Thomas Wittek (3776263)

Diplomarbeit
im Fach Spezielle Wirtschaftsinformatik -
Informationsmanagement

Integration heterogener Inhalte in individualisierbaren Informationsdiensten

Eine prototypische Implementierung

Themensteller: Prof. Dr. Detlef Schoder

Vorgelegt in der Diplomprüfung
im Studiengang Wirtschaftsinformatik
der Wirtschafts- und Sozialwissenschaftlichen Fakultät
der Universität zu Köln

Köln, 2007

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AST	–	Abstrakter Syntaxbaum (*Abstract Syntax Tree*)
BNF	–	Backus-Naur-Form
BSD	–	Berkeley Software Distribution
CDDL	–	Common Development and Distribution License
CPL	–	Common Public License
CRUD	–	Create, Read, Update, Delete
CSV	–	Comma Separated Values
DOM	–	Document Object Model
DSL	–	Domain Specific Language
DSS	–	Decision Support System (Entscheidungsunterstützungssystem)
EBNF	–	Erweiterte Backus-Naur-Form
EDI	–	Electronic Data Interchange
EPL	–	Eclipse Public License
ETL	–	Extract, Transform, Load
FTP	–	File Transfer Protocol
GNU	–	GNU's Not Unix
GUI	–	Grafische Benutzeroberfläche (*Graphical User Interface*)
HTML	–	Hypertext Markup Language
HTTP	–	Hypertext Transfer Protocol
IID	–	Individualisierbarer Informationsdienst
JPEG	–	Joint Photographic Experts Group
IPTC	–	International Press Telecommunications Council
JDK	–	Java Development Kit
JSON	–	JavaScript Object Notation
LGPL	–	GNU Lesser General Public License (LGPL)
MIME	–	Multipurpose Internet Mail Extensions
MPEG	–	Moving Picture Experts Group
MPL	–	Mozilla Public Licese
NewsML	–	News Markup Language
NITF	–	News Industry Text Format
NX	–	News Extensions
OLAP	–	Online Analytical Processing
OOA	–	Objektorientierte Analyse (*Object-oriented analysis*)
OOD	–	Objektorientierter Entwurf (*Object-oriented design*)
OOP	–	Objektorientierte Programmierung (*Object-oriented programming*)
PDF	–	Portable Document Format

PNG	–	Portable Network Graphics
RDF	–	Resource Description Framework
REST	–	Representational State Transfer
RMI	–	Java Remote Method Invocation
RNC	–	RELAX NG Compact Syntax
RNG	–	RELAX NG
RSS	–	Rich Site Summary (RSS 0.91–0.95)
		RDF Site Summary (RSS 0.90, RSS 1.0–1.1)
		Really Simple Syndication (RSS 2.0)
SAX	–	Simple API for XML
SOAP	–	Ursprünglich Simple Object Access Protocol
		Mittlerweile nur noch „SOAP“
SportsML	–	Sports Markup Language
SQL	–	Structured Query Language
SWT	–	Standard Widget Toolkit
UML	–	Unified Modeling Language
URL	–	Uniform Resource Locator
W3C	–	World Wide Web Consortium
WAP	–	Wireless Application Protocol
WML	–	Wireless Markup Language
WSDL	–	Web Service Definition Language
XHTML	–	eXtensible Hypertext Markup Language
XML	–	eXtensible Markup Language
XSLT	–	XSL Transformations

1 Einleitung

1.1 Problemstellung

Die vorliegende Diplomarbeit entstand im Rahmen eines Forschungsprojekts[1], das sich unter anderem mit der Untersuchung individualisierbarer Informationsdienste (IID), wie der Individualisierten Zeitung (IZ), beschäftigt.

Solche Dienste bündeln Informationen unterschiedlicher Quellen entsprechend den persönlichen Informationsbedürfnissen und Präferenzen eines Abonnenten.[2] Dafür sind zahlreiche Quellen denkbar. Beispiele sind Nachrichtenartikel präferierter Themen, Wetterberichte bestimmter Regionen, Börsenkurse der gehaltenen Aktien oder auch News-Feeds.[3]

Die Quellen und die von ihnen gelieferten Informationen können sehr heterogen sein. Der Abruf der Informationen aus den Quellen geschieht über unterschiedliche Protokolle und Abfrufschnittstellen. Die gewonnenen Informationen können selbst bei gleicher Art (z.B. Nachrichtenartikel) unterschiedlich strukturiert oder gar unstrukturiert vorliegen, was ihre weitere Verwendung in der Verarbeitungskette erschwert oder unmöglich macht. Metadaten der einzelnen Inhalte sind je nach Quelle unterschiedlich ausgezeichnet oder fehlen möglicherweise ganz.

Es ergibt sich eine Vielzahl zu berücksichtigender Fälle, die den Zugriff auf die Informationen sowie deren Verarbeitung wesentlich erschwert. Um die Heterogenitäten zu überwinden und den entstehenden Verarbeitungssaufwand zu reduzieren, müssen die gewonnenen Daten einem Integrationsprozess unterzogen werden. Für diesen Gesamtprozess der Inhaltsverarbeitung konnte kein am Markt[4] erhältliches Softwareprodukt gefunden werden.

Die verfügbaren Softwarepakete für die Verlagsbranche sind durchgängig auf einen manuellen, redaktionellen Workflow ausgelegt. Die Content-Manangement-Systeme fokussieren die redaktionelle Verwaltung von Inhalten, nicht jedoch deren Gewinnung oder Integration für eine automatisierte Weiterverarbeitung.

---

[1] Das Forschungsprojekt wird durch den Lehrstuhl für Wirtschaftsinformatik, insbesondere Informationsmanagement, unter der Leitung von Prof. Dr. Detlef Schoder an der Universität zu Köln geleitet.

[2] Für eine Definition eines „individualisierbaren Informationsdienstes“ siehe Abschnitt 2.2.1.

[3] Weitere, konkrete Beispiele findet man im Anhang in Unterkapitel 9.1.

[4] Die Recherche umfasste u.a. des IFRA Anbieterverzeichnis (`http://www.ifra.com/WebSite/ifra.nsf/HTML/suppliers` (Abruf: 2007-07-20)), die Ausstellerliste der Drupa 2004 (`http://www.drupa.de/kati-cgi/kati/vis/custom/ext2/index.cgi?fair=drupa04&cp=0&lang=1` (Abruf: 2007-07-20)), das Produktverzeichnis von contentmanager.de (`http://www.contentmanager.de/itguide/produktfinder.html` (Abruf: 2007-07-22)) sowie eine direkte Recherche im WWW.

Wohl aber existieren Softwareprodukte, die in einzelnen Aufgaben unterstützend eingesetzt werden können und sollen. Es existiert z.B. ein breites Angebot an kommerziellen und nicht-kommerziellen Datenbanken, die relationale Daten und/oder XML-Daten speichern können.

Ebenso gibt es eine Vielzahl an Softwarepaketen zur Datenintegration, jedoch zielen diese Softwarepakete hauptsächlich auf die Integration und Aggregation von großen Mengen an Bestands- und Transaktionsdaten eines Unternehmens zur Verwendung in Data Warehouses ab. Dementsprechend unterstützen sie nur die dafür üblichen Quellen wie EDI, relationale Datenbanken oder XML-Dokumente. Viele weitere Informationsquellen können mit ihnen also nicht ohne weiteres erschlossen werden. Sie ermöglichen zwar oft die Entwicklung von Schnittstellen zu weiteren Quellen, es ist jedoch zu erwarten, dass der verbleibende Restnutzen dieser Softwarepakete die hohen Lizenzkosten sowie den zusätzlichen Aufwand zur Einarbeitung in die Programmierschnittstellen dieser sehr komplexen Systeme nicht rechtfertigen kann.

1.2 Zielsetzung

Da keine adäquate bestehende Lösung auf die sich ergebende Problemstellung gefunden werden konnte, soll im Rahmen dieser Arbeit ein prototypisches Softwaresystem konzipiert und entwickelt werden. Es soll einen Zugriff auf die sehr heterogenen Informationsquellen ermöglichen sowie die Verarbeitung der aus ihnen gewonnenen Daten so weit wie möglich vereinfachen. Die durch das System angebotene Schnittstelle verbirgt alle technischen Details des Zugriffs auf die Informationen und stellt eine integrierte Sicht auf die Daten zur Verfügung.

Die zur Verfügung gestellten Daten werden außerhalb des in dieser Arbeit zu entwickelnden Systems von mehreren Komponenten (Erkennung von Duplikaten, Kategorisierung, etc.)[5] weiterverarbeitet sowie über unterschiedliche Ausgabemedien präsentiert. Zur Verringerung des Aufwands bei Verarbeitung und Ausgabe der Daten sollen deren Austauschformate soweit wie möglich durch Transformationen vereinheitlicht werden. Beim Zugriff auf die Quellen müssen Probleme, die sich durch das Verhalten der Quellen (wie z.B. hohe Antwortzeiten, hohe Dynamik) ergeben, berücksichtigt und behandelt werden.

Der zu entwickelnde Prototyp soll möglichst alle definierten Anforderungen erfüllen und ohne nennenswerte Einschränkungen lauffähig sein. Im Rahmen dieser Arbeit können jedoch nicht alle Forderungen, die an ein vollständiges Produktivsystem zu stellen sind, erfüllt werden. So ist eine Optimierung des Laufzeitverhaltens nur

[5] Für eine genauere Beschreibung des Kontextes siehe Unterkapitel 2.2.

begrenzt möglich. Ein Spielraum für spätere Optimierungen soll jedoch eingeplant werden.

Außerdem wird der Prototyp nur eine begrenzte Anzahl an Informationsquellen und Austauschformaten einbinden. Es soll aber möglich sein, den Prototypen durch nachträgliche Entwicklung weiterer Komponenten zu erweitern. Dazu ist eine flexible Architektur zu schaffen, die es erlaubt, über möglichst einfache Schnittstellen zusätzliche Funktionalität in Form nachträglich zu entwickelnder Komponenten zu ergänzen.

1.3 Aufbau der Arbeit

Zunächst werden in Kapitel 2 die für die restlichen Ausführungen nötigen Grundlagen geschaffen. Diese umfassen die Definition wichtiger Begriffe, sowie eine Beschreibung des Kontextes, in dem diese Arbeit entsteht. Ebenso wird die Problemstellung weiter konkretisiert und schließlich wird ein Überblick über mögliche Informationsquellen gegeben.

Anschließend werden in Kapitel 3 die konkreten Anforderungen definiert, die an das zu entwickelnde System zu stellen sind. Die Anforderungen werden Top-Down, beginnend bei den Anwendungsfällen, über grundsätzliche Anforderungen, bis hin zu detaillierteren Kriterien definiert und priorisiert.

Daraufhin werden in Kapitel 4 grundlegende Lösungsansätze für die entstehenden Probleme erarbeitet. Konzepte der Datenintegration werden auf eine grundsätzliche Eignung für die Erfüllung der Anforderungen hin überprüft und an die konkreten Bedürfnisse der Problemstellung angepasst. Ebenso werden Lösungsmöglichkeiten für einzelne Teilprobleme, wie den Abruf der Informationen, die möglichen Abfragen an das System sowie die Transformation zwischen den Austauschformaten diskutiert.

Aufbauend auf den vorgeschlagenen Lösungen wird in Kapitel 5 ein prototypisches Softwaresystem entworfen, das die Anforderungen mithilfe der erarbeiteten Lösungen erfüllen soll. Nach der Beschreibung grundsätzlicher Architekturentscheidungen folgt die Beschreibung des Entwurfs einzelner Teilkomponenten des Systems. Ebenso wird ein einfaches, generisches Datenmodell geschaffen, das der Identifikation, der Speicherung und dem Austausch der Informationen dient. Für die Informationen der konkret in dieser Arbeit angebundenen Quellen werden zudem quellübergreifend einsetzbare Austauschformate vorgestellt.

In Kapitel 6 wird schließlich die Implementierung der entworfenen Software dokumentiert. Wo möglich, werden bestehende Bibliotheken und Softwareprodukte zu Hilfe gezogen. Die wesentlichen Implementierungsentscheidungen sowie spezifische

Details einzelner Lösungen werden erläutert. Es folgen eine Leistungsbewertung sowie Hinweise zur Dokumentation des entwickelten Systems.

Die Arbeit wird in Kapitel 7 durch eine kurze Zusammenfassung der erreichten Ergebnisse sowie einen Ausblick auf mögliche Weiterentwicklungen abgeschlossen. Evtl. nicht erreichte Ziele und alternative Lösungsmöglichkeiten werden erläutert.

Innerhalb des Textes wird häufig Bezug auf vorige oder kommende Textstellen genommen. Um sicherzustellen, dass die Verweise eindeutig für den Leser sind, seien hier kurz die in den Referenzen verwendeten Namen der Gliederungsebenen erwähnt. Die Arbeit ist auf oberster Ebene in Kapitel, darunter in Unterkapitel, Abschnitte und Unterabschnitte gegliedert.

2 Grundlagen

2.1 Begriffliche Grundlagen

2.1.1 Information

Der Begriff **Information** wird sehr unterschiedlich verwendet und je nach Wissenschaft anders definiert.[6] Sehr viele Autoren[7] teilen jedoch die Ansicht, dass Informationen **(durch einen Menschen) gedeutete Daten bzw. Nachrichten** sind.

Aufgrund der Betonung der Deutung durch den Menschen wird klar, dass Informationen an sich nicht greifbar sind. Zur Übermittlung müssen sie in eine formalisierte Darstellung überführt werden. Derart repräsentierte Informationen nennt man **Daten**.[8]

Die Abgrenzung zwischen Daten und Informationen wird also im Wesentlichen darüber entschieden, ob man sie im Kontext einer physikalischen Darstellung (z.B. zur Verarbeitung durch eine Maschine) oder einer Deutung durch einen Menschen betrachtet.

