Auffinden von und Zugriff auf Datenquellen

Seminar: Mastering the Information Explosion
Information Integration and Information Quality

23. Juni 2006

Bearbeiter: Dragan Sunjka, d_sunjka@informatik.uni-kl.de

Betreuer: Dipl.-Inf. Boris Stumm, stumm@informatik.uni-kl.de

FB Informatik, LG Datenverwaltungssysteme, TU Kaiserslautern

1 Motivation

Die immer schnellere Expansion des Internets und die rasante Entwicklung des E-Business hat in den letzten Jahren zu einer wahren Informationsexplosion geführt [LVF+03]. Neue Formen von Organisationen (sog. *virtual organizations*) fordern und fördern neue, globale Kooperationsarten und organisationsübergreifende Informationsintegration. Historisch getrennte Datenquellen können heute dank schnellerer Netzwerk- und Rechnertechnologien miteinander verbunden werden. Die für einen Anwendungsfall relevanten Datenquellen zu finden und auf sie zuzugreifen stellt in einer verteilten, heterogenen und dynamischen Umgebung ein Problem dar, zu dessen Lösung in dieser Ausarbeitung sowohl klassische als auch neuere Architekturen und technologische Ansätze vorgestellt werden. In einer dynamischen Umgebung andere wichtige Aspekte wie z.B. Schema-Mapping, Schema-Matching, etc. werden in anderen Ausarbeitungen im Rahmen dieses Seminars behandelt.

2 Umgang mit autonomen Datenquellen

Informationssysteme großer Unternehmen sind notwendigerweise verteilt. Es ist unausweichlich, dass verschiedene Abteilungen diverse Systeme benutzen, um Daten zu erzeugen, speichern und abzufragen. Die Vielfältigkeit daraus entstehender Systeme (bzw. Datenquellen) wird von mehreren Faktoren gesteuert [HaLR03]:

fehlende Koordination zwischen den Unternehmenseinheiten, unterschiedliche Anpassungsgeschwindigkeit an neue Technologien, Fusionen und Aquisitionen von Unternehmen, geographische Entfernung der kollaborierenden Gruppen, etc.

Aus der Verteilung ergibt sich eine gewisse Autonomie der Organisationen bzw. der kooperierenden Teilsysteme. Unter Autonomie versteht man den Grad zu dem ein System (insbesondere DBMS) unabhängig von anderen Systemen betrieben wird. Dies bezieht sich auf Kontrolle, nicht auf Daten, da ein Mindestmaß an Kooperationsbereitschaft innerhalb einer Organisation vorausgesetzt wird. Nach [ÖzVa99] unterscheidet man zwischen drei Autonomieklassen:

- Designautonomie
  Bezieht sich auf die Freiheit des lokalen DBMS bezüglich des Datenmodells (relational, XML, usw.) und des Schemas (z.B. Benennung, Grad der Abdeckung, Grad der Normalisierung). Die Freiheit, dies jederzeit ändern zu können, ist im Hinblick auf Informatiosintegration besonders problematisch.
- Kommunikationsautonomie
  Lokale DBMS können jederzeit aus dem integrierten System ein- und austreten, entscheiden wie, wann und mit welchen Systemen kommuniziert wird und welcher Teil der Datenmenge bzw. Anfragemöglichkeiten (Prädikate, Sortierung, usw.) zur Verfügung gestellt wird.
- Ausführungsautonomie
  Bezieht sich auf die Freiheit bzgl. der Wahl der Scheduling- und Optimierungsstrategie, des Transaktionsmanagements, etc. Beispielsweise kann ein DBMS lokale und externe Anfragen unterschiedlich behandeln, *golden customers* bevorzugt behandeln, Antwortzeiten garantieren, etc.

Diese Gestaltungsfreiheit der einzelnen Systeme hat unterschiedliche lokale Entscheidungen zur Folge. Da jede dieser Quellen ursprünglich in einem eigenen Kontext aufgesetzt und genutzt worden ist, sind diese Datenquellen somit unweigerlich heterogen, und zwar in technischer, logischer und semantischer Hinsicht.

Die technische Heterogenität ergibt sich aus der Kommunikationsautonomie – Datenquellen können z.B. über proprietäre Programmierschnittstellen, standardisierte Anfragesprachen (z.B. SQL, XPath) oder eine Webschnittstelle angesprochen werden. Neben dieser technischen Vielfalt unterscheiden sich die Datenquellen auch im verwendeten Datenmodell. So sind heute neben den marktbeherrschenden relationalen Datenbanksystemen immer noch unternehmenskritische Altsysteme im Einsatz, die oft proprietäre hierarchische oder Netzwerk-Datenmodelle verwenden. Auch innerhalb eines Datenmodells können sich thematisch verwandte Schemata unterscheiden. Strukturelle und semantische Unterschiede stellen eine große Herausforderung dar, wenn es darum geht, Daten aus verschiedenen Quellen zusammenzuführen.

Erst die Kombination von Datenquellen schöpft das volle Potenzial der einzelnen Quellen aus und schafft Mehrwert für Unternehmen. Die oben genannten Autonomiearten und die sich daraus ergebenden Heterogenitäten stellen hohe Anforderungen an Informationsintegrationssysteme. Heterogenitäten zu überbrücken und eine einheitliche, integrierte Sicht auf mehrere Datenquellen bereitzustellen, die für den jeweiligen Anwendungsfall relevant sind, ist die Kernaufgabe der Informationsintegration.

Die genannten Dimensionen eines Informationssystems (Verteilung, Autonomie, Heterogenität) müssen auch bei der unternehmensübergreifenden, weltweiten Informationsintegration berücksichtigt werden. Im Laufe der Jahre wurden von mehreren Organisationen Konzepte und Technologien zur Lösung dieser Probleme entwickelt. Einige werden in den folgenden Kapiteln vorgestellt, mit besonderem Augenmerk auf die Arten des Zugriffs auf Datenquellen und deren Auffinden.

3 Föderierte DBS: Mediator/Wrapper-Architektur

Eine klassische Möglichkeit, verteilte, heterogene Datenquellen zu integrieren, bietet die Mediator/Wrapper-Architektur. Ein Mediator-basiertes Informationssystem fasst mehrere Datenquellen durch virtuelle Integration zusammen, wobei die Daten nur in den Quellen gespeichert sind [Wiki1]. Gio Wiederhold definierte im Jahre 1992 das Kozept des Mediators [Wied92]:

> "A mediator is a software module that exploits encoded knowledge about certain sets or subsets of data to create information for a higher layer of applications."

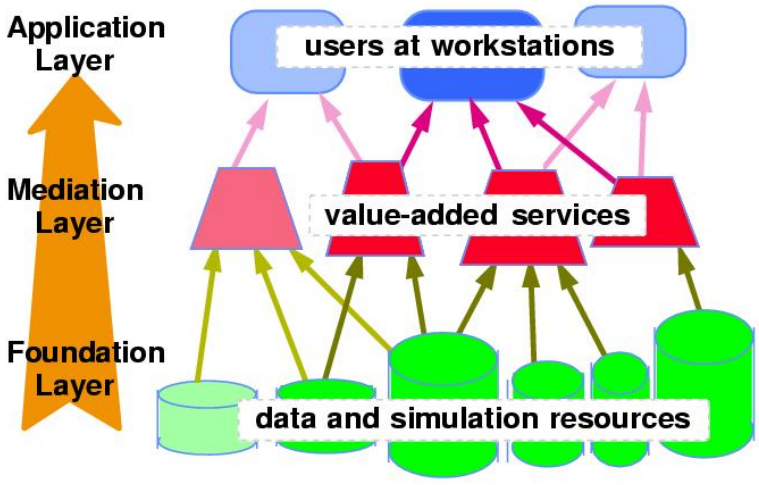

Abb.1: Mediation [Wied92]

Ein Mediator ist eine Middleware, eine vermittelnde Schicht zwischen den Anwendungen und den Datenquellen, die dem Benutzer durch Integration von Datenquellen Mehrwertdienste (sog. *value added services*) zur Verfügung stellt (siehe Abb. 1). Wenn verteilte Datenquellen kombiniert werden, können sie zur Entscheidungs- und Planungsunterstützung in höherwertigen Anwendungen beitragen. Es wäre z.B. nicht wirtschaftlich, eine eigenständige Datenbank für solche Entscheidungs-Unterstützungs-Systeme zu pflegen – vielmehr muss auf bestehende Datenbestände zurückgegriffen werden. Mediatoren stellen verschiedene Funktionen bereit, wodurch die Anwendungen die benötigten Informationen in geeigneter Form (syntaktisch und semantisch) erhalten [Köst03].

Mediatoren enthalten domänenspezifischen Code und werden von Domänen-Experten entwickelt. Um die gewünschte Funktionalität bereitzustellen, führen Mediatoren typischerweise die folgenden Schritte durch [Wied99]:

1. Suche, Zugriff und Extraktion von relevanten Daten aus mehreren heterogenen Ressourcen
2. Abstraktion und Transformation von Daten (mit Hilfe von Metadaten)
3. Integration der transformierten Daten
4. Reduzierung der integrierten Daten durch Aggregation, um die Informationsdichte zu erhöhen

3.1 Wrapper

Die Quellen eines Mediator-basierten Informationssystems werden durch so genannte Wrapper verpackt und mit dem Mediator verbunden. Primäre Aufgabe eines Wrappers (engl. *wrap up* = einpacken, einwickeln) ist die Kapselung der in einer Datenquelle gespeicherten Daten sowie ihrer Schnittstelle nach außen. Das Ziel dabei ist die Überwindung der Heterogenität einzelner Datenquellen. Wrapper sind jeweils spezialisiert auf eine Ausprägung autonomer, heterogener Quellen.

