen. Jeder Knoten im Baum enthält ein Schema, ob jetzt ein globales, Zwischen- oder lokales Schema. Die Verbindungskanten zwischen benachbarten Schemata stellen jeweils einen Transformationsschritt dar. Sichtdefinitionen für jedes Konstrukt aus dem globalen Schema werden durch einen Top-Down-Durchlauf des Baums erstellt. Zu Beginn ist jede Sichtdefinition auf ein Konstrukt nur das Konstrukt selbst. Jeder Knoten wird von oben nach unten besucht und das Löschen, Erweitern und Hinzufügen wird durch die primitiven Transformationen ausgeführt. Es besteht ebenfalls die Möglichkeit, dass der Baum sich verzweigt. Um zu gewährleisten, dass alle vom globalen Schema ausgehenden Kanten verwendet werden können, werden die Konstrukte aus dem globalen Schema durch eine Disjunktion der entsprechenden Konstrukte aus den Quellschemata ersetzt. Der Baum wird auf diese Weise von der Wurzel bis in die Blätter durchlaufen, bis alle Knoten besucht wurden. Das Ergebnis sind die GAV-Sichtdefinitionen für die globalen Schema-Konstrukte über die lokalen Schemata.

Die LAV-Sichten werden auf ganz ähnliche Weise aus den BAV-Sichten gewonnen. Der Pfad von einem lokalen Schema zum globalen Schema wird wiederum aus dem Metadata Repository von AutoMed abgerufen. Das Vorgehen entspricht dem des GAV-Vorgehens, außer dass mit dem lokalen Schema als Wurzel des Baumes gearbeitet wird. Das Ableiten der LAV-Sichten wird sogar im Vergleich zu den GAV-Sichten einfacher, da es nun nur einen einzigen Pfad ohne Verzweigungen gibt, der verfolgt werden muss.

2.5 Transformation Manipulation Language

Ein Transformationspfad kann durchaus komplementäre Transformationen enthalten, wenn die Zahl und Größe der Schemata in der Datenintegration, die untereinander durch Pfade verbunden sind, wächst. Komplementär ist hier so zu verstehen, dass sich zwei Transformationen gegenseitig aufheben, wie es z.B. bei *addNode(name, q)* und *delNode(name, q)* der Fall wäre, wenn beide primitiven Transformationen in einer zusammengesetzten Transformation vorkommen würden.

Zur Optimierung der Transformationen in AutoMed beschreibt [Tong02] die Transformation Manipulation Language (TML). Ziel dieser Optimierungen ist es, solche Komplemente zu entdecken, den Pfad in einen Pfad ohne die komplementären Transformationen umzubauen und so die Auswertung der Transformationen zu erleichtern. Zunächst wird in diesem Kapitel die Terminologie der TML erklärt, die Definitionen enthält, welche bei den Optimierungen der Transformationen benötigt werden. Diese Optimierungen werden anschließend erläutert.

2.5.1 Terminologie der Transformation Manipulation Language

TML wurde entwickelt, um Transformationen in eine Form zu bringen, welche es ermöglicht, die Schemakonstrukte zu analysieren. TML beschreibt vier Bedingungen, die bei einer Transformation t(i) eines Schemas S(i) in ein Schema S(i+1) berücksichtigt werden müssen: Die positive Vorbedingung enthält die Menge der Konstrukte, die in S(i) vorhanden sein müssen, um t(i) ausführen zu können. Die negative Vorbedingung beinhaltet die Menge der Konstrukte, die in S(i) nicht vorhanden sein dürfen, weil sie durch die Transformation in das Schema ergänzt werden. Analog ist die positive Nachbedingung die Menge der Konstrukte, die in S(i+1) vorhanden sein müssen, und die negative Nachbedingung die Menge der Konstrukte, die in S(i+1) nicht vorhanden sein dürfen.

Die primitiven Transformationen aus 2.1 können in drei Kategorien gegliedert werden: insertion-only, removal-only und insertion-removal. Die Anreicherung oder das Hinzufügen eines Konstrukts sind Insertion-Only-Transformationen, da nur ein einziges Konstrukt einem Schema hinzugefügt wird. Die Löschoperationen sind dementsprechend Removal-Only-Transformationen. Insertion-Removal-Transformationen, wie das Umbenennen, ergänzen ein neues Konstrukt in ein Schema und entfernen gleichzeitig ein anderes Konstrukt aus demselben Schema.

Eine Transformation wird mit TML als insertion-only eingestuft, wenn keine negative Nachbedingung vorhanden ist, da bei dieser Transformation keine Löschfunktion ausgeführt wird. Auf gleiche Weise ist eine Removal-Only-Transformation daran erkennbar, dass sie keine negative Vorbedingung hat, da hier in den Vorbedingungen kein Fehlen eines Konstrukts vorausgesetzt wird. Bei Insertion-Removal-Transformationen existiert sowohl eine negative Vor- als auch eine negative Nachbedingung, da das durch die Transformation eingefügte Konstrukt in der negativen Vorbedingung, das gelöschte Konstrukt in der negativen Nachbedingung stehen muss. Anschaulich gesprochen darf das einzufügende Konstrukt vor der Transformation noch nicht existieren, das gelöschte Konstrukt darf nach der Transformation nicht mehr vorhanden sein.

Ein wohlgeformter Transformationspfad liegt vor, wenn der einzige Unterschied zwischen den Schemakonstrukten S(i) und S(i+1) die speziell durch die Transformation t(i) abgeänderten Konstrukte sind. Außerdem müssen die von t(i) benötigten Konstrukte in den Schemata vorhanden sein.

Zwei Transformationen t(i) und t(i+1) sind komplementär, falls t(i) die Umkehrfunktion von t(i+1) ist und falls die Konstrukte, die in t(i) und t(i+1) abgeändert wurden, die gleichen Instanzen enthalten. In diesem Fall ist das resultierende Schema dasselbe, egal ob beide Transformationen oder eben keine der Transformationen t(i) und t(i+1) angewendet wurden. Anschaulich gesprochen wird ein Konstrukt hinzugefügt, das auch wieder gelöscht wird, oder ein Konstrukt wird mit Informationen angereichert, die wieder entfernt werden.