Oft kann die Grenze zwischen Informationen und Daten aber nicht völlig scharf gezogen werden. Die Begriffe werden häufig synonym verwendet. In dieser Arbeit wird der Begriff **Information** verwendet, wenn sie an einen Menschen gerichtet ist. Das ist auch der Fall, wenn die Informationen aus fachlicher und nicht aus technischer Sicht betrachtet werden. Handelt es sich bloß um eine Repräsentation zur Verarbeitung in einer Maschine[9], so wird der Begriff **Daten** verwendet.

2.1.2 Datenintegration

Die Schwierigkeit der Abgrenzung von Daten und Informationen macht sich auch in den abgeleiteten Begriffen **Datenintegration** bzw. **Informationsintegration** bemerkbar. Man findet beide Begriffe in großer Häufigkeit und weitgehend synonymer Verwendung in der Literatur. Lenzerini definiert die Datenintegration wie folgt:

> *Data integration is the problem of combining data residing at different sources, and providing the user with a unified view of these data.*[10]

---

[6] Für eine ausführliche Diskussion des Informationsbegriffs vgl. zu diesem und den folgenden Absätzen Rechenberg (2003), S. 317 ff.

[7] Vgl. Rechenberg (2003), S. 326; Blieberger u. a. (2001), S. 17 f.; Checkland u. Scholes (1999), S. 303; ISO/IEC (1993) (Zitiert nach: Rechenberg (2003), S. 317–318).

[8] Vgl. Stahlknecht u. Hasenkamp (2004), S. 10.

[9] Tatsächlich wird die zu entwickelnde Software primär nicht durch Menschen sondern andere Softwarekomponenten genutzt. Daher wird der Begriff „Daten“ in dieser Arbeit wesentlich häufiger genutzt.

[10] Diese Definition findet man in Lenzerini (2002), S. 233.

Analog zur Unterscheidung zwischen Informationen und Daten könnte man die Informationsintegration von der Datenintegration abgrenzen: Betrifft die Integration bloß die Repräsentation von Informationen, also die Daten, so könnte man von Datenintegration sprechen. Werden jedoch semantische Veränderungen (z.B. Aggregationen) vorgenommen, so könnte man den Begriff Informationsintegration verwenden.

Die Grenze ist aber auch hier schwer klar zu definieren. Der eigentliche Integrationsablauf, der maschinell durchgeführt wird, muss mit Daten arbeiten. Die Integrationsvorschriften werden jedoch von Menschen definiert. Ebenso ist i.d.R. der Mensch der endgültige Adressat der integrierten Daten.

Da die in dieser Arbeit vorgenommenen Integrationen die Semantik der Informationen unverändert lassen sollen, erscheint der Begriff Datenintegration nach der oben beschriebenen Abgrenzung zutreffend. Allerdings wird im Folgenden hauptsächlich der hier[11] synonym betrachtete Begriff **Integration** genutzt, der auch Teil des Titels der Arbeit ist.

2.1.3 Abgrenzung: Benutzer und Abonnent

Im weiteren Verlaufe des Textes werden häufiger die Begriffe Benutzer und Abonnent verwendet, die hier keineswegs synonym zu betrachten sind: Der **Benutzer** eines Integrationssystems stellt formale Abfragen an das System und erhält strukturierte Daten als Antwort. Der **Abonnent** eines IIDs hingegen stellt selbst keine formalen Abfragen an das System und bekommt keine rohen Daten als Antwort. Vielmehr bekommt er entsprechend seiner Präferenzen ausgewählte und im gewünschten Format einheitlich präsentierte Informationen.

Der Abonnent eines IIDs hat somit eine wesentlich abstraktere Sicht auf das System als der Benutzer (im Sinne oben genannter Definition) des Integrationssystems. Die Datenintegration macht also nur einen – allerdings wichtigen – Teil der Informationsverarbeitung eines IIDs aus, der zusätzlich eine Vielzahl anderer Aufgaben[12] erfüllen muss.

2.1.4 Informationsdienst

Ein **Informationsdienst** vermittelt zwischen Informationsquellen (im Folgenden **Quelle**) und Informationkonsumenten (im Folgenden **Abonnent**). Er bündelt Informationen aus unterschiedlichen Quellen und stellen sie dem Abonnenten zur Verfügung.

---

[11] Im Softwarebereich existiert eine Vielzahl weiterer Integrationen. Zusätzliche Beispiele wären die Prozessintegration oder die Anwendungsintegration.

[12] Z.B. individualisierte Informationsauswahl, Ausgabe in unterschiedlichen Formaten. Für weitere Details siehe Unterkapitel 2.2.

Oft aggregieren solche Dienste Informationen zu einem bestimmten Thema und stellen sie dem Abonnenten in einer einheitlichen Form zur Verfügung. Sie befreien den Abonnenten somit davon, die Quellen eigenhändig nach den Informationen zu durchsuchen und erleichtern (oder ermöglichen teilweise) erst den Zugang zu den gewünschten Informationen. Informationsdienste können auch andere Informationsdienste als Quelle nutzen, ebenso wie sie selbst als Quelle für andere Informationsdienste dienen können.

Beispiele für Informationsdienste sind Wetterdienste, Nachrichtenticker oder auch Internetportale. Die Definition eines **individualisierbaren Informationsdienstes** erfolgt in Abschnitt 2.2.1.

2.2 Individualisierbare Informationsdienste als Kontext der entwickelten Software

Die in dieser Arbeit zu entwickelnde Softwarekomponente ist ein Teil eines individualisierbaren Informationsdienstes. Sie kann nicht ohne ein Verständnis ihres Kontextes – einem IID – verstanden und entwickelt werden.

Um diesen erforderlichen Kontext zu beschreiben, wird im Folgenden zunächst der allgemeine Begriff des IIDs definiert. Zur Konkretisierung des Begriffs werden anschließend einige praktische Anwendungsmöglichkeiten vorgestellt. Um die zu entwickelnde Komponente im Gesamtbild eines IIDs besser einordnen zu können, werden daraufhin die typischen Anwendungsfälle sowie die zur Erfüllung der Anwendungsfälle benötigten Komponenten eines IIDs kurz beschrieben.

2.2.1 Definition eines individualisierbaren Informationsdienstes

Individualisierbarkeit

Unterschiedliche Abonnenten haben unterschiedliche Informationsbedürfnisse. Somit ist es nicht sinnvoll, alle Abonnenten mit den selben Informationen zu versorgen. Die **Individualisierbarkeit** ist daher ein wichtiges Kriterium eines Informationsdienstes[13]. Sie erhöht den Nutzen eines Informationsdienstes wesentlich, da sie die Quote aus relevanter Informationen im Verhältnis zur Gesamtmenge angebotener Informationen verbessert.

Insbesondere weil Informationen und Dienste zunehmend elektronisch zur Verfügung gestellt werden, ist eine weitgehende Automatisierung von Informationsdiensten möglich. Informationen können automatisiert aus unterschiedlichen Quellen abgerufen, aggregiert und in einer integrierten Form angeboten werden. Erst durch diese Automatisierung ist es möglich, Informationsdienste effizient für eine große Anzahl an Abonnenten zu individualisieren.

Solche Informationsdienste können auf zwei wesentliche Arten individualisiert werden: Über direkte und indirekte Individualisierung.[14]

---

[13] Zur Definition eines Informationsdienstes siehe Abschnitt 2.1.4.

[14] In Rossi u. a. (2001), S. 276 f. werden unter *Content Personalization* genau diese Beispiele genannt: Der Abonnenten kann einerseits Personalisierungen explizit definieren, andererseits sind automatische Personalisierungen denkbar, die Informationen über den Abonnenten nutzen. Ebenso werden in Hanani u. a. (2001), S. 206, Figure 1 explizite, implizite, sowie kombinierte Methoden zur Erhebung von Wissen über einen Abonnenten unterschieden.

Direkte Individualisierung: Konfiguration

Bei einer direkten Individualisierung kann der Abonnent die Auswahl der Informationen unmittelbar über eine explizite **Konfiguration** des Informationsdienstes beeinflussen. So kann er beispielsweise bestimmte Quellen ausschließen, oder zusätzliche Quellen selbst definieren. Ebenso könnte er den Umfang der ihm angebotenen Informationen bestimmen oder zwischen unterschiedlichen Ausgabemedien (Web-Browser, Papier, Mobiltelefon, etc.) wählen.

Indirekte Individualisierung: Aus Rückmeldungen lernen

Eine indirekte Individualisierung umfasst alle Möglichkeiten, die die Zusammenstellung der Informationen nur mittelbar – also über einen Zwischenschritt – beeinflussen. So könnte der Abonnent z.B. für einzelne Informationseinheiten oder auch Gruppen von Informationen eine Aussage über deren Relevanz für ihn tätigen. Diese **Rückmeldungen (Feedback)** speichert der Informationsdienst in einem individuellen Profil, das er zur Beurteilung der Relevanz der Informationseinheiten für jeden einzelnen Abonnenten heranziehen kann. Idealerweise nähert sich das Informationsangebot so mit zunehmender Anreicherung des Profils dem tatsächlichen Bedarf des Abonnenten an.

Individualisierbare Informationsdienste als Agenten

Ein Agent unterstützt einen Menschen bei seiner Arbeit, indem er seine Aufgaben (z.B. Sortieren und Filtern von Informationen, Vorschlagen oder gar Ausführen bestimmter Handlungen) unter Rücksichtnahme auf dessen individuelle Anforderungen teilweise oder ganz übernimmt.

Maes unterscheidet[15] **semi-autonome Agenten**, die vom Benutzer durch explizite Regeln programmiert werden müssen, **wissensbasierte Agenten**, die im Voraus manuell mit einer großen Menge an Wissen über die Anwendung und den Benutzer versorgt werden müssen, und **lernende Agenten**, die aus den Interessen, dem Verhalten und den Vorlieben des Benutzers lernen und ihn so zunehmend besser unterstützen können.

Man kann einen Informationsdienst als einen Agenten auffassen, der dem Abonnenten die Aufgabe der Informationsbeschaffung erleichtert. Ein konfigurierbarer Informationsdienst kann als semi-autonomer Agent verstanden werden. Ein Informationsdienst, der aus den Rückmeldungen des Abonnenten Rückschlüsse zieht, kann als lernender Agent verstanden werden. Einem nicht-individualisierbaren Informationsdienst würde hingegen die wesentliche Eigenschaft fehlen, einen Benutzer

[15] Für eine Klassifizierung von Agenten, sowie einige Beispiele vgl. Maes (1994), S. 32 ff.

entsprechend seinen individuellen Anforderungen zu unterstützen. Er wäre dann nicht als Agent zu bezeichnen.

Maes stellt in ihrem Artikel u.a. auch einen Agenten zur Filterung von Nachrichten (im Usenet) vor, der dem Prinzip der indirekten Individualisierung sehr nahe kommt. Ebenfalls beschreibt sie die Möglichkeit des *Social Filtering*, wo die Auswahl der Informationen für einen Abonnenten nicht nur aufgrund seiner eigenen Rückmeldungen, sondern auch auf denen von anderen Abonnenten mit ähnlichen Interessen basiert. [16]

Zusammenfassung

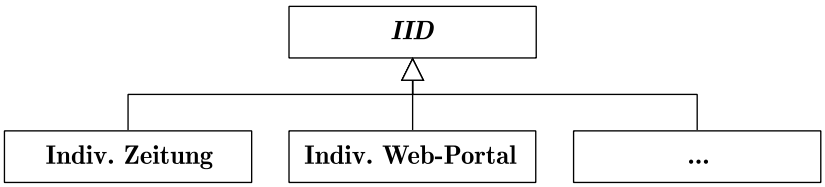
Zusammenfassend hat ein **individualisierbarer Informationsdienst (IID)** folgende Eigenschaften:

- Zugriff auf mehrere, möglicherweise heterogene Quellen.
- Unterstützung des Abonnenten bei der Informationsauswahl. Der Abonnent kann die Auswahl beeinflussen und somit auf seine Bedürfnisse abstimmen.
- Einheitliche Sicht auf die Informationen für den Abonnenten.
- Evtl. Ausgabe über unterschiedliche, vom Abonnenten wählbare Medien.

2.2.2 Anwendungen

Die obige Definition eines IIDs ist allgemein formuliert und noch nicht an eine spezifische Anwendung gebunden. Ein IID ist zunächst ein abstraktes Konzept, das durch Anwendungen konkretisiert werden kann, wie Abbildung 2-1 es veranschaulicht. Einige dieser Anwendungen werden zur Zeit der Entstehung dieser Arbeit am betreuenden Lehrstuhl im Rahmen von Forschungsprojekten untersucht und ausgearbeitet.

Abb. 2-1: Der IID als abstraktes Konzept mit konkreten Anwendungen in UML-Notation



[16] Vgl. zu diesem Absatz Maes (1994), S. 32–33, 38–40.

Eine mögliche Anwendung ist eine **individualisierte Zeitung**[17], die einem Abonnenten ganz nach seinen persönlichen Interessen und Informationsbedürfnissen zusammenstellt wird. Ein weiterer positiver Effekt der Individualisierung ist die Möglichkeit der zielgerichteten Werbung innerhalb einer individuellen Zeitung. Streuverluste werden durch diese Art der Werbung minimiert.

Eine weitere Anwendungsmöglichkeit ist ein **individualisierbares Web-Portal**, auf dem sich ein Abonnent die Informationen ganz nach seinen Wünschen zusammenstellen lassen kann. Zusätzliche Vorteile durch die Nutzung des Internets als Medium ergeben sich vor allem durch die höhere Interaktivität. Es ist wesentlich leichter, ein präzises Profil über die Interessen des Abonnenten zu erstellen. Dazu könnte man den *Clickstream*[18] sowie einfache Rückmeldungen (z.B. Information sagt zu/nicht zu) nutzen.