Abbildung 2 zeigt eine Möglichkeit, ein solches System zu konfigurieren. Hierbei sind unterschiedliche Szenarien denkbar. Zum Beispiel könnten Anwendungen direkt auf Wrapper zugreifen und Mediatoren als Quellen für andere Mediatoren dienen (stufenweiser Mehrwert). Ein Wrapper könnte benutzt werden, um auf mehr als eine Datenquelle zuzugreifen, usw. [Naum99].

Wrapper vereinheitlichen die Schnittstellen heterogener Datenquellen, indem sie von konkreten Technologien (SQL, HTML Formulare, HTTP, CORBA, etc.) abstrahieren. Sie reduzieren die Anzahl der Datenmodelle, mit denen das Informationsintegrationssystem umgehen muss und liefern ein kanonisches Schema. Sie sollten datenquellenunabhängig und wiederverwendbar sein.

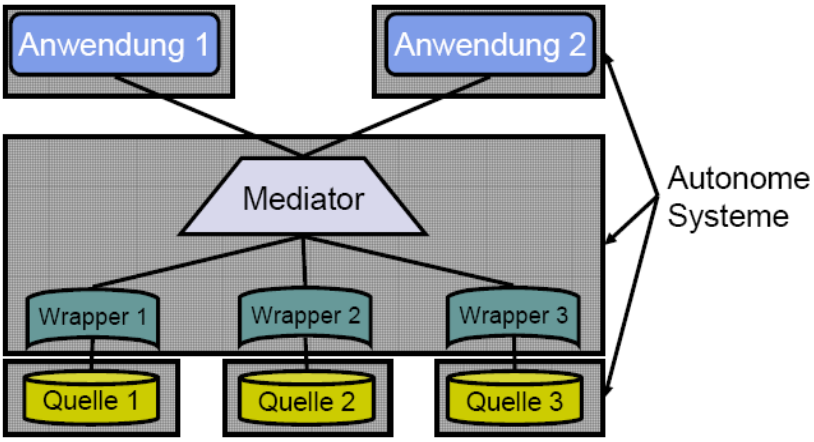

Abb.2: Mediator/Wrapper-Architektur

Am Beispiel des *XML Wrappers* für den *IBM DB2 Information Integrator* soll der Wrapper-Mechanismus veranschaulicht werden [Naum99]. Mit Hilfe des XML Wrappers wird eine relationale Abstraktion von XML Daten erzielt – Anwendungen sehen Tabellen und nicht XML. Die XML-Hierarchie wird gemäß Abbildungs-Strategie auf virtuelle Tabellen (Sichten und sog. *Nicknames*) abgebildet. Zur Extraktion der Attributwerte wird auf XPath zurückgegriffen. Diese virtuellen Tabellen können dann in normalen SQL-Anfragen verwendet werden, d.h. Anfragen an XML-Daten können mit ganzer SQL-Mächtigkeit formuliert werden.

Abbildung 3 zeigt eine Transformation (sog. *Shredding*) eines XML-Schemas in ein relationales Schema.

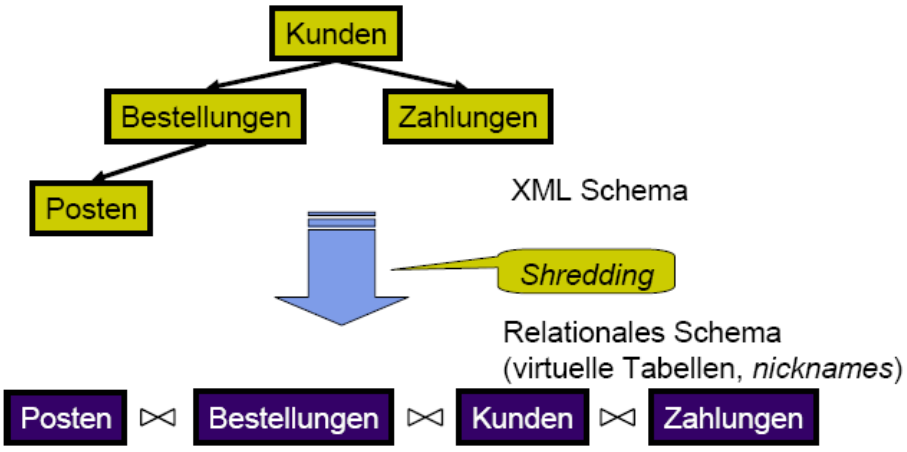

Abb.3: Shredding eines XML Schemas [Naum99]

Das resultierende SQL Schema (bzw. ein Teil davon – hier nur der Nickname kunden_NN) sieht so aus:

```
                                    ...
CREATE NICKNAME kunden_NN(
name            VARCHAR(48) OPTIONS(XPATH './name/text()'),
addresse        VARCHAR(48) OPTIONS(XPATH './address/text()'),
kunden_NN_ID VARCHAR(48) OPTIONS(PRIMARY_KEY 'YES'))
FOR SERVER xml_server
  OPTIONS(XPATH '//customer', FILE_PATH 'customers.xml');
                                    ...
```

Der *XML Wrapper Generator*, entwickelt am IBM Almaden Research Center, automatisiert diese Transformationsschritte und basiert auf dem Prototyp "Clio", der sich mit allgemeinen Schematransformationen beschäftigt.

Ein einfaches Beispiel für ein Mediator-Wrapper-basiertes Informationssystem ist eine Meta-Suchmaschine. Eine Meta-Suchmaschine ist eine Suchmaschine, die eine Suchanfrage an mehrere andere Suchmaschinen gleichzeitig weiterleitet, die Ergebnisse sammelt und aufbereitet. Dies beinhaltet die Eliminierung von Dubletten, die Bewertung der Ergebnisse, Aufstellung eines Ranking, etc. [Wiki2]. Der Zugriff auf einzelne Suchmaschinen kann über veröffentlichte Web Services (mehr dazu im Abschnitt 5) erfolgen oder mittels sog. *Screen Scraping* (Abschnitt 4).

3.2 Weitere Gliederung

Die eigentliche Mediation in virtuell integrierenden Systemen hat sich traditionell nur um die in Abschnitt 3 genannten Schritte 2 bis 4 (die Verarbeitung und Aggregation von Daten zu Informationen) gekümmert. Der erste Schritt, das Auffinden von und der Zugriff auf Datenquellen bestimmt aber im Wesentlichen die Flexibilität solcher Systeme und hat in den letzten Jahren stark an Bedeutung gewonnen. Dies hängt mit der beschriebenen Informationsexplosion zusammen.

Der folgende Abschnitt widmet sich dem Screen-Scraping-Verfahren zum Zugriff auf Webseiten als Datenquellen und erläutert das Prinzip dahinter. Anschließend wird im Abschnitt 4.1 eine immer schneller wachsende Kategorie der Datenquellen im Web vorgestellt und im Hinblick auf Zugriff und Discovery näher untersucht.

Technologien wie Web und Grid Services werden heutzutage verwendet, erweitert und an die Problematik der Verteilung, Heterogenität und Autonomie der Datenquellen angepasst. Dies wurde als notwendig erkannt, um der erwünschten Flexibilität beim Auffinden von und Zugriff auf Quellen Rechnung zu tragen. Diese Ansätze werden in den Abschnitten 5 und 6 vorgestellt.

Eine Zusammenfassung mit einer Übersicht über aktuelle Forschungs-schwerpunkte schließt diese Ausarbeitung ab.

4 Erschließung von Web-Datenquellen

Normalerweise tauschen Programme Datenstrukturen in maschinenverständlicher Form aus. Solche Datenformate sind strukturiert, leicht zu parsen, kompakt und frei von Redundanz. Die für Menschen konzipierten Ausgabeformate sind im Gegenteil dazu meist unstrukturiert, mit redundanten Bezeichnern und Formatierungselementen versehen. Solche Formate sind für automatische Interpretation ungeeignet. Unter *Screen Scraping* (auch: *Data Scraping, Web Scraping, HTML Scraping*, etc.) versteht man die Analyse von Bildschirminhalten (z.B. HTML-Webseiten) per Programmcode zwecks Extraktion von Daten.

Oft wird Screen Scraping auf die Extraktion von Daten aus Webseiten reduziert; die Discovery-Phase, die vor der Extraktionsphase erfolgen muss, ist aber häufig die schwierigere Aufgabe [Will06]. Das Auffinden von relevanten Datenquellen besteht in diesem Fall aus dem Navigieren zu den Webseiten, die die gewünschten Daten enthalten. Dieser Schritt kann so einfach sein wie ein *HTTP-GET-Request* einer URL. Andererseits kann es erforderlich sein, zuerst ein Login auszuführen, eine Reihe von Webseiten zu besuchen, um notwendige Cookies zu bekommen, ein *HTTP-POST-Request* auf einem Eingabe- oder Suchformular auszuführen, die Ergebnisliste zu durchsuchen und schließlich jedem einzelnen Link zu einer Detailanzeige des Ergebnisses zu folgen.

Als Methoden zum Zugriff (in diesem Fall also die Extraktion der Daten aus HTML-Dokumenten) eignen sich besonders reguläre Ausdrücke. Sie erlauben ein Maß an Unschärfe bei der Extraktion und bieten dadurch eine gewisse Robustheit gegenüber geringen Formatierungsänderungen des Präsentationsformats. Die meisten modernen Programmiersprachen unterstützen reguläre Ausdrücke. Der hohe Komplexitätsgrad von bereits einfachen regulären Ausdrücken und deren schwerfällige Wartung sind ein inhärenter Nachteil dieser Technik.

Screen Scraping-Software[1] abstrahiert von technologischen Details wie HTTP-Navigation, Cookies, regulären Ausdrücken, usw. Sie wirkt also wie ein Wrapper um eine Webseite. Wenn jedoch auf eine große Menge von Webseiten zugegriffen werden soll, müssen unterstützende Ansätze herangezogen werden, um Heterogenitäten zu überbrücken. Im Falle von unstrukturierten Daten bieten reguläre Ausdrücke allein keine ausreichende Flexibilität. Vielmehr müssen Ontologien und Methoden der künstlichen Intelligenz benutzt werden, um den Inhalt der Webseite semantisch zu analysieren und die relevanten Inhalte zu erfassen. Zum Beispiel könnte beim Parsen von thematisch verwandten Web-Quellen eine Ontologie benutzt werden, um der potenziell großen Begriffsvielfalt Herr zu werden[2].