Außerdem können zwei Transformationen auch partiell komplementär sein, d.h. die beiden Transformationen können zu einer einzigen Transformation vereinfacht werden. Dies ist der Fall, wenn entweder die positive Vorbedingung von t(i) der positiven Nachbedingung von t(i+1) oder die negative Vorbedingung von t(i) der negativen Nachbedingung von t(i+1) entspricht. Wenn einer der beiden Fälle eintritt, gibt es Teile der Transformationen, die komplementär sind. Zusätzlich muss erfüllt sein, dass das, was t(i) löscht, nicht dem entspricht, was t(i+1) in den negativen Vorbedingungen enthält. Dies wird gefordert, da t(i+1) ein Konstrukt c einführen könnte, das aber eine andere Semantik als das von t(i) gelöschte Konstrukt c hat. Somit kann es nicht als dasselbe Konstrukt angesehen werden. Wenn t(i) andererseits ein Konstrukt c einführt, das in der positiven Vorbedingung von t(i+1) vorhanden sein muss, beschreibt c dasselbe Konstrukt, so dass die Operation in t(i+1) auf c mit der in t(i) vereinfacht werden kann. Falls aber t(i+1) eine Removal-Only-Transformation ist und c gelöscht wird, können t(i) und t(i+1) nicht optimiert werden, da die Transformationen nach der ersten Forderung in diesem Absatz nicht partiell komplementär sind. Somit können partiell komplementäre Transformationen keine Insertion-Only-Transformationen gefolgt von Removal-Only-Transformationen sein.

Um diese sehr komplexe Definition zu verstehen, sei hier ein einfaches Beispiel angegeben: Sei t(i) die primitive Transformation *renameNode(A, B)*, t(i) benennt also einen Knoten mit dem Namen *A* in *B* um. Sei weiterhin t(i+1) die primitive Transformation *delNode(B, q)*, t(i+1) löscht also den Knoten mit dem Namen *B*. Diese beiden Transformationen erfüllen alle Forderungen für partiell komplementäre Transformationen. TML würde diese beiden Befehle somit zu *delNode(A, q')* optimieren.

2.5.2 Optimierung mit TML

Nachdem die Transformationsschritte in TML-Notation übertragen wurden, können die Transformationspfade optimiert werden. Sofern ein Pfad wohlgeformt ist, kann man mit TML feststellen, wann die Reihenfolge zweier Transformationsschritte keine Rolle spielt, wann sie vereinfacht werden können und wann sie komplementär und somit überflüssig sind und gelöscht werden können. Ein Transformationspfad muss dazu als wohlgeformt eingestuft werden, bevor irgendwelche Optimierungsregeln angewendet werden können. Die Wohlgeformtheit wird durch die Optimierungsschritte nicht verletzt.

Die Regeln, die komplementäre bzw. partiell komplementäre Transformationen entdecken, können nur auf benachbarte Transformationen angewendet werden. Daher

muss möglicherweise die Reihenfolge der Transformationen in einem Transformationspfad modifiziert werden, um eine Transformation mit anderen Transformationen im Pfad vergleichen zu können, mit denen sie nicht direkt benachbart ist. Um eine Transformation t(i) an eine andere Stelle zu verschieben und mit einer anderen Transformation t(j) zu vergleichen, muss t(i) rekursiv mit der nächsten Transformation t(i+1) im Pfad den Platz tauschen, bis der gewünschte Vergleichspartner erreicht ist. Bei dieser Vertauschung müssen zwei Bedingungen erfüllt sein: t(i+1) darf keine Vorbedingungen haben, die erst von t(i) gewährleistet werden. Wenn also t(i+1) ein Konstrukt P als positive Vorbedingung benötigt, P aber in der negativen Vorbedingung von t(i) vorhanden ist, würde t(i) das Konstrukt P einfügen. Bei einem Tausch der Plätze würde also das Konstrukt P gar nicht vorhanden sein, obwohl t(i+1) es benötigt. Analog ist der Fall, bei dem t(i+1) ein Konstrukt P in der negativen Vorbedingung hat, es von t(i) aber erst gelöscht wird. Des Weiteren dürfen die Nachbedingungen von t(i+1) nicht zu den Vorbedingungen von t(i) im Widerspruch stehen, um die Wohlgeformtheit zu gewährleisten. Wenn also ein Konstrukt P in der positiven bzw. negativen Nachbedingung von t(i+1) ist, darf P nicht in der negativen bzw. positiven Vorbedingung von t(i) vorhanden sein. Auf ähnliche Weise dürfen die Nachbedingungen von t(i-1) nicht mit den Vorbedingungen von t(i+1) kollidieren, da diese Transformationen nun direkt nebeneinander liegen. Analog gilt dies für die Nachbedingungen von t(i) mit den Vorbedingungen von t(i+2).