Es sind auch durchaus weitere, evtl. spezialisiertere Anwendungen denkbar. So könnte man in einem **unternehmensinternen Portal** aktuelle Informationen anbieten, die von Mitarbeitern, der Geschäftsführung oder auch aus externen Quellen stammen können. Der Abonnent eines solchen Portals könnte durch eine Individualisierung den Umfang beschränken und die Auswahl der für seine Arbeit relevanten Informationen beeinflussen, um eine möglichst effiziente Informationsaufnahme zu erreichen. Zusätzlich könnte ein IID in Redaktionen behilflich sein, um den jeweiligen Redakteuren die individuell für sie relevanten Informationen zukommen zu lassen. Generell kann der Einsatz von IIDen durch die Automatisierung der Informationsauswahl zu einer Erhöhung der Produktivität führen.

Im Folgenden wird stets ein allgemeiner, abstrakter IID betrachtet. Die Ausführung -en sowie die gewonnenen Erkenntnisse beziehen sich nicht auf eine spezifische Anwendung eines IIDs, sondern sollen vielmehr anwendungsübergreifend gültig und verwertbar sein.

2.2.3 Anwendungsfälle

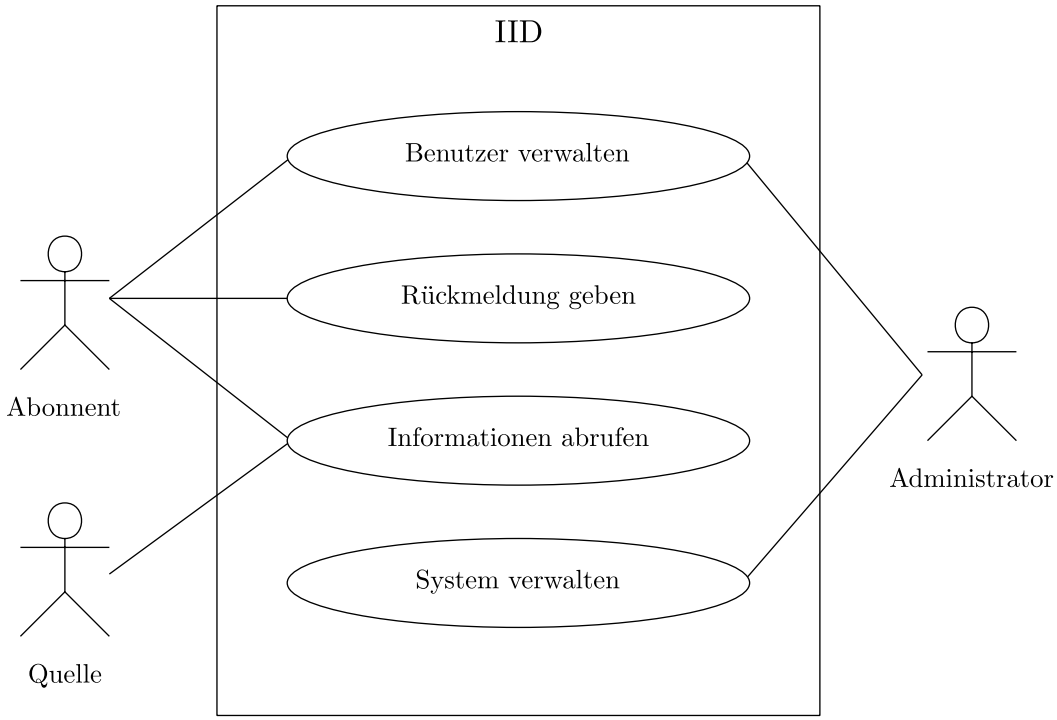
Eine gute Möglichkeit zur Beschreibung von Systemen auf hoher Ebene ist die Darstellung ihrer Anwendungsfälle (*Use Cases*), die die Möglichkeiten der Benutzung des Systems aus externer Sicht definieren. Diese externe Sicht auf einen allgemeinen

[17] Die individualisierte Zeitung ist durch ein internationales Patent geschützt. Für die Patentschrift siehe Schoder u. Sick (2003). Weitere Informationen zur Individualisierten Tageszeitung findet man unter `http://www.medieninnovation.com/` (Abruf: 2007-08-05).

[18] Der *Clickstream* entspricht der Abfolge der abgerufenen Informationen.

IID[19] wird in Abbildung 2-2 (S. 12) als UML-Anwendungsfalldiagramm veranschaulicht.

Abb. 2-2: UML-Anwendungsfalldiagramm für einen individualisierbaren Informationsdienst



Ein IID interagiert mit mehreren Akteuren. Der **Abonnent** kann sich über die **Benutzerverwaltung** registrieren, an- und abmelden und seine persönlichen Daten bearbeiten. Der Anwendungsfall „Benutzer verwalten“ umfasst auch die Individualisierung mittels Konfiguration: So können dort bspw. der Umfang der Informationen sowie die Auswahl bestimmter persönlicher Assistenzen[20] definiert werden. Ebenso hat er die Möglichkeit, den IID über **Rückmeldungen** zu trainieren, die in ein individuelles Profil einfließen, mit dem die Auswahl der Informationen verbessert werden soll. Schließlich kann er die **Informationen abrufen**, die entsprechend seinen individuellen Wünschen zusammengestellt werden.

Diese Informationen können aus vielen unterschiedlichen **Quellen** stammen. Sie werden entsprechend den Präferenzen des Abonnenten ausgewählt und in einem einheitlichen Format zur Verfügung gestellt.

---

[19] Zur Analyse der Anwendungsfälle für eine „Individualisierte Zeitung“ als Anwendung eines IIDs vgl. Kugler (2005), S. 11–15.

[20] Für die Definition einer **persönlichen Assistenz** siehe Unterkapitel 2.4.

Neben dem Abonnenten ist auch ein **Administrator** an der **Benutzerverwaltung** beteiligt. Er kann sich eine Übersicht über die registrierten Abonnenten verschaffen und, falls nötig, Änderungen vornehmen. Ebenso ist er für die **Systemverwaltung** verantwortlich, die alle technischen Verwaltungsaufgaben umfasst. Er hat die Möglichkeit, technische Parameter zu definieren und das System bei Bedarf anzuhalten oder neuzustarten. Die Systemverwaltung umfasst auch die Verwaltung der Quellen, wie z.B. das Hinzufügen neuer oder Entfernen vorhandener Komponenten zur Anbindung bestimmter Quellen sowie die technische Konfiguration (z.B. Abrufintervalle) der Quellen.

2.2.4 Teilaufgaben und Komponenten

Ein IID muss mehrere Aufgaben erledigen, um dem Abonnenten Informationen entsprechend seines individuellen Bedarfs zur Verfügung stellen zu können. Diese Aufgaben werden soweit wie möglich durch separate Komponenten bewerkstelligt. Die Datenintegration ist nur eine dieser Komponenten.

Zum Verständnis der Umgebung, in der die Integrationskomponente entwickelt werden soll, ist es hilfreich, sich auch ein Bild der übrigen Komponenten zu machen. Eine Komponente zur **Benutzerverwaltung** stellt Funktionen zur Pflege der Benutzerdaten zur Verfügung. Die **Kategorisierung** ordnet noch nicht kategorisierten Inhalten geeignete Kategorien zu oder erweitert eine vorhandene Kategorisierung. Ebenso ist es wichtig, **Duplikate** zu erkennen und die Inhalte entsprechend der **individuellen Eignung** zu bewerten. Die **Auswahl** der Inhalte wird im wesentlichen durch eine Gegenüberstellung von Kosten und abonnentenindividuellem Nutzen optimiert. Schließlich werden die ausgewählten Inhalte durch eine **Layout-Komponente** in die gewünschten Ausgabemedien überführt und sind über ein **Frontend** zugreifbar.

Die **Integrationskomponente** ist dafür verantwortlich, den anderen Komponenten die Daten zur Verfügung zu stellen. Sehr viele Komponenten greifen zur Verrichtung ihrer Aufgaben auf die Inhalte zu. Damit hat die Integrationskomponente einen wichtigen Stellenwert im Gesamtbild eines IIDs.

2.3 Bedeutung der Datenintegration in individualisierbaren Informationsdiensten

2.3.1 Bedeutung der Informationen

Die Informationen spielen eine wichtige Rolle in einem IID. Sie stehen im Mittelpunkt des Interesses des Abonnenten und deren maßgeschneiderte Auslieferung ist die Kernaufgabe eines IIDs. Sie sind ein zentraler Punkt in der Definition eines IIDs sowie in der Erfüllung der Anwendungsfälle[21].

Auch aus Entwicklersicht sind die Informationen von hoher Bedeutung. In Abschnitt 2.2.4 wurde verdeutlicht, dass viele andere Komponenten mit den Informationen in Berührung kommen und sie verarbeiten müssen. Die Informationen sind die Grundlage für den gesamten Verarbeitungsprozess innerhalb eines IIDs.

2.3.2 Heterogenität der Daten

Die Informationen können aus unterschiedlichen Quellen stammen. Ihre Datenrepräsentationen können sehr heterogen sein. Um dem Abonnenten die Aufnahme der Informationen zu erleichtern, wird eine einheitliche Präsentation dieser Informationen gefordert.[22]

Aber nicht nur dem Abonnenten erschwert eine unnötige Heterogenität der Darstellung die Informationsaufnahme. Auch innerhalb eines IIDs sorgt die Heterogenität für eine hohe Komplexität der Verarbeitung der Daten, die sich auf zahlreiche, teilweise in Abschnitt 2.2.4 beschriebene, Komponenten auswirkt.

Wie man Abbildung 2-3 (S. 15) entnehmen kann, wird die Heterogenität der Daten im Wesentlichen durch zwei Dimensionen bestimmt: Durch den Strukturierungsgrad und durch das Austauschformat.

Bzgl. des **Strukturierungsgrads** unterscheidet man strukturierte, semi-strukturierte und unstrukturierte Daten[23].

Daten sind **strukturiert**, wenn sie einem separaten Schema folgen, das die genaue Zusammensetzung der Daten aus komplexen oder atomaren Datentypen im Voraus definiert. Relationale Datenbanken speichern Daten strukturiert ab. Auch sind XML-Daten, die einem expliziten Schema (z.B. XML Schema) folgen, i.d.R. strukturierte Daten.

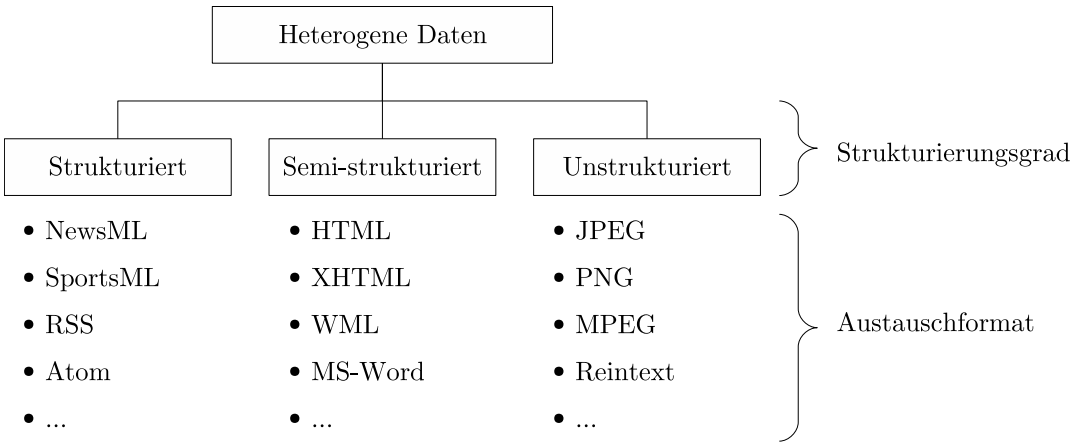
---

[21] Die Anwendungsfälle eines IIDs wurden in Abschnitt 2.2.3 beschrieben.

[22] Vgl. zu diesem Absatz die Definition eines IIDs in Abschnitt 2.2.1.

[23] Für die Definition strukturierter, semi-strukturierter und unstrukturierter Daten vgl. Domenig u. Dittrich (1999), S. 68–69.

Abb. 2-3: Unterscheidungskriterien für heterogene Daten



Semi-strukturierte Daten[24] folgen zwar einem Schema, das jedoch in die Daten eingebettet sein kann und möglicherweise viele strukturelle Freiheiten gewährt. So können z.B. Elemente ausgelassen oder neue hinzugefügt werden. Die Daten können eine selbstbeschreibende Struktur haben, die jedoch nicht explizit definiert wurde. Beispiele sind XML-Daten, die keinem expliziten Schema folgen, oder auch um Strukturelemente angereicherte Reintexte. Web-Seiten zählen ebenfalls zu den semi-strukturierten Daten.

Unstrukturierte Daten folgen keinem Schema, oder das Schema definiert lediglich die Zusammensetzung der Daten aus Bytes oder Zeichen, nicht jedoch die inhaltliche Struktur. Multimediadaten (Audio, Foto, Video) oder Texte, die keine Information über die inhaltliche Strukturierung enthalten, sind Beispiele für unstrukturierte Daten.

Darauf aufbauend können die Daten anhand ihres **Austauschformats** unterschieden werden. Bei relationalen Daten ist es das Datenschema, bei XML-Dokumenten kann es die XML-Schema-Definition sein, bei Abbildungen ist es das Dateiformat (z.B. JPEG oder GIF). Beispiele dafür können ebenfalls Abbildung 2-3 entnommen werden.