1 Screen-Scraper, www.screen-scraper.com

2 So könnte z.B. der Begriff "Anzahl der Schlafzimmer" in verschiedenen Zeitungsannoncen unterschiedlich repräsentiert sein – eine Ontologie löst solche semantischen Heterogenitäten idealerweise automatisch auf.

Ein Beispiel für die Anwendung des Screen Scraping in einer einfachen Form ist das *LastMinuteTravel*-Projekt. Es handelt sich dabei um eine Web-Applikation, die mittels Screen Scraping heterogene Web-Datenquellen integriert. Ziel des Projektes *LastMinuteTravel* ist es, durch die Verknüpfung unterschiedlich strukturierter Internet-Datenbanken zum Thema Last-Minute-Travel, ein Tool zu entwickeln, das aktuelle Flugangebote zusammenfasst und übersichtlich darstellt [LastM06]). Dazu wird eine spezifizierte Anfrage an die Server-Schnittstelle der jeweiligen Webseite geschickt, aus der Daten entnommen werden sollen. Im Programm LastMinuteTravel wurde diese Abfrage über die HTML-Methode *GET* realisiert, die über die Internet-Adresse (URL) die jeweiligen vom Nutzer eingegebenen Abfragedaten (wie Reisezielort, Datum und Preis) und ggf. zusätzliche Attribute als Parameter an den Server schickt. Die Resultate der Anfrage werden dann über ein automatisiertes Verfahren aus der HTML-Seite extrahiert und an den Mediator weitergeleitet. Zur Extraktion werden z.B. die Tabellenstruktur und die jeweiligen Spalten ausgelesen, in der die gesuchten Daten enthalten sind.

Diese Methode des Zugriffs auf Daten ist inhärent fehleranfällig und inflexibel, da Änderungen an der Webseite die Extraktion erschweren bzw. unmöglich machen können [DNF]. Screen Scraping ist in bestimmten Fällen dennoch eine sinnvolle und manchmal auch die einzige Methode zum Zugriff auf Datenquellen [Wiki3]:

- als Schnittstelle zu einem Legacy System, das keinen anderen Mechanismus anbietet, der zur aktuellen Hardware kompatibel wäre
- als Schnittstelle zu einem System, das keine komfortable API zum Zugriff bietet

4.1 Hidden-Web

Unter dem *World Wide Web* (WWW) wird meist “nur” das *Surface Web* verstanden – die sichtbaren Webseiten, die von Suchmaschinen mit geeigneten Indexierungs- und Suchverfahren erschlossen werden. Traditionelle Suchmaschinen benutzen *Crawler* um die statischen Webseiten zu indexieren. Ein *Crawler* ist ein Softwareagent, der das WWW systematisch nach bestimmten Elementen durchsucht und rekursiv der Linkstruktur folgt [Wiki4]. Der nicht-statische Teil des WWW, der dynamisch aus Datenbankinhalten generiert wird, ist aber bei weitem größer. Dieses sog. *Deep-Web* (auch *Hidden-Web*, das versteckte Web) umfasst sowohl Datenbanken als auch nicht-textuelle Dateien, wie z.B. Multimedia, Software, PDF (Google erfasst mittlerweile auch PDF und andere Formate), die über das Web erreichbar sind [Alba].

In einer im Jahre 2000 durchgeführten Studie [BP] ergaben sich interessante Erkenntnisse:

- es existieren mehr als 200 000 Deep-Web-Seiten
- das Deep-Web enthält ca. 7500 Terabyte Daten, im Vergleich zu den 19 Terabyte im Surface-Web

- das Deep-Web ist eine der am schnellsten wachsenden Kategorien neuer Datenquellen im Internet
- solche versteckten Webseiten haben thematisch oft schmaleren, aber tieferen Inhalt
- und oft verlässlichere Informationen [BP]

4.1.1 Auffinden von Hidden-Web-Quellen

Das Auffinden von solchen versteckten Datenquellen lässt sich auf das Auffinden von Webseiten, die als Einstiegspunkt ins Hidden-Web dienen, reduzieren. Dies sind typischerweise Seiten mit HTML-Formularen. Bei der Suche nach solchen potenziellen Einstiegspunkten lassen sich schrittweise folgende hilfreiche Einschränkungen identifizieren [BC04]:

- Textuelle Formulare: Im Formular muss mindestens ein Textfeld als Eingabefeld vorhanden sein, nicht nur Schaltflächen, Listen, etc.
- Anfrageformulare: Es darf sich nicht um Login-Seiten handeln; es müssen Formulare sein, die Anfragen entgegen nehmen und Informationen zurückliefern.
- Hidden-Web-Formulare können beliebig komplex aufgebaut sein, mit mehreren Schaltflächen, Eingabefeldern, etc. Das große Problem liegt dabei in der automatischen Erfassung der Funktionalität des Anfrageformulars, das für den menschlichen Nutzer konzipiert ist. In [Webe00] werden deshalb Metadaten vorgeschlagen, mit denen Anbieter diese Funktionalität in maschinenverständlicher Art und Weise beschreiben können. Wenn solche Metadaten nicht vorliegen, erscheint es angebracht, sich auf einfache Formulare mit einem Textfeld zu beschränken.

Die Aufgabe, die es hierbei zu lösen gilt, ist das automatische Finden und Erkennen von solchen Formularen. Die manuelle Methode skaliert nicht und sollte im Hinblick auf die starke Expansion solcher Datenquellen nur als letzte Möglichkeit in Betracht gezogen werden.

Das automatische Auffinden von Formularen lässt sich z.B. mittels Google-Suche nach einem bestimmten Thema initialisieren. Auf die so gewonnene Ergebnismenge kann man lokales *Crawling* durchführen, um Formulare zu entdecken. Um in dieser Menge von Formularen potentielle Hidden-Web-Quellen zu entdecken, können folgende Prämissen hilfreich sein [BC04]:

- Ergebnisse sinnloser Eingaben in einem Hidden-Web-Formular sind immer gleich groß: so liefert eine zufällige Zeichenfolge (Phantasiewort) als Anfrage immer die gleiche Ergebnissseite als Resultat (z.B. eine Seite mit dem Text "0 hits found matching your search").
- Ergebnisse *sinnvoller* Eingaben sind immer größer als Ergebnisse sinnloser Eingaben (Größe in Byte gemessen).

Ob eine Eingabe sinnvoll ist oder nicht, lässt sich nur im Rahmen eines konkreten Anwendungsfalls entscheiden. So wäre z.B. bei der Suche nach Hidden-Web-Quellen zum Thema "Haustiere" eine Eingabe "Katzen" als sinnvoll zu bezeichnen.

Von diesen Feststellungen ausgehend kann man eine Reihe von Anfragen an die Webseite schicken und die Ergebnisse auf Größenunterschiede untersuchen.

Ob derartige Heuristiken nutzbringend eingesetzt werden können, ist Gegenstand aktueller Forschung. Um die Qualität automatischer Erkennung und Indexierung zu erhöhen erscheint es in dieser Hinsicht hilfreich, Web-Quellen mit maschinenverständlichen Metadaten anzureichern. Neben solchen Ansätzen gibt es auch zahlreiche Hidden-Web-Kataloge, die Web-Datenquellen manuell in Hierarchien klassifizieren (z.B. die Yahoo! Directory[3]).

Google Sitemaps[4] ist ein Experiment zum Crawlen von Webseiten. Um die Arbeit des Crawlers zu ergänzen, macht Google daher den Vorschlag, dass Webmaster selbst eine Liste mit den URLs aller Dateien bereitstellen, die der Crawler beim nächsten Besuch abrufen soll. Für die automatisierte Erstellung dieser "Google Sitemap" stellt Google auch eine kostenlose Software (*Sitemap Generator*) zur Verfügung, die beispielsweise die URL-Liste anhand der (Apache) Access-Logs erstellen kann. Dies ist insbesondere sinnvoll, wenn Websiten dynamischen Inhalt enthalten bzw. Seiten, die nicht so leicht durch das Verfolgen von Links gefunden werden können.

4.1.2 Klassifikation von und Zugriff auf Hidden-Web-Quellen

Wenn solche Hidden-Web-Quellen gefunden wurden, ist zunächst deren Klassifikation durchzuführen. Dies ist notwendig, um später deren Relevanz in einem Anwendungsfall feststellen zu können. Unter Klassifikation versteht man die Zuordnung von Quellen zu Kategorien bzw. Subkategorien, die hierarchisch angeordnet sind. Biespiele für die manuelle Klassifikation sind zahlreiche Yahoo-ähnliche Kataloge von Web-Datenbanken[5]. Während diese Art der Klassifikation sehr präzise ist, leidet sie, wie alle manuellen Methoden, unter der fehlenden Skalierbarkeit. Andererseits können auf diese Weise u.a. unerwünschte Hidden-Web-Quellen[6] leicht erkannt und bei der Indexierung ignoriert werden.

In [IpGS01] wird eine Methode der automatischen Klassifikation von Web-Datenbanken in Themen-Hierarchien vorgestellt, unter Benutzung von maschinellem Lernen und ausgewählten Datenbankanfragen. Das sog. *Query Probing* stellt eine neuartige und effiziente Art und Weise dar, um Web-Datenbanken zu klassifizieren, ohne die eigentlichen Dokumente aus diesen Quellen zu analysieren. Der Algorithmus macht sich allein die Anzahl der Dokumente in den Ergebnissen auf ausgewählte Anfragen zunutze und trifft dann eine Entscheidung bzgl. der Klassifikation der Web-Quelle. Dies macht den Ansatz besonders effizient und skalierbar.