Das Ergebnis einer Transformation ist das Konstrukt, das durch die Transformation ergänzt bzw. gelöscht wurde. Das Ergebnis einer zusammengesetzten Transformation, bestehend aus zwei Transformationen, kann über *aggregate insertion, aggregate removal*, *net insertion* und *net removal* ausgewertet werden. Eine *aggregate insertion* zweier Transformationen ist die Vereinigung aller Konstrukte, die durch diese Transformationen eingefügt wurden. Ein *aggregate removal* enthält entsprechend die Vereinigung aller Konstrukte, die von den Transformationen gelöscht wurden. Die *net insertion* zweier Transformationen beinhaltet die *aggregate insertion* ausschließlich des *aggregate removals*, also die tatsächlich neu eingeführten Konstrukte; *net removal* ist der umgekehrte Fall. Die hieraus hervorgehende vereinfachte Transformation, welche die Verkettung beider Transformationen t(m) und t(n) beschreibt, hat als positive Vorbedingung das, was beide Transformationen benötigen, bevor irgendeine Transformation ausgeführt wird. In einigen Fällen können positive Vorbedingungen von t(m) durch t(n) entfernt werden, so dass deren Existenz und die Existenz der damit betroffenen Konstrukte in der vereinfachten Transformation nicht von Bedeutung sind. Die Konstrukte im *net removal* müssen jedoch vorhanden sein, bevor die vereinfachte Transformation angewendet werden kann. Die Konstrukte, auf denen die Konstrukte der *net insertion* aufbauen, müssen ebenfalls in der positiven Vorbedingung der vereinfachten Transformation vorhanden sein. Da das Konstrukt in der negativen Vorbedingung von t(i) durch t(i) für beliebiges i eingefügt wird, hat die vereinfachte Transformation als negative Vorbedingung die *net insertion* von t(n) und t(m). Nachdem diese Transformation ausgeführt wurde, sind alle Konstrukte vorhanden, die vor der Ausführung bereits existiert haben, inklusive der *net insertion*, exklusive des *net removals*. Also enthält die negative Nachbedingung das *net removal* von t(m) und t(n).

Inwieweit die beschriebenen Optimierungsschritte in AutoMed automatisch ausgeführt werden, wurde aus den betrachteten Quellen nicht ersichtlich. In diesen wird nur

geschrieben, dass TML die automatische Optimierung unterstützen soll. In welcher Form dies geschieht, bleibt offen.

3. SEMEX

Im Gegensatz zum domänenunabhängigen AutoMed ist SEMEX [DHN+04], Abkürzung für SEMantic EXplorer, ein Werkzeug für die Datenintegration in der speziellen Anwendungsdomäne des Personal Information Management (PIM). Mit diesem System kann der Benutzer auf eine Reihe von Informationsquellen zugreifen, seien es nun persönliche oder öffentlich zugängliche, unstrukturierte oder strukturierte Quellen. SEMEX extrahiert aus unterschiedlichen Anwendungen die Zusammenhänge verschiedener Informationen.

Die persönlichen Informationen auf einem Computer sind durch Applikationen verwaltet (wie z.B. e-Mail, Kalender oder Dateien). Um ein bestimmtes Informationssegment zu finden, muss entweder ein Dateisystem durchsucht oder eine einzelne Applikation eingesetzt werden. Die Integration von mehreren Informationsteilen kann so jedoch nur manuell ablaufen. Da der Mensch sich Informationen über Zusammenhänge oder Assoziationen, also Beziehungen zwischen Objekten, merkt, sollte eine logische Sicht auf die Daten aufgebaut werden, die ebenfalls solche Assoziationen zwischen Daten darstellen kann.

Sobald man eine logische Sicht auf die persönlichen Informationen hat, können externe Quellen mit diesen in Verbindung gebracht werden und so persönliche Aufgaben, die die Integration von mehreren externen Quellen benötigen, vereinfachen. Die Datenbestände können von vielen verschiedenen Benutzern gemeinsam verwendet werden. Die Herausforderung zur Implementierung dieser Komponente ist die Entwicklung von Werkzeugen, die es vereinfachen, externe Quellen (auch durch technisch nicht bewanderte Benutzer) einzubinden.

Im Folgenden werden der Aufbau, die On-the-fly-Integration von SEMEX und die verschiedenen Möglichkeiten von Queries in SEMEX beschrieben.

3.1 Der Aufbau von SEMEX

Die wesentlichen Elemente der Architektur von SEMEX sind das Domänenmodell, Assoziationen, Instanzen, das Data Repository, das Schema Matching und die Reference Reconciliation. Abb.3 zeigt diese Architektur. Im Folgenden wird kurz auf diese Begriffe eingegangen, wobei der Schwerpunkt auf Reference Reconciliation liegen wird. Für ausführlichere Darstellungen der Komponenten sei auf [DoHa05] verwiesen.

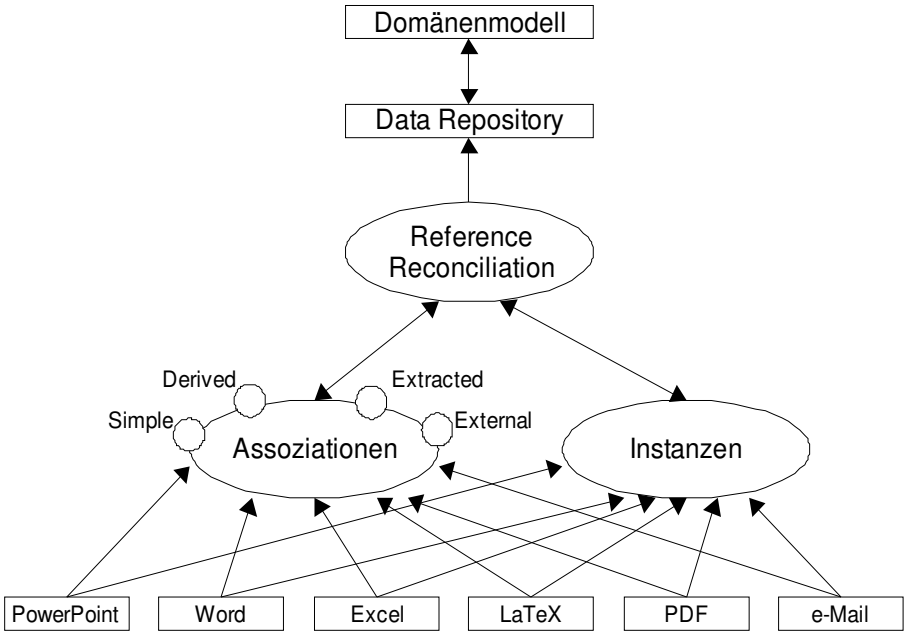

Abb.3: Die Architektur von SEMEX

Wie z.B. auch in SQL haben die einzelnen Objekte und Klassen in SEMEX Attribute und Schlüsselattribute. Letztere sind für jedes Objekt eindeutig und identifizieren es somit aus der gesamten Menge von Objekten. Schlüssel können auch aus mehreren Attributen zusammengesetzt sein, für Studenten könnten z.B. Universität und Matrikelnummer als Schlüssel dienen. Neben diesen beiden Attributarten existieren ebenfalls u.U. Fremdschlüssel, die auf ein anderes Objekt verweisen.