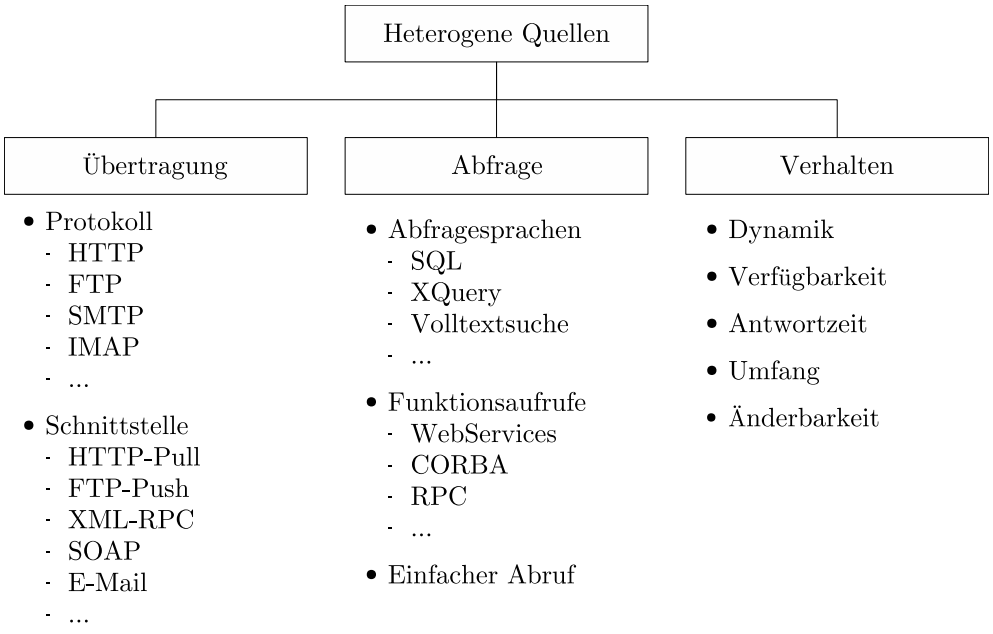
2.3.3 Heterogenität der Quellen

Nicht nur die Daten selbst unterliegen einer Heterogenität. Auch die Quellen, aus denen sie gewonnen werden, können sehr heterogen sein.

[24] Für eine ausführliche Behandlung semi-strukturierter Daten und den mit ihnen verbundenen Problemen vgl. Abiteboul (1997), S. 2–5.

Quellen sind heterogen, wenn sie sich in einem oder in mehreren Kriterien unterscheiden. Um also die Heterogenität von Quellen definieren zu können, muss man zunächst die Kriterien identifizieren, in denen sie sich unterscheiden können. Abbildung 2-4 gibt einen Überblick über diese Kriterien.

Abb. 2-4: Unterscheidungskriterien für heterogene Quellen



Unterschiede ergeben sich aus der Übertragung der Informationen über Protokolle und Schnittstellen, aus verschiedenen Möglichkeiten der Abfrage der Daten sowie aus bestimmten zu berücksichtigenden Verhaltensweisen der Quellen.[25]

Zunächst kann man die **Übertragung** nach **Protokollen** unterscheiden. Protokolle sind Vereinbarungen zwischen den Kommunikationspartnern, die den Aufbau, die Überwachung und den Abbau von Verbindungen beschreiben.[26] Für den Zugriff auf die Quellen sind insbesondere die Protokolle auf der Anwendungsschicht[27] wie z.B. HTTP, FTP und SMTP/POP3/IMAP interessant.

Auf den Protokollen bauen unterschiedliche **Schnittstellen** auf, die den Zugriff auf die Daten gewährleisten. Solche Schnittstellen können sehr einfach sein und kaum

[25] Bzgl. der Heterogenität von Informationsquellen vgl. zu diesem und den folgenden Absätzen Leser u. Naumann (2006), S. 62–64.

[26] Vgl. zu dieser Definition Stahlknecht u. Hasenkamp (2004), S. 95 und Tanenbaum (2002), S. 27.

[27] Die Schichtung von der Datenübertragung wurde in ISO/IEC (1994), S. 32–33, standardisiert.

Einschränkungen definieren (z.B. HTTP-Pull[28], FTP-Push[29], XML-RPC, REST). Sie können aber auch sehr komplex sein (z.B. SOAP) und eine Reihe von Technologien und Standards nutzen, um die Schnittstellen zu definieren.

Auf einer noch höheren Ebene kann der Zugriff auf die Daten der Quellen über verschiedene **Abfragemöglichkeiten** geschehen.

Insbesondere beim direkten Zugriff auf Datenbanken kann man über leistungsfähige **Abfragesprachen** wie SQL oder XQuery auf die Datenbestände zugreifen. Solche Quellen erlauben es i.d.R., gezielt auf bestimmte Daten aus den gesamten Bestand zuzugreifen.

Quellen, die einen Zugriff auf die Daten über **Funktionsaufrufe** erlauben, sind meist ebenfalls gezielt nach bestimmten Daten aus ihrem gesamten Datenbestand abfragbar. Die Auswahl der Daten erfolgt über die Parameter, die von den Funktionen angeboten werden. Diese Form des Abrufs ist weniger flexibel als der Abruf über Abfragesprachen, da sie den Zugriff nur über explizit definierte Funktionen erlaubt. Die möglichen Abfragen sind somit durch den Informationsanbieter begrenzt, was ihm eine größere Sicherheit bietet.

Sehr häufig wird man auf Quellen treffen, die keine gezielten Abfragen über Abfragesprachen oder Funktionen erlauben. Dies können Nachrichtenticker sein, die lediglich die neuesten Meldungen übermitteln. Auch RSS-Feeds sind nicht gezielt abfragbar, sondern liefern stets nur die neuesten Einträge aus. Ein Abruf der Daten aus solchen Quellen liefert also unter Umständen nur einen kleinen Ausschnitt des gesamten Datenbestandes dieser Quellen. Bei solchen Quellen ist es nicht möglich, gezielt nach einer Meldung zu suchen, die bspw. über ein bestimmtes Thema an einem bestimmten Datum berichtet. Die Daten müssen regelmäßig über **einfache Abrufe** gewonnen werden. Die Selektion erfolgt dann in einem nachgelagerten Schritt.

Schließlich kann man die Quellen anhand unterschiedlicher Kriterien unterscheiden, die ihr **Verhalten** beschreiben.

So ist die **Dynamik** der Informationen ein wichtiges Kriterium. Der Börsenkurs einer Aktie ändert sich sehr häufig (hohe Dynamik), ein Nachrichtenartikel wird nach seiner Veröffentlichung eher selten geändert (geringe Dynamik).

Die **Verfügbarkeit** einer Quelle beschreibt die Fähigkeit, zu einem bestimmten Zeitpunkt zugreifbar zu sein. Es gibt eine Vielzahl von Faktoren (Hardwarefehler,

[28] Unter einer **Pull-Schnittstelle** wird verstanden, dass die Daten beim Anbieter (regelmäßig) abgefragt werden.

[29] Unter einer **Push-Schnittstelle** wird verstanden, dass die Daten vom Anbieter (regelmäßig) in das System des Kunden transferiert werden.

Netzwerkstörungen, Softwarefehler, Wartungsarbeiten etc.), die die Verfügbarkeit einer Quelle negativ beeinflussen können.

Insbesondere wenn viele individuelle Abfragen an eine Quelle gestellt werden, ist deren **Antwortzeit** von Bedeutung. Sie ist die Zeit, die von dem Absetzen einer Abfrage bis zum Erhalt des Ergebnisses vergeht.

Ebenso ist der **Umfang** der von einer Quelle angebotenen Informationen ein wichtiges Kriterium. Insbesondere der Zugriff auf sehr umfangreiche Datenbestände muss eine besondere Beachtung finden. So ist es in diesem Falle z.B. nicht praktikabel, den gesamten Bestand einer Quelle zu extrahieren, wenn nur ein Bruchteil davon benötigt wird. Diese Thematik wird jedoch in Abschnitt 4.2.3 genauer betrachtet.

Die **Änderbarkeit** der von einer Quelle angebotenen Informationen ist für einen IID kaum relevant. Trotzdem soll sie hier als Unterscheidungskriterium einer Quelle nicht unerwähnt bleiben. Die Informationen sind änderbar, falls sie durch den Abonnenten bearbeitet werden können. Diese Funktion ist jedoch nicht Bestandteil eines Informationsdienstes. Die Informationen werden in unabhängigen Systemen produziert und bearbeitet, wohingegen der Informationsdienst diese Informationen lediglich aggregiert.

2.3.4 Komplexität durch Heterogenität

Die oben beschriebene Heterogenität der Daten sowie der Quellen, aus denen sie gewonnen werden, erschwert die Verarbeitung der Daten wesentlich. Dieses Problem wiegt umso schwerer, wenn man bedenkt, dass die Daten durch eine Vielzahl von Komponenten (siehe Abschnitt 2.2.4) verarbeitet werden müssen. Diese Komponenten müssten ohne eine Integration sehr viele unterschiedliche Fälle berücksichtigen, was zu einer hohen Komplexität der Verarbeitung der Daten führt.

Diese Komplexität soll durch ein Integrationssystem reduziert werden. Die Anforderungen an ein solches System werden Kapitel 3 definiert.

2.4 Mögliche Informationsquellen

Um sich ein genaueres Bild der Bandbreite der zu verarbeitenden Informationen zu verschaffen, werden im Folgenden die in einem IID denkbaren Informationsquellen beschrieben. In dieser Arbeit werden folgende Arten von Quellen unterschieden: Nachrichten, Blogs, Bilder, Werbung und persönliche Assistenzen.

Eine Übersicht mit zahlreichen Beispielen zu möglichen Quellen ist im Anhang in Unterkapitel 9.1 zu finden. Da ein Ziel dieser Arbeit die Implementierung eines Softwaresystems zur Integration der Informationen dieser Quellen ist, sind auch einige technische Eigenschaften der Quellen von Interesse. Daher wird für die im Anhang dargestellten Beispiele ebenfalls angegeben, welche Arten von Informationen (Text, Bild, Ton oder Video) die Quellen liefern, in welchem Austauschformat die Informationen vorliegen und über welche Protokolle sie abgerufen werden können.

Abschließend werden diejenigen Quellen ausgewählt, die im Rahmen dieser Arbeit mit entsprechenden Software-Komponenten an das System angebunden werden.

2.4.1 Nachrichten

Nachrichten sind sehr wichtig für einen IID. Sie informieren den Abonnenten über aktuelle Ereignisse – idealerweise zielgerichtet entsprechend seiner Interessen.

Nachrichten bezieht man als kommerzieller Dienstleister primär von Nachrichtenagenturen, die ein breites Spektrum an zumeist qualitativ hochwertigen, redaktionellen Inhalten anbieten. Über entsprechende Schnittstellen können die Inhalte automatisiert abgerufen werden.

Im Leistungsangebot vieler Agenturen findet man sowohl reine **Textdienste** als auch **Multimediadienste**. Erstere liefern die Nachrichtenmeldungen in Rohform in die Redaktionssysteme der Kunden, wo sie durch die Redakteure weiterverarbeitet werden. Die Multimediadienste liefern hingegen redaktionell aufbereitete Artikel, die um Fotos, Illustrationen und gelegentlich sogar um Ton- oder Videoinhalte angereichert sind und ohne manuelle Eingriffe veröffentlicht werden können. Kann oder möchte man keine eigene Redaktion stellen, sind die Multimediadienste zu bevorzugen. Daher werden reine Textdienste im Folgenden nicht weiter betrachtet.

Es gibt eine große Anzahl an Nachrichtenagenturen, wobei sich die Breite des Informationsangebots stark unterscheiden kann. So decken einige Agenturen praktisch alle Themenbereiche ab, wobei andere sich auf Nischen, wie etwa die Sportberichterstattung, konzentrieren.

Nachrichtenagenturen stellen jedoch nicht die einzige Möglichkeit dar, an Nachrichten zu gelangen. Ebenso gibt es eine Vielzahl an Web-Seiten, die oft zu einem

bestimmten Themengebiet (z.B. Heise Online[30] für IT-Nachrichten), durchaus aber auch ressortübergreifend (z.B. Spiegel Online[31]), aktuelle Nachrichten liefern.

Konkrete Beispiele findet man im Anhang in Abschnitt 9.1.1.

2.4.2 Blogs

Eine weitere, umfangreiche[32] Quelle stellen **Blogs** (auch: *Weblogs*) dar. Die Qualität der Inhalte kann stark variieren und ist ein kontrovers[33] diskutiertes Thema. Sie können aber insbesondere für spezielle Themen eine sehr gute Ergänzung zu den klassischen redaktionellen Inhalten sein. Weitere Vorteile sind die hohe Aktualität sowie die Nähe zum Geschehen. Blogs werden meistens von Einzelpersonen geführt, können aber auch von mehreren Personen oder von Firmen (*Corporate Blogs*) unterhalten werden.

Eine Blog-Variante stellen sog. *Podcasts* dar. Analog zu Blogs kann praktisch jeder bestimmte Inhalte auf einer Web-Seite zur Verfügung stellen. Die Inhalte werden bei Podcasts nicht wie bei Blogs in Textform, sondern als Tonaufnahme oder als Video bereitgestellt.

Blogs können neben dem Betrachten mit einem Web-Browser i.d.R. auch in Form von *Feeds* (auch: *News Feeds*, *Web Feeds*) abgerufen werden. Dabei werden die Inhalte in einem XML-Format zur Verfügung gestellt, was deren Weiterverarbeitung stark vereinfacht. Gängige Formate sind RSS (in den Versionen 0.90[34], 0.91[35], 0.92[36], 1.0[37] und 2.0[38]) sowie Atom (in den Versionen 0.3, 0.4, 0.5 und 1.0[39]).

Die Artikelinhalte sind i.d.R. in HTML ausgezeichnet, wobei die Feeds oft nur einen Ausschnitt, gelegentlich auch nur den Titel der Artikel beinhalten. Solche Feeds

---

[30] Der Heise-Online-Newsticker kann über `http://www.heise.de/newsticker/` (Abruf: 2007-08-15) erreicht werden.

[31] Aktuelle Nachrichtenartikel aus der Redaktion des Spiegels findet man unter `http://www.spiegel.de/` (Abruf: 2007-08-15).

[32] Blogcensus.de (`http://www.blogcensus.de/`) zählte am Abrufdatum (2007-08-28) 110.033 deutschsprachige Blogs.

[33] Blogs wurden von Jean-Remy von Matt als „Klowände des Internets“ bezeichnet. Für die E-Mail, die diese Behauptung enthält, siehe Scholz (2007).

[34] Für die Spezifikation zu RSS 0.90 siehe Netscape (1999).

[35] Für die Spezifikationen zu RSS 0.91 siehe UserLand Software (2000a).