---

3 http://dir.yahoo.com

4 http://www.google.com/webmasters/sitemaps/docs/de/about.html

5 Beispiele: InvisibleWeb, www.invisibleweb.com
Search Engine Guide, www.searchengineguide.com – Stand 9.2.2006

6 Diskussionsforen könnten im Gegensatz zu Datenbanken mit wissenschaftlichen Artikeln als weniger zuverlässige Quellen klassifiziert werden

Der Zugriff auf Hidden-Web-Quellen erfolgt über Eingabemasken bzw. Suchformulare. Solche Quellen verwenden aber meist unterschiedliche Anfragesprachen. Die Menge der unterstützten Operatoren ("AND", "OR", "PHRASE", etc.) und deren Semantik können auf jeder Hidden-Web-Seite unterschiedlich sein. Beispielsweise kennt IMDB.com den Operator "AND" bei der Suche nicht. Eine Anfrage wie "Casablanca and Bogart" würde deshalb nicht das gewünschte Ergebnis liefern – das Wort "and" wird dabei als ein normaler Suchbegriff interpretiert. Solche Anomalien gilt es zu entdecken. Das Problem der Schnittstellen-Heterogenität muss also beim Zugriff auf solche Datenquellen angegangen werden. Eine Möglichkeit, die Syntax der Anfragesprache zu erlernen, ist auch hier das maschinelle Lernen. Ein derartiger Ansatz wird in [BC04] vorgestellt. Dabei werden nach bestimmten Schablonen Testanfragen verschickt und Ergebnisse untersucht, mit dem Ziel, unterstützte Operatoren automatisch zu erkennen.

5 Service-Orientierung und Web Services

Ein verteiltes System benötigt Dienste, die die Kommunikation zwischen den Komponenten ermöglichen. Idealerweise auf offenen Standards aufbauend, bietet eine derartige Middleware die Möglichkeit entfernter Methodenaufrufe bzw. des Nachrichtenaustausches zwischen den Komponenten. Im Zeitalter von E-Business, selbstanpassbaren und mobilen Systemen gewinnen flexible, dynamische Architekturen immer mehr an Bedeutung. Diese sollten zumindest in zwei Dimensionen anpassungsfähig sein [BHT+03]:

- veränderbare Anforderungen, wie z.B. der Bedarf nach neuen Diensten oder Funktionalitäten, können flexible Kollaborationen zwischen den Teilnehmern voraussetzen
- wandelnder Kontext, z.B. fehlerhafte (oder zu langsame) Kommunikations-kanäle oder Hardwareausfälle können (soweit möglich) zur Ersetzung von nicht erreichbaren Komponenten führen

Service-orientierte Architekturen (SOA) sind eine wichtige und prominente Art von dynamischen Softwarearchitekturen, die zur Laufzeit rekonfiguriert werden können. Solche Systeme können auf bestimmte Ereignisse reagieren und ihre Komponenten reorganisieren. Diese Dynamik wird durch die drei Rollen in einer SOA (Abb. 4) realisiert. Die zentrale Komponente darin ist der Service. Services sind abstrakte Dienste, die von einer Person oder Organisation angeboten werden. Diese sind durch eine Dienstbeschreibung spezifiziert und über genau definierte Nachrichten aufrufbar. Der Dienstanbieter betreibt und stellt einen Dienst (z.B. eine Datenquelle) zur Verfügung. Klienten nutzten diesen Dienst bei Bedarf.

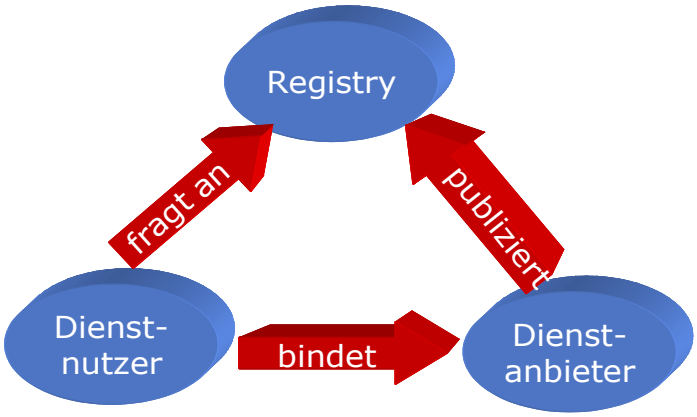

Abb.4: Rollen in einer Service-orientierten Architektur

Da sich der Dienstanbieter und Dienstnutzer in einer dynamischen Umgebung zu Beginn der Interaktion nicht kennen, ist eine Vermittlungsstelle (Registry) notwendig. Dienstanbieter publizieren Dienste bei Vermittlungsstellen, die diese Dienstbeschreibungen speichern und auf Anfrage einem Nutzer zustellen. Sobald der Klient eine Dienstbeschreibung empfängt, die seinen Anforderungen genügt, kann er mit Hilfe dieser Beschreibung mit dem Dienstanbieter interagieren. Der Kernaspekt dabei ist die Entkopplung der einzelnen Komponenten voneinander.

Web Services wurden im Rahmen dieses Paradigmas entwickelt. Die technologische Umsetzung dieses Szenarios wird durch standardisierte Protokolle und Spezifikationen realisiert. Der Zugriff auf Web Services wird durch SOAP (*Simple Object Access Protocol*) beschrieben. Dienstbeschreibungen werden in WSDL-Dokumenten (*Web Service Description Language*) festgehalten und in UDDI-Servern (*Universal Description, Discovery, and Integration*) veröffentlicht.

RPC-Mechanismen (*Remote Procedure Call*, dt.: entfernter Methodenaufruf) wie RMI, COM oder CORBA werden von den Firewalls meist blockiert [Merz05]. SOAP ist ein XML-basiertes Protokoll, das normalerweise auf HTTP (oder SMTP) aufsetzt. Sowohl dessen weite Verbreitung als auch die Fähigkeit, Firewalls zu überbrücken[7], haben zur Wahl von HTTP als Kommunikationsbasis beigetragen [Fand05]. Interoperabilität und Portabilität sind die großen Vorteile, die sich daraus ergeben. Web Services haben mittlerweile eine recht weite Verbreitung gefunden, insbesondere in B2B-Anwendungen.

Obwohl sich die Daten innerhalb von Datenquellen ändern können, ist virtuelle Datenintegration eine form statischer Integration. Ein Szenario vollständig dynamischer Integration schließt die Existenz transienter Datenquellen ein, die verteilt über mehrere Organisationen über ein Netzwerk miteinander verbunden sind und jeder Zeit aus dem System austreten bzw. in das System eintreten können. An dieser Stelle setzten Web Services an, indem sie die notwendige Infrastruktur für organisationsübergreifende Integration zur Verfügung stellen. UDDI hat aber keine

7 Firewalls blockieren den Port 80 (HTTP) normalerweise nicht

weite Verbreitung gefunden – die Dynamik von Web Services ist in der Realität relativ eingeschränkt. Außerdem muss der Zugriff auf heterogene Datenquellen vereinheitlicht werden, will man sich dem idealen Szenario der vollständig dynamischen Integration von Datenquellen annähern. Dieser Problematik widmet sich u.a. Grid Computing und insbesondere das OGSA-DAI-Projekt, das im Abschnitt 6.2 näher vorgestellt wird.

Als Beispiel für einen Web Service im Kontext der Datenintegration sei an dieser Stelle *MetaMatrix Dimension* genannt [MetaM06]. Mit Hilfe von deklarativen und visuellen Werkzeugen können standardisierte Web Services generiert werden, die als Wrapper um relationale Datenquellen fungieren. Die Datenquelle wird somit als Web Service einem breiten Spektrum an Applikationen zugänglich gemacht. MetaMatrx-Web-Services liefern XML-Datenstrukturen als Ergebnisse von Anfragen auf relationale Datenquellen. Eine automatische Abbildung von SQL nach XML wird jedoch nicht angeboten – die Schemata müssen manuell analysiert, Konflikte aufgelöst und Abbildungen gefunden werden.

5.1 Ein Screen-Scraping-Wrapper als Web Service

Am Beispiel eines *.NET* Web Service, der den Inhalt einer Webseite parst, soll die Möglichkeit, Web Services als Screen-Scraping-Wrapper zu verwenden, illustriert werden[8].

Solche Web Services werden im .NET-Framework deklarativ mittels WSDL-Dokumenten spezifiziert und erzeugt. In diesen Dienstbeschreibungen werden sowohl Eingabeparameter als auch die Daten, die aus der Webseite geparst werden sollen, beschrieben. Die Spezifikation der Ausgabe ist die Stelle, an der mittels regulären Ausdrücken beschrieben wird, was genau geparst werden soll.

Eingabeparameter können an den Web Server übergeben werden, die den Inhalt der erzeugten Webseite beeinflussen. Rückgabedaten werden in einer Reihe von XML-Elementen im WSDL-Dokument spezifiziert. Das Schlüsselelement dabei ist *<match>*, das einen .NET-regulären Ausdruck enthält.