3.1.1 Domänenmodell

Benutzer und Anwendungen interagieren in SEMEX über ein Domänenmodell von persönlichen Informationen, das mit einem objektorientierten Metamodell mit Hilfe von Klassen und Assoziationen definiert ist. Das Standard-Domänenmodell beinhaltet verschiedene Klassen, z.B. Person, Veröffentlichung und Nachricht, sowie Assoziationen, wie „Autor von", „zitiert", „Sender" und „erwähnt in". Das Metamodell unterstützt z. Z. keine Vererbung.

Ein wichtiger Aspekt eines PIM-Systems ist die Möglichkeit, dass Benutzer das Domänenmodell individuell anpassen können. Das Standard-Domänenmodell kann auf verschiedene Arten manuell individualisiert werden, z.B. durch Verfeinerung, Modifizierung oder Generalisierung bereits vorhandener Klassen.

3.1.2 Assoziationen und Instanzen

Eine Grundvoraussetzung in der SEMEX-Architektur ist die Bereitstellung einer Vielzahl von Mechanismen, um Objekt- und Assoziationsinstanzen aus verschiedenen Quellen extrahieren und in das Domänenmodell importieren zu können. Die Hauptmechanismen werden hier kurz dargestellt.

a) *Simple:* Oft sind Objekte und ihre Assoziationen schon leicht ersichtlich in den Datenquellen gespeichert und müssen nur noch in das Domänenmodell übertragen werden. Als Beispiel sei hier eine Kontaktliste oder ein (e-Mail-)Adressbuch genannt, die bereits viele wichtige Attribute von Personen enthalten. Bei einem Adressbuch sind die verschiedenen Attribute bereits in einem Schema, das SEMEX bekannt ist, gespeichert, so dass sie nur noch durch einen Importer ausgelesen und in das Domänenmodell übertragen werden müssen.
b) *Extracted:* Eine große Menge an Objekten und Assoziationen kann durch die Analyse von speziellen Dateiformaten gewonnen werden. Aus LaTeX-Dateien oder Präsentationen können z.B. die Autoren extrahiert werden, Zitate können durch den Vergleich von LaTeX- und Bibtex-Dateien ermittelt werden.
c) *External:* Externe Dienste können viele Assoziationen bereitstellen. SEMEX bietet die Möglichkeit über ein Web-Interface direkt Assoziationen zu gewinnen.
d) *Defined:* Genauso wie Sichten in einer Datenbank abgeleitete Relationen definieren, kann man auch Objekte und Assoziationen aus einfacheren definieren. Ein Beispiel wäre die Erstellung einer Assoziation mit dem Namen Co-Autor zwischen Autoren und Veröffentlichungen.

3.1.3 Data Repository

Nach [CDH+05] speichert SEMEX die neu gewonnenen Assoziationen in einer eigenen Datenbank. Diese Informationen werden mit dem Resource Description Framework (RDF), einer formalen Sprache zur Bereitstellung von Metadaten im Internet, dargestellt. Als RDF-Implementierung kommt das Semantic Web Framework Jena[1] zum Einsatz. SEMEX hält die Informationen in dieser Datenbank aktuell, indem es regelmäßig den Desktop absucht, Instanzen und Assoziationen extrahiert und die Datenbank inkrementell erweitert.

3.1.4 Schema Matching

Zum Importieren von Datensätzen findet der Schema-Matching-Algorithmus von SEMEX zunächst Übereinstimmungen zwischen den Attributen aus der externen Datenquelle und Attributen, die bereits im Domänenmodell vorhanden sind. Neben der Nutzung von früheren Matching-Ergebnissen nutzt SEMEX weitere Vorgehensweisen, um Gemeinsamkeiten bzw. Namensgleichheit zu erkennen. Es berücksichtigt u.a. die Namen der Klassen, in denen die Attribute zu finden sind. Ein einzubindender wissenschaftlicher Text könnte z.B. eine Spalte „conf" enthalten, die aber im Domä-

[1] open source bei [Jena06]

nenmodell nicht zu finden ist. Das Domänenmodell enthält aber eine Klasse „Conference", die übereinstimmende Attribute zu Werten in der Spalte „conf" hat. Des Weiteren wird das Wissen verwendet, dass es überlappende Daten geben kann. In diesem Beispiel könnte beobachtet werden, dass verschiedene Instanzen in der Spalte „conf" auf Instanzen des Attributs „name" der Klasse „Conference" passen. Also schlägt SEMEX vor, diese Spalte auf das Attribut „name" zu mappen. Die Ausgabe dieses Teils besteht aus Tripeln <*a, c, p*>, wobei das Attribut *a* aus der zu importierenden Quelle auf das Attribut *p* der Klasse *c* im Domänenmodell passt.

Im Folgenden sucht der Algorithmus Klassen aus dem Domänenmodell, die durch Daten aus der externen Quelle bzgl. der verwandten Attribute gefüllt werden könnten. Der Output sind Klassenkombinationen, von denen jede aus einer Menge von Klassen besteht und mit Tripeln aus dem ersten Schritt assoziiert werden. Als Beispiel könnte das Attribut „jahr" im zu importierenden Objekt auf „geburtsdatum" der Klasse „Person", auf das „jahr" der Klasse „Conference" oder auf beide gemappt werden, ein anderes Attribut des selben Objekts könnte wieder in einer ganz anderen Klasse im Domänenmodell landen. Die tatsächlichen Assoziationen werden erst im nächsten Schritt festgelegt, hier werden nur alle möglicherweise passenden Kombinationen berechnet.