[36] Für die Spezifikationen zu RSS 0.92 siehe UserLand Software (2000b).

[37] Für weitere Informationen sowie die Spezifikation zu RSS 1.0 siehe RSS-DEV Working Group (2000).

[38] Für die Spezifikation der aktuellen Revision von RSS 2.0 siehe RSS Advisory Board (2006).

[39] Die Spezifikation von Atom 1.0 wurde von der IETF als „Proposed Standard“ in RFC 4287 (siehe Nottingham u. Sayre (2005)) veröffentlicht. Ältere Atom-Versionen werden nicht offiziell unterstützt.

werden *Partial Feeds* genannt, im Gegensatz zu den *Full Feeds*, die den vollständigen Artikel beinhalten. Den vollständigen Inhalt erhält man bei Partial Feeds auf der verlinkten Web-Seite im HTML-Format.

Feeds werden nicht nur zur Verbreitung von Blog-Inhalten, sondern auch auf Nachrichtenportalen oder für eine Vielzahl anderer, regelmäßig erstellter Informationen verwendet.

Im Anhang in Abschnitt 9.1.2 sind neben weiteren technischen Details einige Beispiele für populäre Blogs zu finden.

2.4.3 Bilder

Illustrationen und Fotos lockern textuelle Inhalte nicht nur auf, sie geben auch zusätzliche Informationen und erlauben es dem Leser so, sich ein genaueres Bild des beschriebenen Sachverhalts zu machen.

Oft liefern Nachrichtenagenturen die Meldungen schon mit Bildern aus, wie weiter oben in Abschnitt 2.4.1 beschrieben wurde. Es gibt aber auch (kostenlose sowie kommerzielle) Bilddatenbanken, die unabhängig von bestimmten Meldungen Bilder anbieten. Diese Bilder können zur Anreicherung von Artikeln genutzt werden, die evtl. keine Bilder enthalten.

Die Bilder sind oft entsprechend ihrem Inhalt mit Schlagworten versehen, was eine Zuordnung zu den Artikeln erleichtert. Jedoch können die Schlagworte allein nicht garantieren, dass ein Bild auch tatsächlich für einen bestimmten Artikel geeignet ist. Ob eine automatische Zuordnung dieser Bilder zu textuellen Inhalten mit angemessener Genauigkeit möglich ist, kann im Rahmen dieser Arbeit nicht beantwortet werden.

Die Bilddatenbanken sind i.d.R. darauf ausgelegt, in Redaktionssystemen eingebunden oder manuell über Web-Seiten abgefragt zu werden. Trotzdem sollen sie nicht unerwähnt bleiben. Abschnitt 9.1.3 gibt einige Beispiele.

2.4.4 Werbung

Gerade in kommerziellen Anwendungen eines IIDs spielt **Werbung** eine wichtige Rolle. Wie schon in Abschnitt 2.2.2 erwähnt wurde, ermöglicht die Individualisierung eine wesentlich zielgerichtetere Werbung.

Werbeanzeigen werden häufig direkt von den Werbenden bzw. den beauftragten Werbeagenturen gebucht. Dazu kann man eine Web-Anwendung anbieten, über die

ein Werbender seine Anzeigen inkl. Angaben z.B. über die Zielgruppe sowie das Budget einstellen kann. Natürlich ist dieser Prozess auch über eine manuelle Bearbeitung der Buchungen denkbar.

Insbesondere im Internet dominieren automatisierte Werbesysteme, bei denen die Werbung über einen Vermittler platziert wird. Die Werbenden registrieren ihre Anzeigen bei diesem Vermittler, die Anbieter der Inhalte stellen dem Vermittler eine Werbefläche zur Verfügung. Der Vermittler platziert dann die Anzeigen in den zur Verfügung gestellten Werbeflächen. Es besteht somit kein direkter Kontakt mehr zwischen Werbenden und den Anbietern der Inhalte. Ein Beispiel für dieses Prinzip stellt Google AdSense[40] dar. Dieses Prinzip ist jedoch nicht nur auf das Internet beschränkt: Google schaltet für seine Kunden im Rahmen des Dienstes Print Ads[41] Werbung in US-Zeitungen.

Eine spezialisierte Form der Werbung stellen Kleinanzeigen dar, in denen i.d.R. einzelne Objekte zum Verkauf oder zur Miete angeboten werden. Kleinanzeigen können analog zu Werbeanzeigen direkt oder über einen Vermittler geschaltet werden. Sie sprechen jedoch eine wesentlich kleinere Zielgruppe an, als es bei Werbeanzeigen der Fall ist. Dementsprechend unterscheiden sie sich auch deutlich in Form und Größe, sowie den damit verbundenen Kosten. Auch hier kann man dem Abonnenten eine individuell auf ihn zugeschnittene Auswahl an Anzeigen anbieten und somit Streuverluste minimieren.

Die Anzeigen liegen üblicherweise in Bild- (Werbeanzeigen) oder Textform (Kleinanzeigen) vor. Falls die konkrete Anwendung eines IIDs die Informationen über das Internet anbietet, so sind zusätzlich Ton- oder Videoaufnahmen als Medium denkbar.

Im Anhang werden in Abschnitt 9.1.4 die Möglichkeiten, in einem IID Werbung zu schalten, inkl. zusätzlicher technischer Informationen kurz zusammengefasst.

2.4.5 Persönliche Assistenzen

Unter **persönlichen Assistenzen** werden hier Informationen verstanden, deren Auswahl der Abonnent unmittelbar durch Definition bestimmter Kriterien beeinflusst. So könnte er sich die Wetterprognosen für eine bestimmte Region, die Kurse

[40] Die Beschreibung des Dienstes Google AdSense kann unter `http://www.google.com/adsense/` (Abruf: 2007-08-16) eingesehen werden.

[41] Die Beschreibung des Dienstes Google Print Ads findet man unter `http://www.google.com/adwords/printads/` (Abruf: 2007-08-16).

einer Menge von Aktien oder aktuelle Wohnungsangebote einer bestimmten Gegend mit einer bestimmten Größe innerhalb einer bestimmten Preisspanne anzeigen lassen.

Im Gegensatz zu den anderen Quellen, bei denen die Auswahl der Informationen durch das Interessenprofil des Abonnenten geschieht, hat der Abonnent hier also die Möglichkeit, die Informationen explizit durch Definition bestimmter Kriterien auszuwählen. Die Inhalte der Persönlichen Assistenzen sind meist durch eine recht hohe Dynamik gekennzeichnet.

Man kann aber keine scharfe Grenze zwischen persönlichen Assistenzen und den zuvor genannten Quellen ziehen. So könnte ein Abonnent Inhalte eines bestimmten Blogs explizit auswählen (oder ausschließen) – die Auswahl wird also unmittelbar durch eine Konfiguration beeinflusst. Ebenso könnte der Ort für Wetterprognosen automatisch aus seinen Stammdaten abgeleitet werden, möglicherweise interessante Artikel aus Shops oder Marktplätzen könnten anhand seines Interessenprofils ohne eine explizite Angabe von zusätzlichen Kriterien ausgewählt werden. Man kann für eine bestimmte Quelle in vielen Fällen also nicht klar sagen, ob sie ausschließlich als persönliche Assistenz dient oder ob sie auch anderweitig eingesetzt werden kann.

In Abschnitt 9.1.5 werden einige mögliche Persönliche Assistenzen dargestellt. Tatsächlich sind aber noch viele weitere Assistenzen denkbar, wie z.B. Vorschläge von TV-Sendungen oder Kulturveranstaltungen entsprechend der individuellen Interessen, die Lottozahlen der gespielten Systeme, etc..

Es ist ebenfalls denkbar, bestehende Kommunikationsdienste wie E-Mail, Foren oder Newsgroups als persönliche Assistenz einzubinden. Eine vollständige Integration inkl. Rückkanal (also auch zum Schreiben von Nachrichten) wird unter Beachtung des erforderlichen Entwicklungsaufwands nicht sinnvoll sein. Bei einer Integration ohne Rückkanal ist man auf zusätzliche Dienste oder Programme (E-Mail-Client, News-Reader, Web-Browser) angewiesen.

2.4.6 Auswahl in dieser Arbeit anzubindender Quellen

Im Rahmen dieser Arbeit wird nur eine begrenzte Menge an Quell-Komponenten implementiert werden können. Um trotzdem ein repräsentatives Ergebnis zu erhalten, wird mit Ausnahme der Werbung von jedem Quell-Typen mindestens eine Quelle angebunden.

Konkret werden Nachrichten aus den Diensten AFP Multimedia, dpa-Webline und Heise-Online integriert. Der Grund für diese Auswahl liegt vor allem an der Verfügbarkeit von Beispieldaten, die bei der Entwicklung genutzt werden können.

Zur Anbindung von Blogs wird eine generische Feed-Komponente entwickelt werden, die möglichst alle gängigen Feed-Formate verarbeiten kann. Bilderdienste werden aufgrund der oben erwähnten Zweifel an der Qualität einer automatisierten Zuordnung zu Artikeln nicht angebunden. Allerdings werden Bilder, die in Nachrichtenartikeln referenziert werden und somit über eine klare Zuordnung verfügen durchaus in die Integration mit einbezogen. Als Persönliche Assistenz soll sollen Aktienkurse von Yahoo Finance eingebunden werden.

Für Werbeanzeigen wird keine Quell-Komponente entwickelt. Vermittelte Werbung auf Web-Seiten wird direkt durch den Vermittler in die Seiten eingebettet und entzieht sich somit einer Integration. Für durch den Werbenden direkt schaltbare Werbung kann man als Anbieter der Werbefläche das Format praktisch selbst bestimmen. Eine Integration erscheint auch hier nicht notwendig.

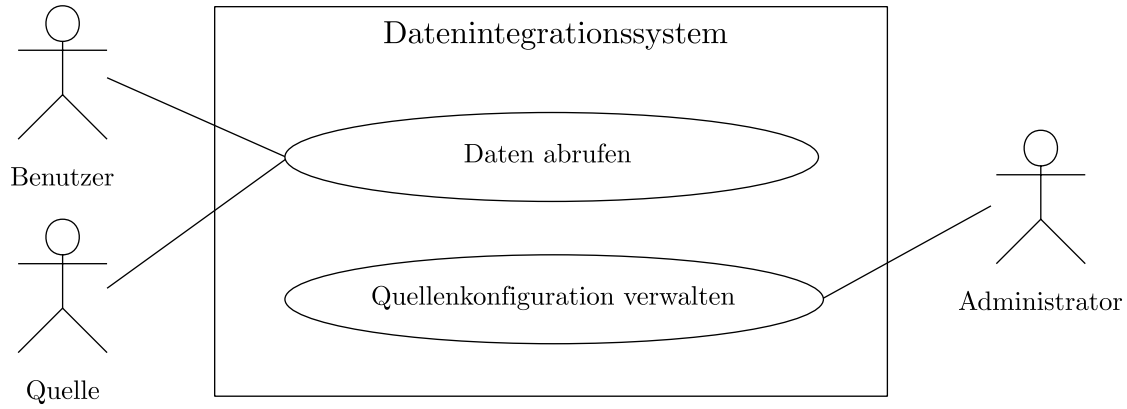
3 Anforderungen

3.1 Anwendungsfälle

In der Objektorientierten Analyse (OOA) werden Anwendungsfälle genutzt, um die Möglichkeiten der Benutzung eines Systems zu dokumentieren. Die Anwendungsfälle enthalten die komplette Funktionalität der Software und dienen somit oft als Ersatz für die traditionellen funktionalen Anforderungen.[42]

Die Interaktionen mit dem in dieser Arbeit zu entwickelnden Integrationssystem (im Folgenden auch synonym „System“ genannt) sind sehr einfach. Wie man Abbildung 3-5 entnehmen kann, gibt es nur zwei Anwendungsfälle.

Abb. 3-5: UML-Anwendungsfalldiagramm für das Integrationssystem



Der **Benutzer** nutzt das Integrationssystem[43], um **Daten abzurufen**. Dabei muss das Integrationssystem evtl. auf die **Quellen** zugreifen, die die Rohdaten liefern. Diese Daten müssen aufbereitet werden und werden dem Benutzer dann in einer einheitlichen Form übergeben.

Zusätzlich hat ein **Administrator** die Möglichkeit, die **Quellenkonfiguration zu verwalten**. Dies umfasst, wie schon in Abschnitt 2.2.3 beschrieben, das Hinzufügen neuer oder Entfernen vorhandener Komponenten zur Anbindung bestimmter Quellen, sowie die technische Konfiguration (Adressen, Zugangsdaten, Abrufintervalle etc.) der Quellen.

Aus Benutzersicht ist die Heterogenität der Quellen also idealerweise nicht sichtbar. Sie ist ein technisches Detail, das für den Benutzer nicht von Relevanz ist und auf

[42] Vgl. Cockburn2000, S. 13.

[43] Der **Benutzer** des Integrationssystems wird in der Regel der IID sein, da das Integrationssystem eine Teilkomponente des IIDs ist. Er ist nicht mit dem **Abonnenten** (Siehe Abschnitt 2.1.3) zu verwechseln, der mit dem Integrationssystem nicht direkt interagiert.

dem hohen Abstraktionsniveau der Anwendungsfälle üblicherweise nicht modelliert wird. Trotzdem (oder gerade weil) die Heterogenität aus Sicht des Benutzers idealerweise nicht sichtbar ist, ist sie ein zentraler Aspekt, der in den Anforderungen festgehalten werden muss.

Diese teilweise sehr technischen Anforderungen können in der erforderlichen Präzision nur sehr schwer in Form von Anwendungsfällen erfasst werden. Wesentlich einfacher und vor allem kompakter ist die Definition der Anforderungen in diesem Kontext mittels natürlicher Sprache. Daher wird auf eine Verfeinerung und eine detaillierte Beschreibung der Anwendungsfälle verzichtet und die Gesamtheit der Anforderungen an die zu entwickelnde Komponente wird im Folgenden traditionell in Form von Anforderungslisten dargestellt.