Am Beispiel der folgenden HTML-Seite

```
<HTML>
  <HEAD>
      <TITLE>Sample Title</TITLE>
  </HEAD>
  <BODY>
      <H1>Some Heading Text</H1>
  </BODY>
</HTML>
```

8 http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconBuildingASPNETWebServices.asp

sei das Vorgehen demonstriert. Das folgende Codestück, insbesondere die zwei <match>-Elemente und deren *pattern*-Attribute, gibt an, dass Inhalte der HTML-Elemente <H1> und <TITLE> geparst werden sollen. Dieser ist Teil eines umfassenderen WSDL-Dokuments, das einen Web Service beschreibt, der den Inhalt einer Webseite parst.

```
[...]
<operation name="TestHeaders">
  <http:operation location="MatchServer.html"/>
     <input>
        <http:urlEncoded/>
     </input>
     <output>
       <text xmlns="http://microsoft.com/
                          wsdl/mime/textMatching/">
          <match name='Title'
                pattern='TITLE&gt;(.*?)&lt;'/>
          <match name='H1' pattern='H1&gt;(.*?)&lt;'/>
       </text>
     </output>
</operation>
[...]
```

Der Entwickler verfasst die WSDL-Dienstbeschreibung und legt insbesondere im "Operation"-Teil fest, welche regulären Ausdrücke beim Parsen verwendet werden. Das .NET-Framework erstellt aus dieser WSDL-Datei einen Web Service, der von beliebigen Anwendungen als Datenquelle genutzt werden kann. Das Framework bietet Werkzeuge an, die aus solchen WSDL-Dokumenten automatisch Proxy-Klassen erzeugen, welche von Klienten zum Zugriff auf diesen Web Service benutzt werden können.

Durch die Verwendung von Web Services können Vorteile der Service-Orientierung genutzt werden: der Zugriff auf Webseiten per Screen Scraping wird von der Anwendung entkoppelt und kann als Dienst gekapselt komfortabel wiederverwendet werden. Bei eventuellen Änderungen am Format der Webseite muss nur der Web Service ggf. angepasst werden; dessen Schnittstelle bleibt unverändert.

6 Grid Computing

Der Begriff *Grid* ist in Anlehnung an den Begriff *power grid* entstanden, was Energieversorgungsnetz bedeutet. Ein Grid soll ähnlich dem Energieversorgungs-netz funktionieren – bei Bedarf kann man bestimmte Ressourcen in Anspruch nehmen und sie nach Gebrauch wieder freigeben.

Unter Grid Computing versteht man die koordinierte Nutzung geographisch verteilter Ressourcen in sog. virtuellen Organisationen [FKT01]. Eine virtuelle Organisation ist ein Zusammenschluss von mehreren Parteien (Personen oder Organisationen), die gemeinsam an einem Projekt bzw. Problem arbeiten und auf heterogene Ressourcen über Verwaltungsdomänen hinweg zugreifen, um ein gemeinsames Ziel zu erreichen. Es geht also nicht darum, eine einzelne Applikation oder Anwendungsgruppe mit verteilten Prozessen, die auf einen Anwendungskontext zugeschnitten sind, zu realisieren. Stattdessen geht es um die Schaffung einer umfassenden Infrastruktur zur Nutzung verteilter Ressourcen.

Heterogenität und Autonomie von Ressourcen wird durch deren Virtualisierung überbrückt. Virtuelle Grid-Ressourcen abstrahieren von plattformspezifischen Konfigurationen, dem Aufenthaltsort der Ressource und deren konkreter Schnittstelle. Nach einmaligem Anmelden im System geschieht jede weitere Authentifizierung transparent im Hintergrund, vom Benutzer verborgen.

In einer solchen Grid-Umgebung ist es möglich, jeder virtuellen Organisation eine Rolle zuzordnen, sodass ihre Mitglieder bestimmte Zugriffsrechte erhalten. Es ist möglich, detailiert festzulegen, welche Ressourcen unter welchen Bedingungen von wem genutzt werden können. Rechte können delegiert werden, sodass sich die Benutzer nicht wiederholt authentifizieren müssen, um neue Ressourcen benutzen zu dürfen (*Single-Sign-On*).

Aufgrund der hohen Anzahl beteiligter Komponenten und der inhärenten Unzuverlässigkeit der Netzwerke und Rechner können Einzelausfälle im Grid nicht verhindert werden. Grid-Software muss in der Lage sein, flexibel auf Fehlersituationen jeder einzelnen Komponente zu reagieren. Es ist nicht schwer nachzuvollziehen, dass sich diese Anforderungen mit denen an Informations-integrationssysteme (IIS) überdecken. Aspekte wie das Auffinden von und der Zugriff auf Datenquellen in diesen dynamischen Umgebungen werden zu Schlüsselfaktoren zur Bestimmung der Flexibilität und Mächtigkeit eines IIS. Damit beschäftigen sich u.a. Standards, die in den kommenden Abschnitten genauer vorgestellt werden.

6.1 Erweiterung von Web Services

Web Services liefern die technologische Basis für Grid-Anwendungen, erfassen aber nicht deren gesamte Breite: interaktive Applikationen, Echtzeit-Anwendungen oder datenintensives Supercomputing lassen sich schlecht mit konventionellen zustandslosen Web Services realisieren. Grid-Anwendungen behandeln oft komplexe mathematische Probleme und müssen dafür zustandsbehaftet sein, um z.B. Zwischenergebnisse zu speichern. Dazu kommt, dass die Lebenszeit eines Web Service von seinem Server und nicht vom Zugriff der Clients abhängt. Dies hat zur Folge, dass sich alle Clients einen Zustand teilen [Schr05]. Sie sind außerdem nicht auf Sicherheitsmechanismen wie Single-Sign-On und Delegation von Rechten zugeschnitten. Des weiteren gibt es keine standardisierten Mechanismen für einen

einheitlichen Zugriff auf heterogene Datenquellen – Web-Service-Wrapper wie MetaMatrix (Abschnitt 5.1) sind sehr zahlreich und ohne einheitliche Schnittstellen und Zugriffsmechanismen versehen.

Grid Computing kann bis zu einem gewissen Grad als eine Erweiterung von Web Services, die die oben genannten Mängel von Web Services zu beseitigen versuchen, angesehen werden. Grundlagen für das Grid-Konzept schufen Ian Foster, Carl Kesselman und Steven Tuecke in den späten 90er Jahren. Weiterentwickelt wird es vom *Global Grid Forum* (GGF), einem internationalen offenen Standardisierungsgremium tausender Entwickler und anderer Beteiligter [Jabb05].

6.2 Datenverwaltungsysteme und Grid Computing – OGSA-DAI

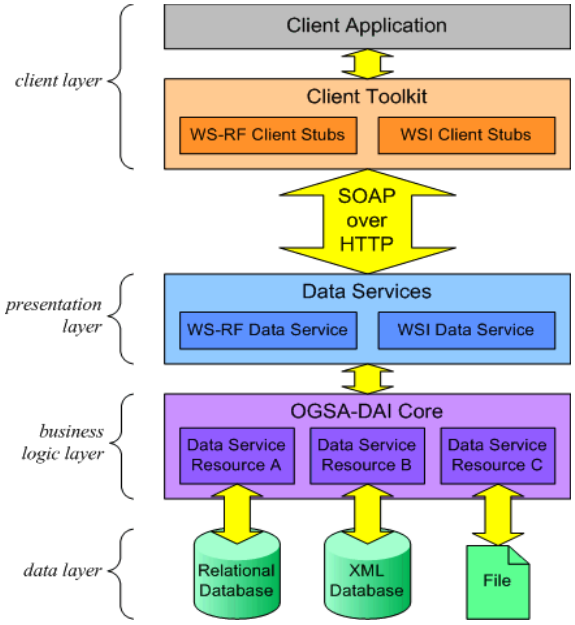

Abb.6: OGSA - DAI Architektur

Die sehr vielfältigen, heterogenen, geographisch verteilten Datenquellen einerseits und Grid-Protokolle und Dienste, die das dynamische Auffinden von und den Zugriff auf Dienste standardisieren, andererseits führten u.a. zum Projekt *Open Grid Service Architecture – Data Access and Integration* (OGSA-DAI) von der UK Database Task Force. Dies ist ein Zusammenschluss englischer Universitäten unter Beteiligung von staatlichen Wissenschaftseinrichtungen und Software-Unternehmen wie IBM [OGSA2]. Die *Data Access and Integration Services Working Group* (DAIS-WG) im GGF formuliert Standards für Datenquellenzugriff und Integration. OGSA-DAI basiert zu diesem Zeitpunkt auf der DAIS-Spezifikation aus dem Jahr 2003. Es ist beabsichtigt, dass OGSA-DAI eine Referenzimplementierung des finalen DAIS-Standards darstellt [OGSA1].

OGSA-DAI ist eine in Java entwickelte Middleware, die es ermöglicht, auf Datenquellen (XML-basiert, relational, usw.) standardisiert zuzugreifen. Dies geschieht mittels OGSA-DAI-Web Services. Datenquellen werden durch Ressourcen eines Grid-Service-Typs (*Data Service*) gekapselt und können nur über deren Operationen angesprochen werden. Das DAI-Framework überbrückt allerdings nicht die gesamte Heterogenität der Datenquellen: relationale Datenbanken und XML-Datenquellen müssen mit den zugehörigen Mechanismen (SQL bzw. XPath) angesprochen werden. Konkrete Treiber, der genaue Aufenthaltsort, benötigte Zugangsdaten und ähnliche Produktdetails werden vom Benutzer verborgen. Derzeit gibt es eine WSRF (*Web Service Resource Framework*)-basierte und eine WSI (*Web Services Interoperability*)-Variante des Frameworks.

Zur Zeit werden gängige relationalen DBMS sowie einige XML-Datenquellen (Apache Xindice, eXist) unterstützt. Auch ein Zugriff auf das lokale Dateisystem des Rechners, auf dem der Service läuft, ist möglich.

Die OGSA-DAI Architektur ist mehrschichtig aufgebaut (Abb.6). Der *Data Layer* setzt sich aus Data-Service-Ressourcen zusammen, die die unterliegenden Quellen kapseln. Die *Data Layer*-*Business Logic Layer*-Schnittstelle wird durch sog. Data-Source-Accessors realisiert, die die jeweiligen Zugriffsarten umsetzen (JDBC, XPath, Dateizugriff).