Abschließend schlägt der Schema-Matching-Algorithmus Assoziationen zwischen Instanzen von Klassen in den Kombinationen, die aus dem vorherigen Schritt resultieren, vor. SEMEX könnte z.B. angeben, dass die Instanz „Komitee" über die Assoziation „Mitglied von" mit der Instanz „Person" in Verbindung steht und ein Zusammenhang zu „Conference" über die Assoziation „organisiert von" besteht. Der Algorithmus berücksichtigt alle möglichen Assoziationen zwischen den Klassen in einer gegebenen Klassenkombination und löscht die vorgeschlagenen Kombinationen, die Integritätsconstraints verletzen würden. Das Ergebnis dieses Schrittes ist eine Menge von möglichen Mappings, die der Benutzer weiter verfeinern oder akzeptieren kann. Es sei noch erwähnt, dass in manchen Fällen der Algorithmus keine brauchbaren Mappings finden kann, weil dem Domänenmodell einige benötigte Klassen oder Assoziationen fehlen. In diesem Fall schlägt SEMEX dem Benutzer vor, das Domänenmodell zu erweitern.

3.1.5 Reference Reconciliation

Der Begriff „Datensatz" spielt in diesem Abschnitt eine wichtige Rolle. Datensatz bedeutet hier eine von möglicherweise vielen Repräsentationen eines Objekts in der Domäne. Sie besteht aus einer Menge von Attributen. Die Attribute eines Datensatzes sind eine Untermenge der Attribute, die mit der Klasse im Domänenmodell verbunden sind. Die spezielle Menge von Attributen in einem Datensatz ist verschieden, abhängig von der Datenquelle, aus der sie extrahiert wurde. Einige Attribute enthalten eine Menge von Werten, andere können Referenzen auf andere Objekte sein.

Unter Reference Reconciliation versteht man im Allgemeinen das Auffinden verschiedener Datensätze, die dasselbe Objekt „der realen Welt" repräsentieren. Wurden mehrere solcher Objekte gefunden, werden diese durch Reference Reconciliation zusammengefügt. Sollten die einzelnen Datensätze der Objekte nicht nur sich überschneidende Attribute enthalten, sondern auch Attribute, die nur in einer Darstellung vorhanden sind, so wird das gemeinsame Objekt um diese angereichert.

Frühere Ansätze für Reference Reconciliation hatten als Zielsetzung, Tupel einer einzigen Datentabelle abzugleichen. Dabei wurde angenommen, dass alle Objekte dieselbe Menge von Attributen haben, jedes Attribut einen einzelnen Wert hat und die Datensätze eine angemessene Anzahl an Attributen (zumindest mehr als zwei) haben.

Für SEMEX kann dieser Ansatz nicht verfolgt werden, da die Datenquellen sehr heterogen sind und z.B. verschiedene Mengen von Attributen enthalten oder die gleichen Attribute mehrere Werte beinhalten. So kann eine Person mehrere e-Mail-Adressen haben. Außerdem kann jede Assoziation sehr beschränkte Informationen enthalten, also nur ein oder zwei Attribute bereitstellen.

Bei existierenden Ansätzen wurde das Reference-Reconciliation-Problem gelöst, indem Datensätze paarweise betrachtet wurden. Im Fall von Personen und im Personal Information Management (PIM) im Allgemeinen ist jeder einzelne Datensatz eher wenig aussagekräftig, enthält also nur wenige Attribute. Der Ansatz von SEMEX besteht darin die Datensätze nach und nach anzureichern, indem sie mit anderen abgeglichen werden. Ein angereicherter Datensatz zu einem Objekt kann somit, wie am Anfang dieses Kapitels erwähnt, eine Menge von Werten für jedes Attribut enthalten. Der Reference-Reconciliation-Algorithmus bekommt eine Menge von Referenzen als Input und arbeitet in drei Schritten:

Im ersten Schritt erfolgt eine Angleichung basierend auf gleichen Werten in Schlüsselattributen: zwei Referenzen werden gemischt, wenn sie einen Wert in einem Schlüssel gemeinsam haben.

Im zweiten Schritt wird die Angleichung basierend auf der Ähnlichkeit von Strings durchgeführt: Wenn also in jedem der übereinstimmenden Attribute der einen Referenz ein Wert in diesem Attribut eine hohe Ähnlichkeit in den Strings der anderen Referenz hat, werden beide Referenzen vereinigt. SEMEX verwendet die *Edit Distance*, um die Ähnlichkeit von Strings zu messen. Unter der *Edit Distance* zwischen zwei Strings versteht man die minimale Anzahl von Operationen, um den einen String in den anderen zu überführen. Eine Operation ist entweder das Abändern, das Einfügen oder das Löschen eines einzelnen Zeichens. Zusätzlich werden domänenabhängige Heuristiken eingesetzt, um bestimmte Strings zu vergleichen, z.B. werden e-Mail-Adressen durch das Wissen über die verschiedenen Komponenten der Adresse und das Wissen, wie Mail-Anbieter die Adressen erzeugen, abgeglichen. Im Fall von Telefonnummern werden möglicherweise fehlende Vorwahlen berücksichtigt.

Im dritten und letzten Schritt wird globales Wissen angewendet: Nachdem mehrere Referenzen zu reicheren zusammengefasst wurden, können nun weitere Entscheidungen zum Mischen basierend auf der Analyse der gesamten Referenz getroffen werden. Als Anmerkung sei noch gesagt, dass in jedem Schritt des Algorithmus konservative Entscheidungen gefällt werden, da die Vermeidung von „false positives", also fälschlicherweise zu einem Objekt zusammengefasste Daten, als besonders wichtig angesehen wird, um ein gutes Browsen von persönlicher Information zu gewährleisten. Es folgen zwei wichtige Beispiele für ein solches globales Wissen:

a) *Time-series comparison:* Der Time-Series-Analysierer wählt Referenzenpaare aus, die zwar in den vorherigen Schritten als ähnlich eingestuft, aber nicht kombiniert wurden. Er sammelt dann für jede Referenz eine Menge von Zeitmarken von den e-Mails, die mit dieser Referenz in Verbindung gebracht wurden. Wenn diese Mengen nur geringe oder gar keine Überschneidungen haben, werden die

Referenzen gemischt. Diese Heuristik ist besonders gut für Personen anwendbar, die das Institut gewechselt haben.

b) *Suchmaschinen-Analyse:* Der Suchmaschinen-Analysierer gibt Texte von zwei Referenzen in Google ein und vergleicht die Top-Treffer. Zwei Referenzen zu derselben Person neigen dazu, ähnliche Top-Treffer zu haben.