3.2 Überblick

Die Definition eines IIDs (siehe Abschnitt 2.2.1) fordert den Zugriff auf heterogene Quellen:

> **Anforderung 1:** Der Abruf von Daten aus mehreren, möglicherweise heterogenen, digitalen Quellen **muss** möglich sein.

Die von den Quellen angebotenen Daten können in sehr unterschiedlichen Formaten vorliegen:

> **Anforderung 2:** Es **muss** berücksichtigt werden, dass die Daten von den Quellen in heterogenen Austauschformaten angeboten werden.

Der integrierte Zugriff auf die vom Integrationssystem bereitgestellten Daten wird, wie in Abschnitt 2.3.4 vorgeschlagen wurde, idealerweise über eine einfache Schnittstelle ermöglicht:

> **Anforderung 3:** Zum Zugriff auf die Daten **muss** durch das Integrationssystem eine Schnittstelle zur Verfügung gestellt werden, die die Komplexität des Zugriffs auf die heterogenen Quellen kapselt.

Die Daten werden durch zahlreiche Komponenten verarbeitet (siehe Abschnitt 2.2.4). Die durch die Heterogenität der Daten entstehende Komplexität der Verarbeitung muss minimiert werden:

> **Anforderung 4:** Die durch die Vielzahl an Formaten, in denen die Daten vorliegen können, entstehende Komplexität der Verarbeitung **muss** durch Transformationen im Integrationsprozess so weit wie möglich reduziert werden.

Dies ermöglicht zudem die in der Definition eines IIDs verlangte einheitliche Sicht auf die Daten sowie Bereitstellung der Daten über unterschiedliche Ausgabemedien.

Da man ex-ante kaum vorhersehen kann, welche Quellen ein IID alle einbinden wird, muss die Architektur des Integrationssystems die Anbindung weiterer Quellen ermöglichen:

Anforderung 5: Die Architektur des Systems **muss** flexibel genug sein, um auch nachträglich die Anbindung weiterer Informationsquellen zu erlauben.

Entsprechend den im vorigen Abschnitt identifizierten Anwendungsfällen muss eine Möglichkeit zur Verwaltung der Quellen bestehen:

Anforderung 6: Es **muss** möglich sein, die Quellen über eine Konfiguration zu verwalten.

In den nachfolgenden Abschnitten werden diese, noch sehr allgemein gehaltenen, Anforderungen verfeinert. Der Schwerpunkt liegt auf jenen Anforderungen, die sich aus der Heterogenität der Daten (Abschnitt 2.3.2) und der Quellen (Abschnitt 2.3.3) ergeben, da dort besonders viele externe Einflüsse berücksichtigt werden müssen. Wo es nötig ist, werden die Anforderungen erläutert oder durch Beispiele illustriert.

Ziel ist es nicht, das System bis in die letzten Details zu spezifizieren. Insbesondere werden einige nicht-funktionale Anforderungen wie Vorgaben für Antwortzeiten oder die Verfügbarkeit des Systems[44] nicht spezifiziert, da sie für die Implementierung eines Prototypen von nachrangiger Bedeutung sind. Es soll allerdings klar definiert werden, welche Funktionalitäten ein solches System implementieren muss und mit welchen Situationen es zurechtkommen soll.

3.3 Heterogene Informationsquellen

Anforderung 1 (S. 26) verlangt, dass das System mehrere heterogene Quellen zur Informationsgewinnung nutzen kann. Die Kriterien zur Unterscheidung der Quellen wurden in Abschnitt 2.3.3 identifiziert. Diese möglichen Heterogenitäten müssen in die Anforderungen einfließen. Sie werden im Folgenden einschließlich einer der Bedeutung für einen IID entsprechenden Priorität festgehalten.

Übertragung

Betrachtet man die **Übertragung** der Daten, so kann man die Quellen zunächst nach den verwendeten **Protokollen** unterscheiden:

Anforderung 1.1: Der Zugriff auf die Quellen über unterschiedliche Protokolle **muss** möglich sein.

Auch können die Quellen **Schnittstellen** unterschiedlichster Art anbieten, die für den Zugriff genutzt werden müssen:

---

[44] Antwortzeiten und Verfügbarkeit werden lediglich als **externe** zu berücksichtigende Einflüsse beim Zugriff auf die Quellen berücksichtigt.

Anforderung 1.2: Der Zugriff auf die Quellen über unterschiedliche Schnittstellen **muss** möglich sein.

Abruf

Auf einer höheren Ebene kann man verschiedene **Abrufmöglichkeiten** unterscheiden.

Der Zugriff über **Abfragesprachen** wird in einem IID eher die Ausnahme als die Regel sein, da man üblicherweise keinen direkten Zugriff auf die Datenbanken der Informationslieferanten bekommt. Daher ist diese Art des Zugriffs von untergeordneter Bedeutung:

Anforderung 1.3: Der Zugriff auf Quellen über Abfragesprachen wie SQL oder XQuery **kann** ermöglicht werden.

Eine weitere Möglichkeit des Zugriff auf die Daten stellen **Funktionsaufrufe** dar, die dem Informationsanbieter durch eine begrenzte Anzahl an möglichen Abfragen eine größere Sicherheit bieten. Diese Form des Zugriffs wird daher eine wesentlich höhere Relevanz als der direkte Zugriff über Abfragesprachen haben:

Anforderung 1.4: Der Zugriff auf Quellen über parametrisierte Funktionsaufrufe, wie z.B. Web-Services oder RPC-Techniken, **muss** ermöglicht werden.

Mindestens ebenso wichtig sind Quellen, die keine gezielten Abfragen über Abfragesprachen oder Funktionen erlauben:

Anforderung 1.5: Der Zugriff auf Quellen, die keine gezielte Abfragen erlauben, **muss** über regelmäßige, einfache Abrufe ermöglicht werden. Dabei **muss** berücksichtigt werden, dass diese Quellen möglicherweise nur kleine Ausschnitte (oft die neuesten Informationen) ihres Datenbestandes anbieten.

Verhalten

Auch in ihrem **Verhalten** können sich die Quellen stark unterscheiden. Die **Dynamik** der Informationen ist ein sehr wichtiges Kriterium:

Anforderung 1.6: Es **muss** berücksichtigt werden, dass sich die Informationen, die eine Quelle zur Verfügung stellt, sowohl sehr häufig als auch sehr selten ändern können.

Eine mangelnde **Verfügbarkeit** der Quellen kann das Informationsangebot eines Informationsdienstes einschränken. Dies sollte nach Möglichkeit vermieden werden:

Anforderung 1.7: Ein Zugriff auf Daten aus Quellen, die eine geringe Verfügbarkeit aufweisen, **soll** durch geeignete Maßnahmen auch im Fehlerfalle möglich sein.

Sehr hohe **Antwortzeiten** können den Verarbeitungsprozess stark verlangsamen, was nicht wünschenswert ist:

> **Anforderung 1.8:** Quellen, die sehr hohe Antwortzeiten haben, **sollen** den Verarbeitungsprozess nicht übermäßig verlangsamen.

Der **Umfang** der von einer Quelle angebotenen Informationen kann stark variieren und muss bei der Entwicklung berücksichtigt werden:

> **Anforderung 1.9:** Der Zugriff auf sehr umfangreiche Quellen **muss** möglich sein.

Die **Änderbarkeit** der von einer Quelle angebotenen Informationen ist für einen IID nicht von Relevanz und wird daher auch nicht in den Anforderungen berücksichtigt.

3.4 Heterogene Daten

Die von dem Integrationssystem aus den Quellen abgerufenen Daten können, wie in Abschnitt 2.3.2 dargestellt wurde, in sehr unterschiedlichen Austauschformaten vorliegen. Das Integrationssystem muss also eine Vielzahl an Eingabeformaten verarbeiten können.

Viele potenzielle Quellen stellen ihre Daten in **strukturierter** Form bereit (z.B. Nachrichtenartikel im NewsML-Format, News-Feed im RSS- oder Atom-Format, Dienste mit Web-Services-Schnittstellen). Der Zugriff auf solche Daten ist unverzichtbar für einen IID:

> **Anforderung 2.1:** Die Verarbeitung von Austauschformaten, die Daten in strukturierter Form enthalten, **muss** möglich sein.

Ebenso können viele nützliche Informationen (z.B. Web-Inhalte im HTML-Format) in Form **semi-strukturierter Daten** vorliegen. Ein großer Teil potenzieller Informationen kann durch die Einbindung semi-strukturierter Daten erschlossen werden. Deren Verarbeitung ist jedoch mit einem höheren Aufwand verbunden, sodass eine Einbindung solcher Quellen nicht pauschal verlangt werden kann:

> **Anforderung 2.2:** Die Verarbeitung von Austauschformaten, die Daten in semi-strukturierter Form enthalten, **soll** möglich sein.

Unstrukturierte Daten sind in einem IID nur in Form von Multimedia-Inhalten (z.B. Fotos) relevant:

> **Anforderung 2.3:** Die Verarbeitung von Austauschformaten, die Daten in unstrukturierter Form enthalten, **muss** nur für Multimedia-Inhalte (Foto, Audio, Video) möglich sein.

Die einzelnen zu unterstützenden Austauschformate (RSS, HTML, JPEG etc.) werden an dieser Stelle nicht spezifiziert. Welche Austauschformate im Detail benötigt werden hängt von der konkreten Ausprägung eines IIDs und den anzubindenden Quellen ab. Eine Übersicht über technische Details möglicher Quellen findet man im Anhang in Unterkapitel 9.1.

3.5 Schnittstelle zur Abfrage der Daten

In Anforderung 3 (S. 26) wird eine transparente Schnittstelle gefordert, die die Komplexität des Zugriffs auf die heterogenen Quellen verbirgt.

Diese Schnittstelle sollte möglichst generisch sein, d.h. sie sollte im Idealfall alle nötigen Abfragen erlauben, ohne für jede neue Quelle angepasst werden zu müssen:

> **Anforderung 3.1:** Die Schnittstelle zur Abfrage der Daten **soll** generisch genug sein, um darüber Daten aus allen Quellen abfragen zu können.

Wo das nicht möglich ist, müssen alternative Schnittstellen angeboten werden:

> **Anforderung 3.2:** Falls Daten einer Quelle nicht über die generische Schnittstelle zum Abruf zugreifbar sind, so **muss** dafür eine alternative, spezifische Schnittstelle angeboten werden.

3.6 Vereinheitlichung der Formate

Die vom Integrationssystem gelieferten Daten müssen gemäß Anforderung 4 (S. 26) in einer leicht zu verarbeitenden Form vorliegen, um den Verarbeitungsaufwand zu reduzieren.

Das Ziel ist es dabei, die Verarbeitung der Daten durch eine Minimierung der syntaktischen und strukturellen Heterogenität[45] zu vereinfachen:

> **Anforderung 4.1:** Die syntaktische und strukturelle Heterogenität **muss** durch eine Transformation der Daten minimiert werden.

Jedoch sollen Strukturen, die Aufschluss über die Semantik der Informationen geben, soweit wie möglich erhalten bleiben. Transformiert man Daten aus einem Datenmodell in ein anderes, das für bestimmte Strukturelemente des Quellformats keine äquivalenten Elemente vorsieht, so gehen evtl. Informationen über die Semantik der betroffenen Ausschnitte der Daten verloren.

Ein einfaches Beispiel dafür ist die Transformation folgender XML-Daten:

---

[45] Unter syntaktischer Heterogenität versteht man Unterschiede in der Darstellung der Informationen. Unter struktureller Heterogenität versteht man die strukturell unterschiedliche Repräsentation gleicher Konzepte. Vgl. dazu Leser u. Naumann (2006), S. 64 ff.

```
<bestellung>
    <position>
        <name>ABC</name>
        <preis waehrung="Euro">23</preis>
    </position>
    <position>
        <name>DEF</name>
        <preis waehrung="Dollar">42</preis>
    </position>
</bestellung>
```

in eine solche HTML-Repräsentation:

```
<ul>
    <li>ABC - 23 Euro</li>
    <li>DEF - 42 Dollar</li>
</ul>
```

In der HTML-Repräsentation gehen wesentliche Informationen verloren. Es ist nicht mehr klar, dass die Liste ein Ausschnitt aus einer Bestellung darstellt. Die Unterscheidung zwischen Name, Preis und Währung ist nicht mehr durch Strukturelemente möglich, sondern nur noch durch Interpretation.

Diesen Verlust an Semantik gilt es so weit wie möglich zu vermeiden:

> **Anforderung 4.2:** Die Semantik, die durch die Struktur der Informationen widergespiegelt wird, **soll** durch die Transformation nicht verloren gehen.

Zudem muss der Transformationsprozess um neue Quell- und Zielformate erweiterbar sein:

> **Anforderung 4.3:** Es **muss** möglich sein, den Transformationsprozess um neue Austauschformate zu erweitern.

3.7 Flexible Architektur

Anforderung 5 (S. 27) fordert eine hinreichende Flexibilität des Systems, um auch nachträglich zusätzliche Quellen anbinden zu können.

Dazu muss das System zunächst entsprechende Schnittstellen zur Verfügung stellen:

> **Anforderung 5.1:** Das System **muss** Schnittstellen zur Verfügung stellen, um Komponenten anzubinden, die auf zusätzliche Quellen zugreifen.

Idealerweise sind solche Komponenten integrierbar, ohne das Kernsystem verändern zu müssen:

Anforderung 5.2: Zusätzliche Komponenten zur Anbindung weiterer Quellen **sollen** ohne Modifikationen am Kernsystem integrierbar sein.

3.8 Verwaltung der Quellen

Anforderung 6 (S. 27) fordert, dass bestimmte Parameter der Komponenten über eine Konfiguration verwaltbar sein müssen. Die Verwaltung umfasst im Wesentlichen die Installation der Komponenten sowie die Einstellung bestimmter Parameter:

Anforderung 6.1: Die Komponenten zur Anbindung der Quellen **sollen** ohne Modifikationen an Quelltexten durch einen Administrator installierbar und de-installierbar sein.