Der *Business Logic Layer* kapselt die Kernfunktionalität des OGSA-DAI Frameworks. Er bietet folgende Funktionalität:

- Ausführung von Perform-Dokumenten: alle Aufträge an die Ressource und Anfragen an die zugehörige Datenquelle erfolgen mit XML-basierten Perform-Dokumenten (siehe Abschnitt 6.3.3)
- Erstellung von Response-Dokumenten als Antwort auf ein Perform-Dokument
- Session Management: Zwischenergebnisse können gespeichert werden.
- Bereitstellung von Ressourceneigenschaften: diese werden genutzt, um an Metadaten oder den Status der Ressource zu kommen.

Die Präsentationsschicht hat so gut wie keine Aufgaben außer die verbundenen Ressourcen zu verwalten und zugänglich zu machen. Diese Schicht leitet alle Anfragen weiter und reicht Ergebnisse zurück. Eine eigenständige Operation ist *ListResources*, die die Namen der enthaltenen Ressourcen zurückgibt.

6.2.1 OGSA-DAI Grid Services

Es folgt ein Überblick über die verschiedenen Grid-Services im OGSA-DAI-Framework:

- *DAI Service Group Registry (DAISGR):* Dies ist ein Grid-Service zur Registrierung von OGSA-DAI-Grid-Services; durch das Anmelden eines Services an einer Registry wird dieser für Klienten sichtbar gemacht und diese können dann weitere Informationen über den Service bzw. die dahinter steckende Datenquelle abrufen.

- *Grid Data Service Factory* (GDSF): Eine GDSF repräsentiert genau eine Datenressource und ist in der Lage Grid-Data-Services zu erstellen, mit deren Hilfe Klienten mit der Datenressource interagieren können. Über die GDSF haben Klienten die Möglichkeit weitere Informationen über die Ressource abzufragen, um feststellen zu können, ob die jeweilige Ressource ihre Anforderungen erfüllen kann.
- *Grid Data Service* (GDS): Der GDS ist die zentrale Komponente im OGSA-DAI-Framework – sie unterstützt den Zugriff auf Datenquellen.

Um das Zusammenspiel der Komponenten zu verdeutlichen, wird im Folgenden ein typisches Datenzugriffs-Szenario im OGSA-DAI-Framework gezeigt (siehe Abb.7):

- eine Grid-Data-Service-Factory (GDSF) wird instantiert und bei der DAI-Service-Group-Registry (DAISGR) registriert
- Die GDSF repräsentiert hier eine Datenressource, z.B. eine Datenbank mit wissenschaftlichen Berichten
- wenn nun ein Klient auf die Datenbank zugreifen möchte, kann er entweder direkt mit der GDSF kommunizieren oder diese über die DAISGR identifizieren
- im nächsten Schritt fordert der Klient die GDSF auf, eine Grid-Data-Service (GDS) zu instanzieren
- der Klient interagiert mit der GDS mittels Perform-Dokumenten
- die GDS liefert Antworten mittels Response-Dokumenten
- zum Schluß kann der Klient die GDS explizit beenden oder sie von allein (Timeout) terminieren lassen

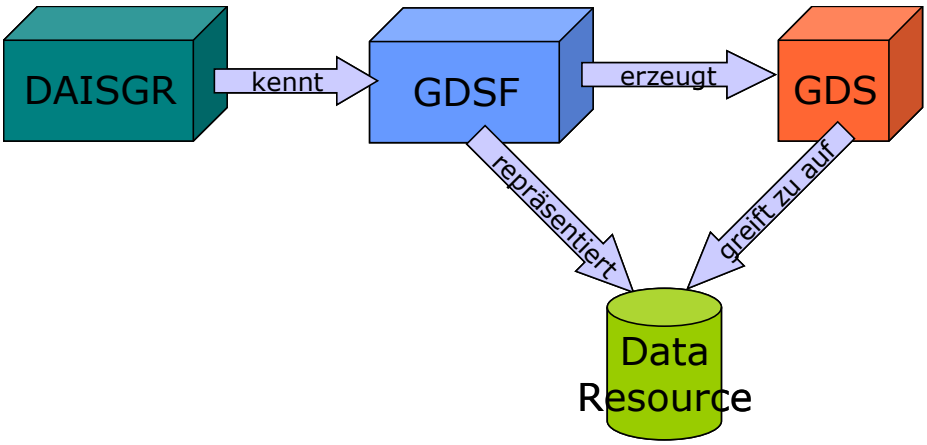

Abb. 7: Zusammenspiel zwischen den OGSA-DAI-Komponenten

6.2.2 OGSA-DAI Perform-Dokumente

Die Kommunikation mit Datenquellen ist nachrichtenbasiert und wird mit einer einzelnen Operation in der Schnittstelle des Data-Service umgesetzt. Zur Beschreibung der Aufträge dienen umfangreiche Perform-Dokumente (XML-basiert). Abb. 8 zeigt die Anfragebearbeitung im OGSA-DAI-Framework.

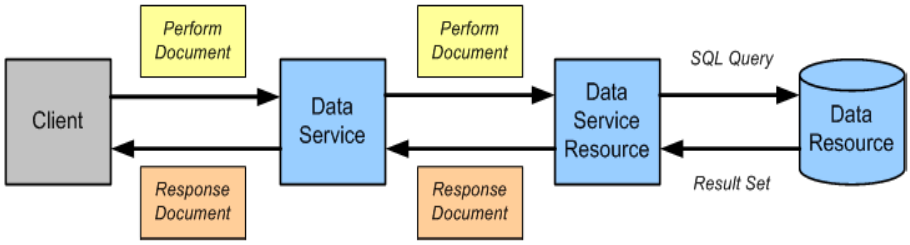

Abb. 8: Anfragebearbeitung im OGSA-DAI-Framework

Der Data-Service reicht das Dokument ungeprüft an die Ressource weiter, diese analysiert es und setzt die Wünsche des Benutzers bei korrekter Formulierung in Datenbankanfragen oder andere Aktionen um. Ergebnisse werden in einem Response-Dokument an den Benutzer des Dienstes zurückgeliefert. Perform-Dokumente bestehen im Wesentlichen aus Aktivitäten, die von konkreten Aktionen abstrahieren. Aktivitäten können z.B. sein:

- Anfrage an die Datenquelle: Lese- und Schreibzugriffe an die Datenbank, Aufruf einer *Stored Procedure*, Einfügen von größeren Datenmengen (*BulkLoad*) etc.
- Transformation von Daten: Anfrageergebnisse können mit XSLT in ein gewünschtes Format umgewandelt oder u.a. per GNU ZIP komprimiert werden.
- Übertragung von Ergebnissen (Delivery-Aktivität): mittels bestimmter Grid-Datendienste (GridFTP) können Daten an andere Ressourcen geschickt werden
- Benachrichtigung: Erlaubt die Erstellung von Nachrichten aufgrund von Ereignissen, etwa die Beendigung einer Aktivität.

Ein Perform-Dokument besteht aus einer Menge von Aktivitäten, die untereinander verknüpft werden können. Entscheidend dafür ist die Behandlung von Daten als Datenstrom. Jede Aktivität kann über benannte Input- und Output-Streams verfügen. Eine typische Aktivitätenfolge wäre eine SQL-Abfrage, die keinen Input benötigt, aber einen Strom von Daten erzeugt. Dieser wird durch eine Transformationsaktivität umgewandelt und danach eventuell von einer Delivery-Aktivität versendet oder im Response-Dokument an den Anfragesteller zurückgeliefert.

Die zwei größten Vorteile nachrichtenbasierter Kommunikation mittels einer einzigen Methode in der Schnittstelle sind:

- Erweiterbarkeit:
  Wenn sich das Verhalten von gekapselten Ressourcen ändert, muss die Schnittstelle nicht angepasst werden, da die Funktionalität nicht in der Schnittstelle sondern in der Nachricht liegt.

- Performanz:
  Die Zahl der Operationsaufrufe (und die damit verbundenen Verzögerungen) wird durch Angabe mehreren Aktivitäten in einer einzigen Nachricht deutlich reduziert.

6.2.3 Fehlende Integrationskonzepte in OGSA-DAI

OGSA-DAI abstrahiert von herstellerspezifischen Details einzelner Datenquellen. Zwischen den beiden verbreitetsten Datenquellen-Klassen, den relationalen und XML-Datenbanken, besteht allerdings eine Kluft, die nicht so einfach zu überbrücken ist. OGSA-DAI versucht dies nicht und stellt getrennte Zugriffsmechanismen zur Verfügung.

Durch das Konzept der temporären Sitzungen, sog. *Sessions*, lassen sich Kontexte bzw. Zustände über den Zeitraum mehrerer Anfragen aufrecht erhalten. Sie sind allein zur Speicherung beliebiger Daten fähig, ohne weitere angebotene Funktionalität. In Verbindung mit einer Delivery-Aktivität (Übertragung von Ergebnissen) ist es z.B. möglich, die Ergebnisse eines Auftrages in die Session einer anderen Ressource zu schicken. Eine spätere Anfrage könnte diese Daten mittels einer BulkLoad-Aktivität temporär in die dortige Datenbank einfügen und letztlich einen Join mit einer anderen Tabelle ausführen. Dieses Szenario zeigt zugleich die einzige Möglichkeit von OGSA-DAI, Daten aus mehreren Quellen zu integrieren. Echte datenbankübergreifende Operationen werden nicht angeboten [Jabb06].

Des weiteren handelt es sich bei dem Angebot von OGSA-DAI-ServiceGroup-Registry lediglich um statische Listen von Ressourcen, frei von jeder Suchmöglichkeit auf die zugehörigen Metadaten. Die Auswahl, welche Datenquelle die passende ist, bleibt dem Benutzer überlassen.