Eine weitere Herausforderung des PIM liegt darin, Objekte in mehrere Klassen einzuordnen. Den eben dargestellten Algorithmus mit angemessenen klassenspezifischen Heuristiken isoliert auf jede einzelne Klasse anzupassen würde aber die vielen Interaktionen zwischen den Objekten nicht hinreichend berücksichtigen. Der Algorithmus zur Einteilung in mehrere Klassen wird hier nur an einem Beispiel dargestellt, das aus [DoHa05] stammt:

Gegeben seien folgende Referenzen mit Attributen:

paper	*pa1 =*	*(„Conflict Detection Tradeoffs for Replicated Data", „703-746", {pe1, pe2}, c1)*
	pa2 =	*(„Conflict detection tradeoffs for replicated data", „703-746", {pe3, pe4}, c2)*
person	*pe1 =*	*(„M.J. Carey", null, i1)*
	pe2 =	*(„M. Livny", null, i1)*
	pe3 =	*(„Mike Carey", „carey@cs.wisc.edu", i2)*
	pe4 =	*(„Miron Livny", „miron@cs.wisc.edu", i2)*
inst	*i1 =*	*(„Univ. of Wisconsin Madison", null)*
	i2 =	*(„WISC", „1210 W. Dayton St., Madison, WI 53706-1685")*
conf	*c1 =*	*(„ACM Transactions on Database Systems", „1991")*
	c2 =	*(„TODS", „1991")*

Die erste Phase des Algorithmus gleicht die Objekte einer Klasse einzeln ab. Im Beispiel könnte die Phase die beiden Datensätze *pa1* und *pa2* mischen, da sie eine starke Ähnlichkeit aufweisen. Diese Phase identifiziert auch einige Paare als Kandidaten, also Paare, die möglicherweise noch abgeglichen werden, wenn der Algorithmus weitere Hinweise auf einen Zusammenhang findet.

Die zweite Phase erstellt einen Abhängigkeitsgraphen. Jedes gemischte Paar und jedes Kandidatenpaar stellt einen Knoten dar. Jeder Knoten erhält ein Ähnlichkeitsmaß (eine Zahl zwischen 0 und 1) für die beiden Datensätze, die den Knoten bilden, die in Phase 1 errechnet werden. Eine Kante zwischen den Knoten n und m bedeutet, dass wenn die Ähnlichkeit für m neu berechnet wird, die Ähnlichkeit von n berücksichtigt werden sollte. Der Graph für das Beispiel ist in Abb.4 dargestellt.

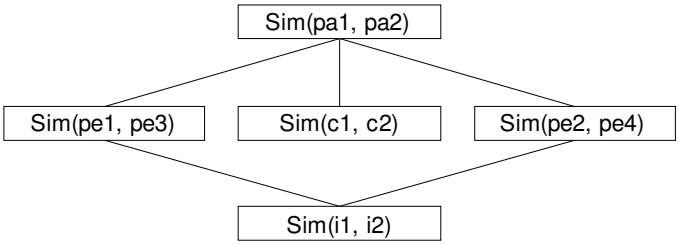

Abb.4: Ähnlichkeitsgraph für das obige Beispiel

Die Vereinigung von *pa1* und *pa2* löst nun die Vereinigung von *c1* und *c2* aus. Nach diesem Vorgang könnten die nächsten betrachteten Objekte die Autorenlisten von *pa1* und *pa2*, also die Vereinigungen von *pe1* mit *pe3* und *pe2* mit *pe4*, sein. Abschließend wäre noch die Vereinigung von *i1* und *i2* möglich.

Anders als bei üblichen Reference-Reconciliation-Vorgängen, die nur einen Schnappschuss einer bestimmten Datenbank betrachten, beschäftigt sich Personal Information Management mit der Verwaltung von größeren Objekten, die längere Zeit überdauern und sich im Lauf der Zeit entwickeln. Im Extremfall ist es noch nicht einmal klar, ob eine Menge von Daten entweder als ein einziges Objekt oder als mehrere Objekte modelliert werden sollen. Dieses Problem könnte dadurch umgangen werden, dass z.B. die verschiedenen Versionen des Anwendungsobjektes „Veröffentlichung" explizit modelliert werden. Komplexe Modelle werden aber zugunsten der Benutzerfreundlichkeit vermieden.

In SEMEX wird zwischen feinkörniger und grobkörniger Reference Reconciliation unterschieden. Erstere geschieht durch den zuvor erklärten Algorithmus. Letztere betrachtet Kandidatenpaare, die aus der feinkörnigen Phase resultieren, und sucht nach Hinweisen, wie der eine Datensatz des Paares mit dem anderen zusammenhängt. Diese Analyse basiert auf der Untersuchung des globalen Wissens über die Referenz. Zur Veranschaulichung betrachten wir die Klasse der Veröffentlichung: Nachdem der feinkörnige Algorithmus angewendet wurde, findet SEMEX eine Menge von Veröffentlichungsobjekten mit ähnlichen Titeln und Autoren, aber evtl. verschiedenen Werten für andere Attribute. SEMEX analysiert dann die Zeitstempel, die mit den Objekten zusammenhängen, um eine mögliche zeitliche Entwicklung zu entdecken.