Anforderung 6.2: Die technischen Parameter zum Zugriff auf die Quellen **sollen** ohne Modifikationen an Quelltexten durch einen Administrator konfigurierbar sein.

3.9 Einschränkungen des Prototyps

Ein Ziel der Arbeit ist es, die Erkenntnisse prototypisch in einem Softwaresystem zu implementieren. Ein Prototyp soll funktionstüchtig sein und die Umsetzbarkeit der Konzepte belegen. Er soll helfen, durch die Benutzung des Systems neue Erkenntnisse zu gewinnen, ebenso wie er schnell und günstig entwickelt werden soll.[46]

Man kann Prototypen in zwei wesentliche Klassen einordnen:[47] **Wegwerfprototypen** (*Rapid Prototyping*) zur Sammlung von Erfahrungen und **wiederverwendbare Prototypen** (*Evolutionäres Prototyping*), die schrittweise verbessert werden und (teilweise) in das endgültige System eingehen. Im Rahmen dieser Arbeit wird die letztere Form eines Prototypen entwickelt.

Der Prototyp soll die weiter oben beschriebenen Anforderungen erfüllen. Es wird ein lauffähiges Basissystem entwickelt. Jedoch kann und soll der zu entwickelnde Prototyp nicht alle denkbaren Einsatzmöglichkeiten abdecken. Insbesondere wird er nur eine begrenzte Anzahl an Quellen und Austauschformaten einbinden. Ebenso wird keine grafische Benutzeroberfläche zur Konfiguration oder Bedienung der Software entwickelt. Es soll aber möglich sein, den Prototypen durch nachträgliche Entwicklung weiterer Komponenten erweitern zu können.

---

[46] Für eine kurze Beschreibung prototypischer Software-Entwicklung vgl. Pomberger u. a. (1991), S. 44–45.

[47] Zur Klassifizierung von Prototypen vgl. Stahlknecht u. Hasenkamp (2004), S. 219–221.

4 Lösungsansatz

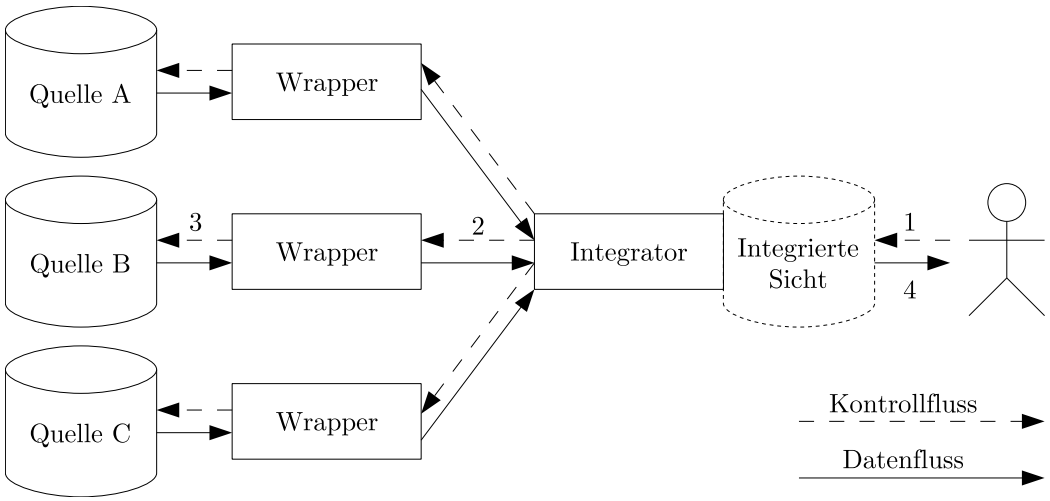
4.1 Datenintegration als Lösung für die Anforderungen

4.1.1 Ansätze zur Datenintegration

Die oben identifizierten Anforderungen sollen im Rahmen dieser Arbeit durch ein Softwaresystem zur Datenintegration gelöst werden. Man kann zwei wesentliche Ansätze der Datenintegration unterscheiden:[48] **virtuelle Integration** (auch:; *On-Demand Retrieval*, *Data Mediation*, *Mediated Query*) und **materialisierte Integration** (auch: *In-Advance Retrieval*, *Data Warehousing*).

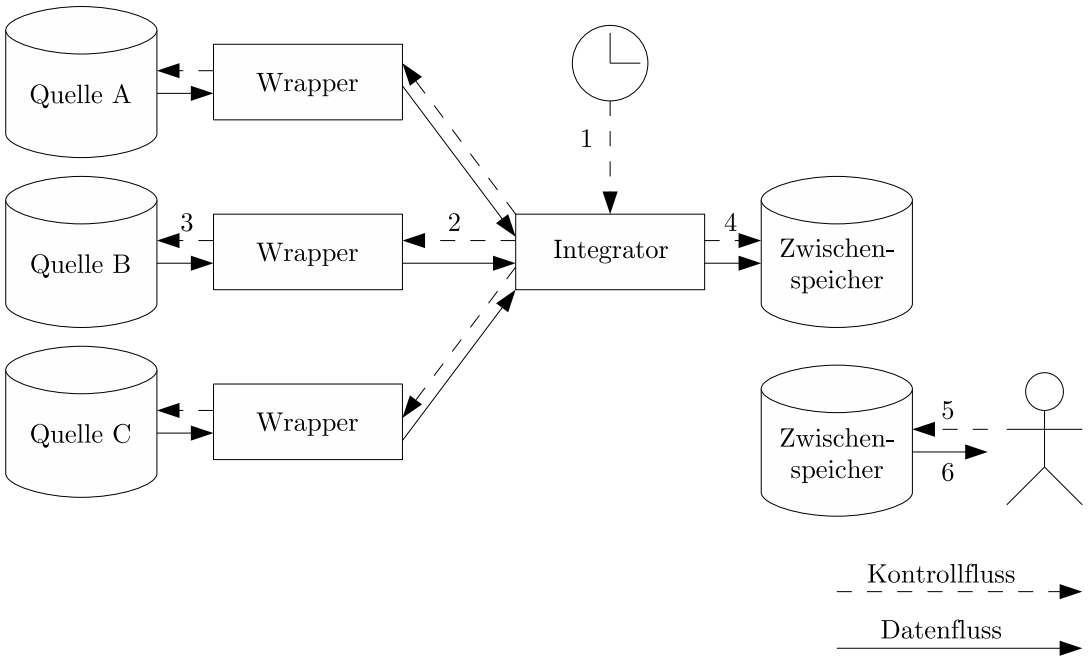
Virtuelle Integration

Abb. 4-6: Ablauf der virtuellen Integration



Den ersten Ansatz veranschaulicht Abbildung 4-6. Der Benutzer stellt eine Abfrage über die integrierte Sicht an das Integrationssystem (Schritt 1). Der Integrator ermittelt nun die für die Beantwortung der Abfrage nötigen Quellen, übersetzt die Abfrage in Teilabfragen an die jeweiligen Quellen und setzt diese Teilabfragen nun an die Wrapper für die jeweiligen Quellen ab (Schritt 2). Die **Wrapper** dienen dazu, den Zugriff auf eine Quelle zu ermöglichen. Sie verarbeiten die Teilabfrage und beantworten sie mit den Daten, die in der jeweiligen Quelle zur Verfügung stehen (Schritt 3). Die Teilantworten werden dem Integrator übergeben, der sie übersetzt, filtert und mit den anderen Teilantworten zu einer Gesamtantwort zusammenfügt. Diese Gesamtantwort wird schließlich dem Benutzer übermittelt (Schritt 4).

Abb. 4-7: Ablauf der materialisierten Integration



Materialisierte Integration

Bei einer materialisierten Integration, die in Abbildung 4-7 dargestellt wird, veranlasst der Integrator zu regelmäßigen Zeitpunkten (Schritt 1) die einzelnen Wrapper (Schritt 2), die Daten, die von Bedeutung sein könnten, aus den Quellen zu extrahieren (Schritt 3). Diese Daten werden durch den Integrator übersetzt, gefiltert und, falls sinnvoll, mit Daten anderer Quellen vereint. Allerdings werden die Daten nicht als Ergebnis einer Abfrage an den Benutzer zurückgegeben, sondern zunächst in einem Zwischenspeicher abgelegt bzw. **materialisiert** (Schritt 4). Diese Vorgehensweise wird i.d.R. in Data Warehouses angewendet und wird dort auch unter der Bezeichnung **Extract, Transform, Load (ETL)** beschrieben. Wenn der Benutzer nun eine Abfrage tätigt (Schritt 5), wird sie unmittelbar aus dem Zwischenspeicher beantwortet (Schritt 6), ohne die ursprünglichen Quellen abzufragen.

Vor- und Nachteile

Beide Ansätze haben Vor- und Nachteile und sind dementsprechend in unterschiedlichen Situationen zu bevorzugen.

Eine **virtuelle Integration** ist vor allem dann **vorteilhaft**, wenn sich die Quelldaten häufig verändern oder wenn die Quelldaten zu umfangreich sind, um sie zwischen-

[48] Vgl. hierzu und zu den folgenden Absätzen Widom (1995), S. 25 und Domenig u. Dittrich (1999), S. 64.

zuspeichern. **Problematisch** ist die virtuelle Integration vor allem dann, wenn die Quellen hohe Antwortzeiten oder eine geringe Verfügbarkeit haben. Sie kann auch unvorteilhaft sein, wenn sehr aufwändige Übersetzungen der Abfragen und/oder Daten zur weiteren Verwendung nötig sind.

Die **materialisierte Integration** bietet vor allem **Vorteile** durch eine schnellere und einfachere Verarbeitung der Abfragen sowie eine höhere Zuverlässigkeit, da man hier nicht darauf angewiesen ist, dass die Quellen permanent verfügbar sind. Außerdem macht dieser Ansatz es erst möglich, Quellen anzubinden, die Ad-hoc-Abfragen über den gesamten Datenbestand nicht unterstützen. Flüchtige Daten können für einen späteren Zugriff vorgehalten werden, ebenso können bestimmte Daten (z.B. Börsenkurse, Wechselkurse, Wetterdaten) zu mehreren Zeitpunkten gespeichert werden, falls deren Ausprägungen im Zeitverlauf von Interesse sind. Schwierig abfragbare Quellen werden durch die materialisierte Integration leichter zugänglich. Da die Daten im Zwischenspeicher repliziert werden, können sich allerdings **Probleme** bei Quellen ergeben, die sehr umfangreich sind oder deren Daten sich sehr häufig ändern.

4.1.2 Abgrenzung zum Integrationsprozess in einem individualisierbaren Informationsdienst

Die in der Literatur beschriebenen Verfahren können nicht unverändert in einen IID übernommen werden. Es ergeben sich unterschiedliche Anforderungen durch die abweichenden Anwendungsbereiche gegenüber einem IID.

Systeme zur virtuellen Integration haben oft den Anspruch, möglichst universell einsetzbar zu sein und versuchen unter großem Aufwand, komplexe Abfragen über mehrere Quellen hinweg zu ermöglichen.[49] Die Abfragen, die in einem IID auftreten, sind jedoch wesentlich einfacher. Wie weiter unten in Abschnitt 4.3.2 gezeigt wird, sind einfache Selektionen ausreichend.

Die wohl populärste Anwendung einer materialisierten Integration ist das Data Warehousing. Data-Warehousing-Systeme werden hauptsächlich für die Datenanalyse und Entscheidungsunterstützung (OLAP, Data Mining, DSS) eingesetzt.[50] Sie benötigen dazu historische (also die Datensätze zu unterschiedlichen Zeitpunkten) und oft quantitative (Umsätze, Anzahl von Transaktionen etc.) Daten und führen häufig sehr starke Aggregationen durch. Ein IID hat jedoch ganz andere Aufgaben.

---

[49] Beispiele für solche Systeme findet man in Popa u. a. (2002), Manolescu u. a. (2001) oder Roth u. Schwarz (1997).

[50] Für einen Überblick über Data Warehouses und OLAP-Technologien siehe Chaudhuri u. Dayal (1997), S. 67–71.

Er ist kein Analysewerkzeug und soll viel mehr eine automatische, abonnentenindividelle Auswahl und Formatierung von Informationen ermöglichen. Es handelt sich i.d.R. nicht um historische oder quantitative Daten. Aggregationen finden nur in Form von Zusammenstellungen der Informationen statt. Unterschiedliche Informationen werden nicht miteinander vermengt. Diese Zusammenstellung ist jedoch nicht mehr Aufgabe der Datenintegration innerhalb eines IIDs, sie wird von nachgelagerten Komponenten (Matching, Layouting) bewerkstelligt.

4.1.3 Adaption der Datenintegration für individualisierbare Informationsdienste

Trotzdem ein IID ganz spezifische Anforderungen an die Verarbeitung der Daten stellt, sind die Konzepte der Datenintegration durchaus nützlich bei der Verwirklichung eines solchen Dienstes. Diese Konzepte müssen aufgrund oben genannter, grundlegender Unterschiede und weiterer Besonderheiten, die weiter unten diskutiert werden, an die Bedürfnisse eines IIDs adaptiert werden.

Vereinigung der Integrationsansätze

Da ein IID sowohl Quellen anbindet, für die eine virtuelle Integration besser geeignet ist (z.B. Wetterprognose, aktueller Kurs einer Aktie), als auch solche, für die eine materialisierte Integration sinnvoller ist (z.B. Nachrichtenartikel, Web-Inhalte), müssen beide Ansätze in der Architektur des Systems berücksichtigt werden.