6.3 OGSA-DQP

OGSA-DQP (*Distributed Query Processing*) ist ein Grid-Projekt, das sich um Datenintegration und koordinierte, verteilte Anfragebearbeitung im OGSA-DAI-Framework kümmert [OGSA2]. Das Service-orientierte DQP-Framework stellt einen Ansatz dar, der

- Anfragen an mehrere OGSA-DAI- und andere Dienste im Grid unterstützt, wobei Datenzugriff und Datenanalyse angeboten werden;
- implizite Parallelisierung der Ausführung von Teilanfragen unterstützt;

• den Data-Services-Standard nutzt, um vom einheitlichen Zugriff auf heterogene Datenquellen und deren Metadaten zu profitieren.

Das DQP-Framework setzt sich aus zwei Diensten zusammen:

- *Grid Distributed Query Service* (GDQS):
  Der GDQS fungiert als Koordinator und Schnittstelle zum Benutzer. In der Initialisierungsphase müssen verfügbare Datenquellen (bzw. deren Metadaten) importiert werden. Dieser Grid-Dienst kümmert sich um die Kompilierung, Optimierung, Partitionierung und Durchführung von Anfragepläne. Der Koordinator ist selbst als ein OGSA-DAI-Data-Service implementiert und stellt daher standardisierte Schnittstellen zum Zugriff zur Verfügung.

- *Grid Query Evaluation Service* (GQES):
  Der GQES führt Teilanfragepläne aus, die ihm vom Koordinator nach der Kompilierung und Optimierung zugeordnet werden. Eine Menge vom QES-Diensten ist in einer Baumstruktur organisiert. Daten fließen dabei von den Blättern, die mit Grid-Data-Services interagieren, bis hin zur Wurzel, dem Koordinator.

Datenanalyse kann optional durch Web Services durchgeführt werden, bevor die Ergebnisse an den Benutzer weitergegeben werden. Dazu müssen bei der Initialisierung des Koordinators die entsprechenden WSDL-Dateien geladen werden. Anfragen an den Koordinator werden mittels standardisierten OGSA-DAI-Perform-Dokumenten geschickt.

OGSA-DQP fungiert somit als ein Mediator über den OGSA-DAI-Data-Services. Die Phase des Auffindens von relevanten Datenquellen muss aber zur Zeit immer noch vom Benutzer durchgeführt werden. Es fehlen also höherwertige Dienste wie semantische Suche und automatisches Auffinden von Quellen, Schemaintegration, Ontologien, usw.

6.4 Das DynaGrid-Projekt

Das DynaGrid-Projekt, zur Zeit aktiv entwickelt an der Universität Hamburg, versucht das OGSA-DAI-Framework um fehlende Funktionalitäten zu erweitern, die für eine automatische Integration heterogener, autonomer, verteilter Datenquellen im Grid unerlässlich sind. Ziel des DynaGrid-Ansatzes ist es, die relativ starren Ansätze des OGSA-DAI-Projektes aufzubrechen. In einer Grid-Umgebung soll ermöglicht werden, dass zur Laufzeit die zu einer Anfrage passenden, vorhandenen Datenquellen bestimmt und die Anfrage entsprechend gestellt wird. Um das zu erreichen, wird die Anfrage von den Strukturen der Datenquellen entkoppelt, die passenden Datenquellen automatisch von der Registry zurückgegeben und von den Datenquellen eine einheitliche Schnittstelle zur Verfügung gestellt. Die Integration von Anfrageergebnissen geschieht ebenfalls dynamisch und benutzerunabhängig.

Eine dynamisch erzeugte Antwortstruktur und eine interne Normalisierung der Begriffe schaffen die Voraussetzungen dafür. Folgende Komponenten mussten dazu entwickelt werden [Jabb06]:

- QueryEngine
  Da es im OGSA-DAI-Framework keinen Data-Service-übergreifenden Dienst gibt, der es ermöglichen würde, Anfragen über mehrere im Grid zugreifbare Quellen zu stellen, musste ein solcher Service implementiert werden.

- Registry
  Klienten haben die Möglichkeit die Servicedaten der DAISG-Registry abzufragen, um so an die registrierten Services zu gelangen. Diese Registry sollte für die Umsetzung der DynaGrid-Vision wiederverwendet und um bestimmte Abfragemöglichkeiten erweitert werden, so dass die QueryEngine genauer spezifizieren kann, welche Informationen die gesuchten Datenquellen enthalten.

- Wrapper
  Zusätzlich zur angebotenen Abstraktionsfähigkeit von Grid-Data-Services sollte es möglich sein, unabhängig von der konkreten Art der Datenquelle Anfragen stellen zu können. Des weiteren dürfen diese Anfragen auch nicht an das konkrete Schema der Datenquellen gebunden sein. Hierzu wurde ein Wrapper neu entwickelt.

Die entscheidende Folgerung aus dem dynamischen Ansatz des DynaGrid-Projektes ist das Fehlen eines fertigen globalen Schemas zum Zeitpunkt einer Anfrage eines Benutzers, wie es im Szenario der statischen Integration der Fall ist. Stattdessen ist das derzeitige Ziel die Umsetzung eines zweistufigen Systems: eine Registry-Komponente verfügt über ein jederzeit aktuelles wenn auch ungenaues Gesamtbild aller Datenquellen und kann zum Anfragezeitpunkt die Menge der möglicherweise relevanten Schemata stark eingrenzen. Nur diese werden von einer zweiten Analysekomponente, einem Schema-Matcher, daraufhin genauer untersucht. Die gefundenen Übereinstimmungen dienen schließlich der Anfragedurchführung.

Abbildung 9 zeigt die grobe Gesamtarchitektur des DynaGrid-Integrations-werkzeuges. Die QueryEngine ist die Kernkomponente des Systems, sie überwacht die Einzelschritte, trägt die Verantwortung und baut letztendlich aus den einzelnen Schemata der Datenquellen ein temporäres gemeinsames Schema auf und fügt die Daten aus den zahlreichen Anfragen zusammen.

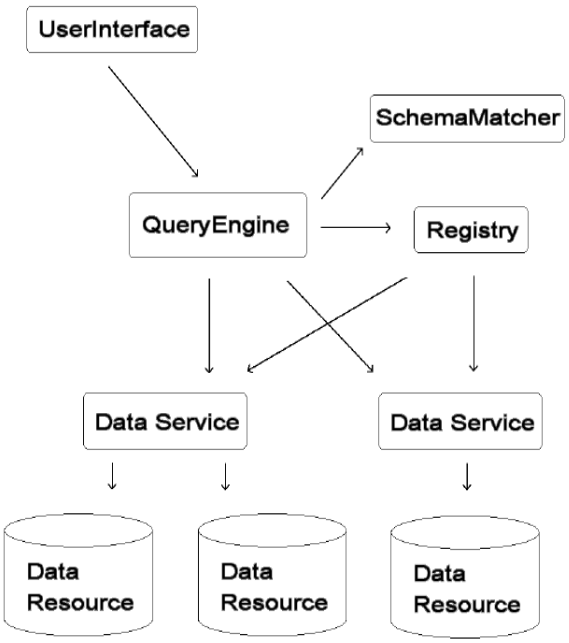

Abb. 9: DynaGrid-Architektur [Jabb06]

Die unterschiedliche Benennung von Relationen, Attributen und Elementen wird durch eine interne Normalisierung der benutzten Begriffe beseitigt. Hierfür werden sowohl bei der Anfrage als auch bei den Resultaten der Datenquellen Bezeichnungen bei einem Ontologie-Service eingereicht und ggf. durch einheitliche Begriffe ersetzt. Dieser Ontologie-Service kann zur Vereinfachung auf bestimmte Themengebiete eingeschränkt sein, um bessere Resultate zu liefern [Jabb06].

6.5 IBM WebSphere Information Integrator & Grid Wrapper

IBM verfolgt einen ähnlichen Weg, um das Potential der Ortstransparenz und des späten Bindens von Grid-Ressourcen in seinen bestehenden Produktfamilien einzubinden [LMD+05]. Die WebSphere Informationsintegrations-Technologien erlauben es, entfernte Datenquellen statisch zu föderieren. Grid Services genießen dagegen den Vorteil der Virtualisierung und abstrahieren vom konkreten Zugriff auf Datenquellen – allerdings müsste die Applikation die Föderationsmechanismen selbst implementieren (wie im DynaGrid-Projekt). IBM hat diese Lücke in ihrem Produktportfolio identifiziert und dazu die Entwicklung des *Grid Wrapper* vorgestellt.

Die in diesem Zusammenhang vorgestellte grobe Architektur des Grid Wrappers (von IBM alphaWorks) soll das Problem lösen (Abb. 10).

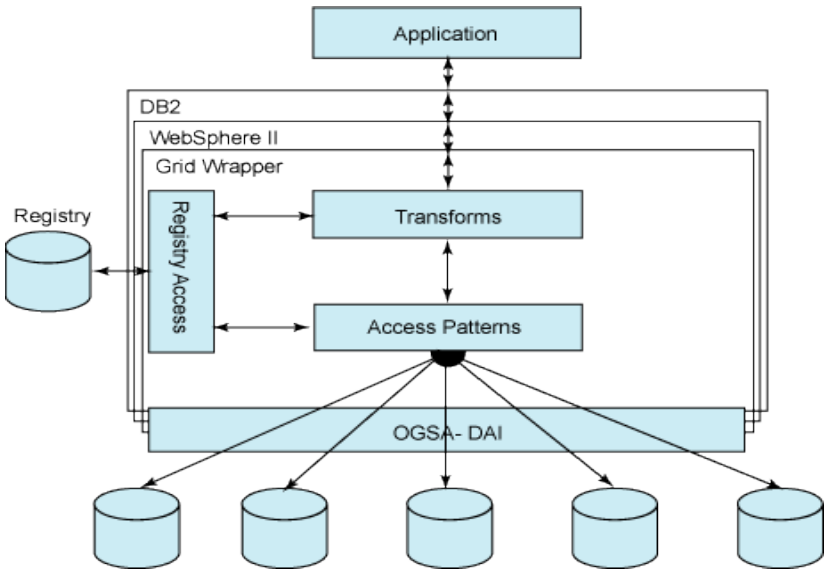

Abb. 10: Kernkomponenten des IBM Grid Wrappers

Dabei wurde drei Schlüsselkonzepte erarbeitet, um die Lücke zwischen WebSphere II und OGSA-DAI zu schließen:

- *Registry*: ähnlich wie bei DynaGrid wird die Registry benötigt, um zur Laufzeit relevante Datenquellen zu finden
- *Access Patterns*: kapseln die Semantik von Datenquellen und die Art, wie sie angesprochen werden müssen, um eine gegebene Anfrage zu bearbeiten
- *Transformationen*: abstrahieren von den Unterschieden zwischen Datenquellen, sodass Applikationen nur ein Modell verwenden können

Auf diese Weise entfernt der Grid Wrapper die Kopplung zwischen WebSphere II und den Datenquellen und ermöglicht deren spätes Binden. Gleichzeitig kann die Applikation den vollen Funktionsumfang von WebSphere zur Föderation nutzen. WebSphere wird somit zum Grid-Gateway für Applikationen.