Die Ergebnisse des grobkörnigen Algorithmus werden dem Benutzer auf eine andere Art dargeboten: Als erstes informiert SEMEX den Benutzer, dass mehrere Objekte als ein einziges dargestellt werden. Zweitens gibt SEMEX dem Benutzer nur die Referenz zu dem jüngsten Objekt an und der Benutzer kann vorherige Objekte auf Wunsch durchsuchen.

3.2 On-the-fly-Integration mit SEMEX

SEMEX baut eine logische Sicht auf Daten auf, was die Integration externer Dateien von mehreren Quellen mit sich bringt. Eine der Hauptaufgaben von SEMEX besteht darin, die Assoziations-Datenbank zu nutzen, um andere Ziele leichter erreichbar zu

machen. SEMEX verwendet diese Datenbank als Hilfsmittel, um externe Datenquellen leichter zu integrieren. Solche Hilfsmittel liefern eine Basis, auf der viele Aufgaben der Datenintegration vereinfacht werden können. Diese werden hier On-the-fly-Integration genannt.

Der wichtigste Bestandteil für On-the-fly-Integration ist die Möglichkeit externe Datenquellen zu importieren. SEMEX stellt Werkzeuge bereit, die den Benutzer durch folgende Schritte beim Datenimport unterstützen:

1. *Datenvorbereitung und Wrapping:* Transformiert die Daten in eine strukturierte Form, mit der SEMEX weiterarbeiten kann. Dies könnten z.B. die Extrahierung von Daten aus Webseiten und das Speichern als XML sein.
2. *Schema matching und mapping:* Ermöglicht die Beschreibung der semantischen Zusammenhänge zwischen der Datenquelle und dem Domänenmodell in SEMEX. Das Ergebnis ist eine Menge von Query-Ausdrücken, die den Import der externen Daten in SEMEX ermöglichen. Ob diese Query-Ausdrücke komplett oder nur teilweise automatisch erstellt werden und wie die Erstellung geschieht, wurde aus den gelesenen Quellen nicht ersichtlich.
3. *Datenimport:* Importiert Daten von der externen Datenquelle nach dem Schema Mapping. An dieser Stelle erfolgt die Reference Reconciliation.
4. *Datenanalyse und Zusammenfassung:* In diesem optionalen Schritt analysiert SEMEX die importierten Daten und versucht Muster zu finden, die für den Benutzer von Interesse sein könnten. So z.B. könnte SEMEX beim Import eines Aufsatzes herausfinden, dass ein großer Teil der Autoren in verschiedenen Texten die Co-Autoren des Benutzers sind oder dass der Aufsatz sich mit anderen Quellen überschneidet, die der Benutzer vorher integriert hat.

3.3 Queries in SEMEX

SEMEX bietet dem Benutzer drei verschiedene Möglichkeiten von Queries. Die erste ist die Suche mit Schlüsselwörtern. SEMEX zeigt alle Objekte, die mit diesem Schlüsselwort in Beziehung stehen. Diese Query kann Mengen von Objekten aus verschiedenen Klassen liefern, im Ergebnis sind sie aber bereits in ihre verschiedenen Klassen eingeteilt (Person, Veröffentlichung usw.). Die zweite Möglichkeit besteht darin, dass der Benutzer schon eine Klasse aus dem Domänenmodell auswählt, bestimmte Werte für einige Attribute dieser Klasse eingibt und somit eine Selektion vornimmt. Zuletzt unterstützt SEMEX ein Query-Interface, mit dem der Benutzer Assoziations-Queries erstellen kann. Eine solche Assoziations-Query ist eine Query, deren Ergebnis aus Tripeln besteht. Jedes Tripel beschreibt eine Assoziation zwischen zwei Objekten. Sobald die Query abgeschlossen ist, kann der Benutzer die Daten durchforsten, indem er den Links, die durch die Assoziationen gebildet werden, folgt. Wenn ein bestimmtes Objekt ausgewählt ist, kann der Benutzer alle Objekte einsehen, die mit diesem in Verbindung stehen. Eine detaillierte Beschreibung der verschiedenen Möglichkeiten findet sich in [CDH+05].

4. Clio

Clio ist ein System, das die komplexen Aufgaben von Transformation und Integration heterogener Daten verwalten und erleichtern soll. Es unterstützt die Generierung und Handhabung von Schemata, der Zusammenhänge zwischen den Schemata und den Mappings von Schemata. Benutzer können mit Clio an die Schema- und Datenbrowser Rückmeldung über die erhaltenen Ergebnisse geben und die Herkunft der Ergebnisse der einzelnen Komponenten Schema Engine, Correspondence Engine und Mapping Engine nachvollziehen. Diese werden im Folgenden erläutert. Eine genauere Übersicht findet sich in [MHH+01].

Eine typische Sitzung mit Clio beginnt damit, dass der Benutzer ein Schema oder mehrere Schemata in das System lädt. Diese können existierende Schemata sein oder ein integriertes Schema enthalten, das manuell oder durch ein Integrationswerkzeug erstellt wurde. Die Schema Engine dient zur Anreicherung des Schemas mit zusätzlichen Constraint-Informationen. Clio verwendet Metadaten wie u.a. Sichtbestimmungen samt Daten, um diese Aufgabe zu erfüllen. Wenn z.B. deklarierte Constraints fehlen, werden die Daten nach möglichen Schlüsseln und Fremdschlüsseln durchsucht. Anschließend werden die Schemata vom Benutzer verifiziert, um die Gültigkeit der generierten Informationen zu gewährleisten. Er kann also Korrekturen in einem Schema vornehmen. Um diesen Prozess zu vereinfachen, bietet Clio eine grafische Benutzeroberfläche an, über die der Benutzer mit dem System kommunizieren kann.

Die Correspondence Engine ist in Clio für das Schema Matching zuständig und erhält als Input jeweils zwei Schemata. Für diese ermittelt sie Paare von Korrespondenzen, die wiederum durch einen Benutzer über eine grafische Benutzeroberfläche verfeinert, geändert oder verworfen werden können.

Zuletzt ermöglicht die Mapping Engine das Erstellen, die Entwicklung und Wartung der Abbildungen zwischen zwei Schemata. Eine Abbildung ist eine Menge von Queries von einer Quelle, aus der die Daten gelesen werden, in ein Zielschema. Clio erzeugt eine Abbildung oder einige Alternativen, die mit den vorhandenen Korrespondenzen und der Information über das Schema konsistent sind. Die Mapping Engine greift also auf Informationen zurück, die sowohl von der Schema Engine als auch von der Correspondence Engine erfasst wurden. Der Benutzer sieht Beispieldaten aus zu importierenden Tabellen und den Inhalt des Zielschemas, wenn diese Tabellen durch die aktuelle Abbildung in das Zielschema übertragen werden würden. Die Beispiele werden so ausgewählt, dass die gegebene Abbildung und die verwendeten Instanzen mit den alternativen Abbildungen und ihren Unterschieden verglichen werden können. In manchen Fällen könnten die Daten, die z.B. durch einen Wechsel von einem inneren Join zu einem äußeren gewonnen werden, durch das Mapping erhebliche Unterschiede aufweisen. In anderen Fällen verändert der Wechsel der Joins das Zielschema in keiner Weise.

Um die Skalierbarkeit und den inkrementellen Aufruf des Werkzeugs zu erlauben, ermöglicht Clio auch das Lesen und Modifizieren von partiellen Abbildungen. Solche Abbildungen können durch eine frühere Benutzung von Clio oder durch ein anderes Integrationswerkzeug erstellt worden sein. Als Beispiel könnte der Benutzer Clio verwendet haben, um ein externes Schema auf das Zielschema zu mappen. Später, nachdem die externe Quelle sich verändert hat, will der Benutzer vielleicht mit Clio eine

neue Abbildung von der modifizierten Quelle in das Zielschema erzeugen. Die alten Abbildungen können dann als Ausgangspunkt für die neue Abbildung dienen.

Der Prozess, in dem die Abbildung definiert wird, ist interaktiv und inkrementell. Clio speichert die aktuelle Abbildung in seiner Wissensbasis und erlaubt dem Benutzer diese durch einen inkrementellen Algorithmus schrittweise auszubauen oder zu verfeinern. Wenn z.B. Korrespondenzen durch die Correspondence Engine ergänzt, gelöscht oder abgeändert werden, verwendet die Mapping Engine die neuen bzw. abgeänderten Korrespondenzen, um die Abbildung zu aktualisieren. Auf ähnliche Weise kann die Mapping Engine die Schema Engine benutzen, um zu überprüfen, ob ein Constraint in einem Schema gültig ist.

5. Schlussbemerkungen

In diesem Text wurden die automatischen Integrationslösungen AutoMed, SEMEX und Clio vorgestellt. Wo SEMEX ein spezieller Integrationsansatz basierend auf persönlichen Informationen ist, sind AutoMed und Clio universeller anwendbar, wobei letzteres einen hohen Grad an Unterstützung vom Benutzer benötigt. Trotz des weit fortgeschrittenen Entwicklungsstandes der einzelnen Systeme gibt es noch viele zu lösende Probleme und offene Fragen, die aus den Quellen nicht ersichtlich wurden, z.B. wie AutoMed die zugrundeliegenden Daten aus den verschiedenen Datenquellen behandelt. Des Weiteren geht aus den Veröffentlichungen ebenfalls nicht eindeutig hervor, welche (Teil-)Aufgaben bei AutoMed und SEMEX tatsächlich automatisch gelöst werden können. Eine Weiterentwicklung der automatischen Integrationslösungen ist also erforderlich. Das Hauptproblem liegt weiterhin in der Semantik, also im Finden der korrekten Matches und der richtigen Zuordnung der zu importierenden Daten. Die Ansätze sind alle vielversprechend, zu einem alltäglichen Einsatz der Systeme ist es aber noch ein recht weiter Weg. Ohne Überprüfung der Ergebnisse bzw. Unterstützung durch den Benutzer sind sie noch nicht anwendbar. Der Titel „Convenience Technology" ist für die automatischen Integrationslösungen noch ein ganzes Stück entfernt.

6. Quellenverzeichnis

Auto06 — Department of Computer Science, Birkbeck College; Department of Computing, Imperial College: The AutoMed Project: An Implementation of Both as View Data Integration. http://www.doc.ic.ac.uk/automed/ . Stand: 25. Mai 2006

CDH+05 — Cai, Y.; Dong X. L.; Halevy, A.; Liu J. M.; Madhavan, J.: Personal Information Management with SEMEX. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD 2005, Baltimore (Maryland), U.S.A., 14. Juni-16.Juni), 2005

DHN+04 — Dong, X.; Halevy, A.; Nemes, E.; Sigurdsson, S. B.; Domingos, P.: SEMEX: Toward On-the-fly Personal Information Integration. In: IIWeb, 2004

DoHa05 — Dong, X.; Halevy, A.: A Platform for Personal Information Management and Integration. In: Proceedings of CIDR, 2005

Jena06 — Hewlett-Packard Development Company, LP: Jena – A Semantic Web Framework for Java. http://jena.sourceforge.net . Stand: Mai 2006

JTM+03 — Jasper, E.; Tong, N.; McBrien, P.; Poulovassilis, A.: View Generation and Optimisation in the AutoMed Data Integration Framework. In: CAiSE Short Paper Proceedings, 2003

MHH+01 — Miller, R. J.; Hernández, M. A.; Haas, L. M.; Yan, L.; Ho, C.T.H.; Fagin, R.; Popa, L.: The Clio Project: Managing Heterogeneity. In: ACM SIGMOD Record, 2001

McPo99 — McBrien, P.; Poulovassilis, A.: Automatic Migration and Wrapping of Database Applications – A Schema Transformation Approach. In: Proc. ER'99, LNCS 1728, S. 96-113. 1999

Tong02 — Tong, N.: Database Schema Transformation Optimisation Techniques for the AutoMed System. Automed Technical Report, März 2002