7 Zusammenfassung und Ausblick

In dieser Ausarbeitung wurden einige traditionelle sowie neuere Architekturen und Ansätze zum Zugriff auf und Auffinden von Datenquellen betrachtet. Die weltweite Vernetzung erhöht die Zahl verfügbarer und potenziell relevanter Datenquellen für einen Anwendungsfall im beträchtlichen Ausmaß und verstärkt den Fokus eines Informationsintegrationssystems auf die Flexibilität und spätes Binden von Quellen. Die klassische Mediator-Wrapper-Architektur und Service-Orientierung als

konzeptuelle Grundlagen sind in fast allen aktuellen Forschungsbestrebungen im Feld der dynamischen Integration wiederzufinden. Das Grid-Computing-Paradigma und die Virtualisierung von Ressourcen sind nach Web Services der nächste Evolutionsschritt, der uns der Vision, in der sich der Benutzer nicht um das Auffinden oder Registrieren von Datenquellen kümmern muss, näher zu bringen verspricht.

Technologisch und konzeptuell gesehen gibt es auf diesem Gebiet noch einiges zu tun. Das OGSA-DAI-Projekt stellt nur die grundlegende Infrastruktur zum Zugriff auf Datenquellen im Grid zur Verfügung. Es stellt eine Referenzimplementierung vom OGSA-DAIS-Standard, das von einflußreichen Organisationen und Unternehmen aktiv unterstützt wird, dar. Um die höherwertigen, datenquellen-übergreifenden Aspekte im OGSA-Framework kümmert sich ein parallel laufendes, relativ junges Projekt – OGSA-DQP (OGSA-*Distributed Query Processing*). OGSA-DQP ist, ähnlich wie die DynaGrid-QueryEngine, ein Grid-Dienst, das eine Ebene über OGSA-DAI anzusiedeln ist. Er fungiert als eine Art Mediator, der Anfragen entgegen nimmt und sie an mehrere OGSA-DAI-Data-Services (als Wrapper) koordiniert verteilt. Das Auffinden von Datenquellen sowie die Konfliktauflösung beim Schema-Import müssen immer noch vom Benutzer bereitgestellt werden. Es wird also eine weitere Schicht über OGSA-DQP benötigt, um auch diese Aspekte zu automatisieren.

IBM hat in einer Studie [BGH+04] einige Ziele und Bemühungen identifiziert, aus denen Prototypen wie z.B. der vorgestellte *Grid Wrapper* resultieren, die zur Zeit getestet und weiterentwickelt und ggf. in IBM's Flaggschiff, den *WebSphere Information Integrator* einfließen werden. Forschungsfelder wie das automatische Auffinden von relevanten Datenquellen, die Berücksichtigung der Autonomie der teilnehmenden Datenquellen durch *Service-Level-Agreements* und standardisierte Sicherheitsmechanismen sind klare Schwerpunkte dieser Studie.

Dadurch, dass die Qualität der Ergebnisse nicht so hoch wie bei nicht vollständig automatisiert integrierenden Systemen einzuschätzen ist, sind auch mögliche Anwendungsgebiete für solche Systeme relativ stark eingeschränkt. Für den Umgang mit Integrationskonflikten bei den konkreten Ausprägungen der Daten gibt es noch kein endgültiges Verfahren. Ob sich diese Konflikte durch die Verwendung von Ontologien oder durch andere Verfahren lösen lassen, ist Bestandteil laufender Forschung.

8 Literatur

[Alba] University at Albany, New York, University Library: *The Deep Web;* Technical Report, 21. Dec. 2005 http://library.albany.edu/internet/deepweb.html

[BC04] A. Bergholz, B. Chidlovskii: *Learning Query Languages of Web Interfaces*, Proceedings of the 2004 ACM symposium on Applied computing

[BHT+03] L. Baresi, R. Heckel, S. Thöne, D. Varro: *Modeling and Validation of Service – Oriented Architectures: Application vs. Style;* ACM SIGSOFT Software Engineering Notes Volume 28, September 2003

[BMBF] Bundesministerium für Bildung und Forschung: *eScience Forum* Stand: 25. Mai 2006 *http://www.e-science-forum.de/de/60.php*

[BGH+04] S. Bourbonnais, V.M.Gogate, L.M. Haas, R.W. Horman, S. Malaika, I. Narang, V. Raman: *Towards an information infrastructure for the grid* IBM Systems Journal, Vol 43, No 4, 2004

[Dahl06] Mathias Dalheimer: *Introduction to Grid Computing*; Vorlesungsfolien: Kommunikationsplattformen für verteilte Applikationen; TU Kaiserslautern, 2006

[DNF] o.V.: *DotNetFramework Glossar* http://www.dotnetframework.de

[Fand05] Sören Fandrich: *Schema Integration and Query Processing in Dynamic Grid Services*; Diplomarbeit, 2005 Universität Hamburg

[FKT01] Ian Foster, Carl Kesselman, Steven Tuecke: *The Anatomy of the Grid - Enabling Scalable Virtual Organizations*; International J. Supercomputer Applications, 15(3), 2001 http://www.globus.org/alliance/publications/papers/anatomy.pdf

[HaLR03] Haas, Lin, Roth: *Data Integration Through Database Federation;* Technical Report, IBM Systems Journal, Vol. 4, 2003

[IpGS01] P. Ipeirotis, L. Gravano, M. Shami: *Probe, Count, and Classify - categorizing hidden-web databases*
Proc. ACM SIGMOD Conf., 2001.
http://qprober.cs.columbia.edu/publications/sigmod2001.pdf

[Jabb06] Christian Jabbusch: *Inhaltsbasierte Erschließung von Datenquellen in Grid – Umgebungen*; Diplomarbeit, Universität Hamburg, Januar 2006

[Köst03] Frank Köster: *Informationssysteme II*, Vorlesungsfolien Sommersemester 2003, Universität Oldenburg

[LastM06] *LastMinuteTravel – Projekt*
Stand: Juni 2006
http://www.wiwiss.fu-berlin.de/lenz/cebit/lastminute/

[LMD+05] Adrian Lee, James Magowan, Patrick Dantressangle, Fabian Bannwart: *Bridging the Integration Gap, Part 1 – Federating Grid Data*; IBM developerWorks Article, 2005

[LVD+03] P. Lyman, H. Varian, A. Dunn, A. Strygin, K. Swearingen: *How Much Information?*; 2003
http://www.sims.berkeley.edu/research/projects/how-much-info/

[MetaM06] MetaMatrix Dimension
Stand Juni 2006
http://www.metamatrix.com

[Naum99] Felix Naumann: *Informationsintegration; Mediator/Wrapper Architektur & Peer-Data-Management*;
Vorlesungsfolien, HU Berlin, 2004

[OGSA1] OGSA-DAI Homepage
Stand: Mai 2006
http://www.ogsadai.org.uk

[OGSA2] M. Nedim Alpdemir: *OGSA-DQP: Service-Based Distributed Query Processing on the Grid*;
Technical Report, University of Manchester, 2003
http://www.nesc.ac.uk/events/ahm2003/AHMCD/pdf/114.pdf

[ÖzVa99] M. Özsu, P. Valduirez: *Principles Of Distributed Database Systems;* Prentice Hall, 1999

[Schr05] Robert Schrader: *Standardisierungen in der Grid Community;* Seminar Semantic Grid, Universität Koblenz-Landau, WS04/05

[Wied92] Gio Wiederhold: *Mediators in the Architecture in Future Information Systems*, The IEEE Computer Magazine, March 1992

[Wiki1] *Wikipedia*, die freie Enzyklopädie: *Mediator-basiertes Informationssystem*, Stand: 30.05.2006 http://de.wikipedia.org/wiki/Mediator-basiertes_Informations system

[Wiki2] *Wikipedia*, die freie Enzyklopädie *Meta-Suchmaschine*, Stand: 30.05.2006 http://de.wikipedia.org/wiki/Metasuchmaschine

[Wiki3] Wikipedia, die freie Enzyklopädie S*creen Scraping*, Stand 30.05.2006 http://en.wikipedia.org/wiki/Screen_scraping

[Wiki4] Wikipedia, die freie Enzyklopädie *Web Crawler*, Stand 04.06.2006 http://en.wikipedia.org/wiki/Web_crawler

[Wied99] Gio Wiederhold: *Value-added Mediation in Large-Scale Information Systems*, Technical Report, Stanford University 1999

[Will06] Todd Wilson: *Data discovery vs. data extraction* (16.03.2006): http://blog.screen-scraper.com/2006/03/16/data-discovery -vs-data-extraction/

[Webe00] Gunnar Weber: *Integration von Datenbanken in Suchmaschinen bei unterschiedlichen Kooperationsgraden*; Technical Report, 2000, Universität Rostock