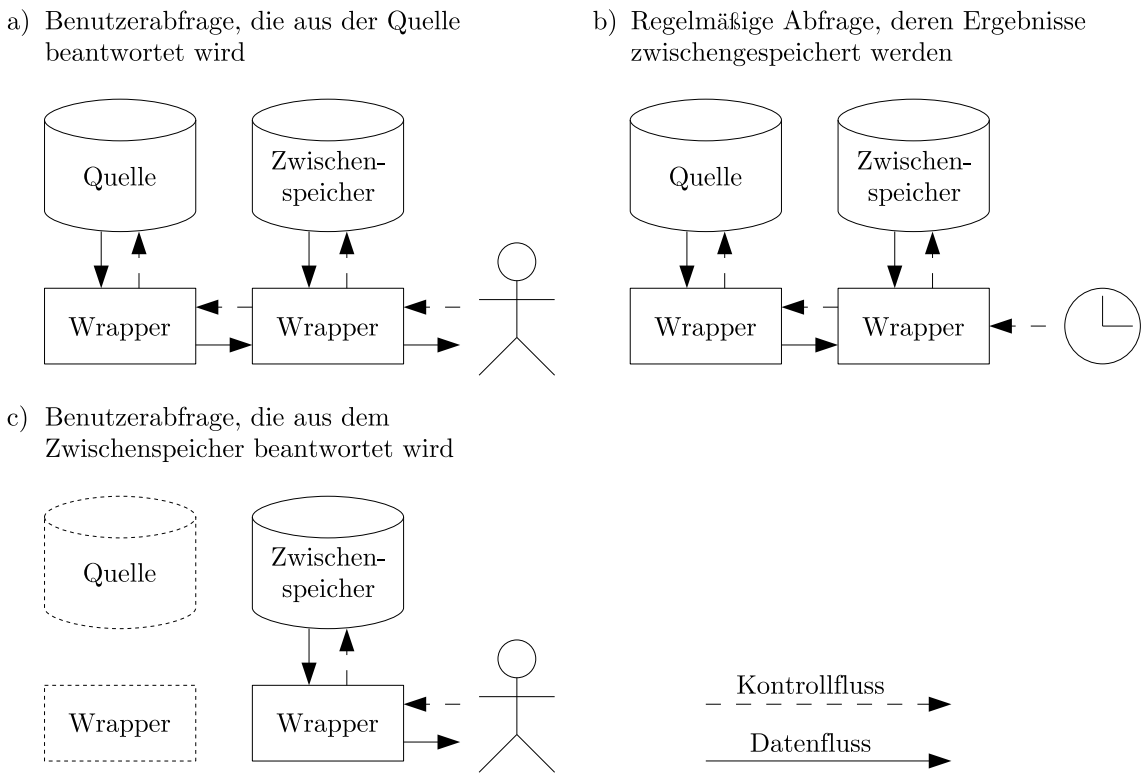
Tatsächlich haben beide Ansätze gewisse Ähnlichkeiten im Verarbeitungsablauf: Beide rufen Daten aus einer Quelle ab (auf Anfrage bzw. regelmäßig), integrieren diese Daten und liefern sie der Abfrage des Benutzer entsprechend aus. Aufgrund der Ähnlichkeit zwischen weiten Teilen beider Ansätze ist es erstrebenswert, die Ansätze in einem gemeinsamen Verfahren zu kombinieren.[51] So kann man zusätzlich in einem gewissen Rahmen die Vorteile beider Ansätze vereinen.

Abbildung 4-8 (S. 37) stellt die Abläufe einer **hybriden Integration** auf einer sehr abstrakten Ebene dar. Es können drei grundsätzliche Abläufe identifiziert werden:

In **Ablauf a)** wird eine Benutzerabfrage entgegengenommen, die der Zwichenspeicher-Wrapper in eine Abfrage an den Zwischenspeicher übersetzt. Falls der Zwischenspeicher die Abfrage nicht beantworten kann, muss sie aus der Quelle beantwortet werden. Dazu wird die Abfrage an den Quell-Wrapper weitergeleitet, der sie in eine Abfrage an die Quelle übersetzt. Die Quelle liefert das Ergebnis an den Quell-Wrapper, der sie an den Zwischenspeicher-Wrapper liefert. Dort wird

[51] Ein möglicher Ansatz zur Kombination der virtuellen mit der materialisierten Integration wird in Hull u. Zhou (1996), S. 484 ff. beschrieben.

Abb. 4-8: Abläufe in einer hybriden Integration



das Ergebnis evtl. im Zwischenspeicher abgelegt und schließlich an den Benutzer geliefert.

Dieser Ablauf entspricht weitgehend dem einer virtuellen Integration mit der Ausnahme, dass hier nur eine Quelle betrachtet wird und dass zusätzlich ein Zwischenspeicher existiert. Dieser kann jedoch ein Dummy sein, der die Abfragen bzw. Ergebnisse einfach zur Quelle bzw. zum Benutzer durchreicht. Der Zwischenspeicher kann allerdings auch nach bestimmten Kriterien (z.B. besonders häufige Abfragen) entscheiden, ob die Ergebnisse zwischengespeichert werden sollen. Mit dieser selektiven Materialisierung kann man einigen Nachteile der virtuellen Integration wie z.B. hohen Antwortzeiten, mangelnder Verfügbarkeit oder einem hohen Verarbeitungsaufwand entgegenwirken.

Ablauf b) stellt eine regelmäßige, i.d.R. zeitgesteuerte Abfrage dar. Der Zwischenspeicher-Wrapper wird aufgefordert, die Daten zu aktualisieren. Dazu schickt er eine entsprechende Abfrage (z.B. Abruf der neuesten Nachrichtenmeldungen) an den Quell-Wrapper. Die Ergebnisse speichert er für eine spätere Verwendung ab, der Benutzer ist an diesem Vorgang nicht beteiligt. Dieser Ablauf entspricht der ersten Phase einer materialisierten Integration. In der hybriden Integration können Abfragen, die typischerweise eine virtuelle Integration vorsehen, um einen

regelmäßigen Abruf und die Zwischenspeicherung bestimmter Daten erweitert werden.

Bei **Ablauf c)** wird eine Benutzerabfrage aus dem Zwischenspeicher beantwortet. Die Quelle muss für diese Abfrage also nicht kontaktiert werden. Dieser Ablauf entspricht der zweiten Phase einer materialisierten Integration. Er ist aber auch auf eine virtuelle Integration mit zusätzlichem Zwischenspeicher anwendbar. Falls die Ergebnisse für eine Abfrage wie in a) oder c) zwischengespeichert wurden, kann sie direkt aus dem Zwischenspeicher beantwortet werden.

Eine reine virtuelle Integration würde Ablauf a) mit einem Dummy-Zwischenspeicher entsprechen. Eine reine materialisierte Integration würde den asynchronen Abläufen b) und c) entsprechen. Wie jedoch oben dargestellt, sind durchaus Mischformen denkbar und sinnvoll.

Eine hybride Integration setzt also einen optionalen Zwischenspeicher ein, der die Abfrage selektiv aus den gespeicherten Daten oder über eine direkte Abfrage der Quelle beantwortet und ggf. speichert. Zusätzlich wird der Zwischenspeicher regelmäßig aufgefordert, seinen Datenbestand zu aktualisieren, was je nach Quellentyp jedoch auch optional sein kann. Quellen, für die eine Zwischenspeicherung nicht in Frage kommt, führen bei der regelmäßigen Aktualisierung schlichtweg keine Operation durch.

Mit einer solchen hybriden Architektur kann für jede Quelle entschieden werden, in welchem Rahmen man die Eigenschaften der virtuellen Integration mit denen der materialisierten Integration kombiniert, um eine optimale Verarbeitung der Daten zu erreichen. Welche Kombinationen in welcher Situation Sinn machen, wird weiter unten in Abschnitt 4.2.3 erläutert.

Teilaufgaben der Integration in einem IID

Der oben in groben Zügen beschriebene Integrationsprozess in einem IID ist nicht trivial und birgt vielfältige Probleme, für die Lösungen gefunden werden müssen.

Zunächst muss das Integrationssystem die Daten aus den Quellen abrufen können. Die Quellen haben, wie schon in Abschnitt 2.3.3 beschrieben, viele Kriterien, in denen sie sich unterscheiden können. Zum leichten Zugriff auf die Quellen muss ein Integrationssystem eine einheitliche Schnittstelle anbieten, über die die Daten abfragbar sind. Die Daten sollen dabei zur einfachen Verwendung in geeignete Formate transformiert und ausgeliefert werden.

Diese Teilaufgaben werden in den nachfolgenden Unterkapiteln genauer untersucht. Die Probleme werden konkretisiert und konzeptionelle Lösungen für diese Teilaufgaben werden erarbeitet.

4.2 Abruf der Daten

4.2.1 Komplexität durch unterschiedliche Quellen

In Unterkapitel 2.4 wurde klar, dass es sehr viele potenzielle Quellen gibt, die man in einem IID anbinden kann. Wie schon in Abschnitt 2.3.3 beschrieben wurde, können diese Quellen sehr heterogen sein. Durch die Kriterien zur Unterscheidung der Quellen entsteht eine Vielzahl an möglichen Fällen, die berücksichtigt werden müssen. Erfreulicherweise sind diese Kriterien jedoch weitgehend unabhängig voneinander und können somit jeweils größtenteils separat betrachtet und gelöst werden.

So kann die Implementierung des Informationsabrufs mittels Volltextsuche unabhängig von der dafür verwendeten Übertragungsschnittstelle geschehen. Man kann die Probleme einzeln lösen und muss für Kombinationen der Probleme keine separaten Lösungen entwickeln. Die Anzahl der nötigen Lösungen ergibt sich additiv aus den Ausprägungen der Kriterien.

Dennoch ist dieser Lösungsaufwand nicht zu verachten. Selbst wenn man davon ausgeht, dass man jede Ausprägung eines Kriteriums unabhängig von den anderen Kriterien lösen kann, so muss immer noch eine Vielzahl an Fällen berücksichtigt werden. Nimmt man in Anlehnung an Abbildung 2-4 (S. 16) 6 Übertragungsschnittstellen, 7 Arten der Informationsabfrage und 2 Ausprägungen je Verhaltenskriterium (Änderbarkeit ausgenommen) an, so ergeben sich schon bei dieser optimistischen Schätzung $6 + 7 + 4 \cdot 2 = 21$ Ausprägungen, die zu berücksichtigen sind. Das ist durchaus ein Faktor, der den Zugriff auf die Daten deutlich erschwert.

4.2.2 Umgang mit dieser Komplexität

In Abschnitt 2.3.3 wurden die Unterscheidungskriterien der Quellen in drei Gruppen unterteilt: Übertragung, Verhalten und Abfrage.

Die **Übertragung**, die das Übertragungsprotokoll sowie die darauf aufbauende Schnittstelle umfasst, kann mit entsprechenden Programmbibliotheken relativ leicht realisiert werden und wird hier daher nicht weiter berücksichtigt.[52] Durch die Abfrage sowie das Verhalten der Quellen ergeben sich jedoch größere Probleme, die im Folgenden genauer analysiert werden.

Das **Verhalten** der Quellen soll durch das Integrationssystem vollkommen transparent für den Benutzer werden. Dabei muss das System entsprechende Maßnahmen ergreifen, um die Probleme aus bestimmten Verhaltenseigenschaften der Quellen zu

[52] In Abschnitt 6.2.2 wird beschrieben, wie der Zugriff über vorhandene Bibliotheken realisiert wird.

behandeln. Aufgrund der Nähe zum Abruf aus den Quellen wird diese Problematik noch im Rahmen dieses Unterkapitels untersucht.

Die Quellen können sehr unterschiedliche Möglichkeiten der **Informationsabfrage** anbieten. Dem Benutzer des Integrationssystems soll jedoch eine einheitliche Schnittstelle zur Abfrage der Daten bereitgestellt werden. Diese Schnittstelle verbirgt die Komplexität, die durch die Heterogenität der Quellen entsteht, und vereinfacht den Informationszugriff wesentlich. Der Abfrage der Daten wird aufgrund der Nähe zum Benutzer sowie ihrer großen Bedeutung ein eigenes Unterkapitel gewidmet.

4.2.3 Auswirkungen des Verhaltens der Quellen

Die in Unterkapitel 2.4 genannten Quellen können sehr unterschiedliche Ausprägungen bzgl. ihrer Verhaltenskriterien haben. Wie schon in Abschnitt 4.1.1 festgestellt wurde, sind je nach Verhalten der Quellen unterschiedliche Integrationsansätze vorteilhaft. Insbesondere erwies sich die Zwischenspeicherung der Daten je nach Situation als hilfreich oder hinderlich bzw. nicht realisierbar. Die Auswirkungen einer Zwischenspeicherung der Daten bei bestimmten Verhaltenskriterien verdeutlicht Tabelle 4-1.

So kann eine Zwischenspeicherung der Daten das Problem sehr hoher Antwortzeiten der Quellen lösen, indem die Abfragen aus einem lokalen Zwischenspeicher beantwortet werden. Allerdings ist ein Zwischenspeicher bzgl. der Aktualität der Daten benachteiligt, da er nur den Stand zum Zeitpunkt des letzten Abrufs widerspiegelt. Bei sehr dynamischen Informationen kann das ein Problem sein, da man in diesem Falle evtl. Daten liefert, die nicht mehr aktuell sind.

Tab. 4-1: Auswirkung der Nutzung eines Zwischenspeichers bei bestimmten Verhaltenskriterien der Quellen

	Antwortzeit	Verfügbarkeit	Dynamik	Umfang
gering	neutral	positiv	neutral	neutral
hoch	positiv	neutral	negativ	negativ

Jedoch sind Verhaltensweisen der Quellen praktisch beliebig kombinierbar und es können sich Kombinationen ergeben, in denen ein Zwischenspeicher sowohl positive als auch negative Auswirkungen haben kann. Tabelle 4-2 (S. 42) stellt die möglichen Kombinationen dar. In einigen Fällen kann man eine eindeutige Empfehlung für oder gegen den Einsatz eines Zwischenspeichers geben. In der Mehrzahl der Fälle sprechen jedoch einige Kriterien für und andere gegen eine Zwischenspeicherung. Ein einfaches Beispiel wäre eine Quelle, die weltweite Wetterprognosen liefert (hoher Umfang),

jedoch auch unter einer hohen Antwortzeiten oder einer schlechten Verfügbarkeit leidet.

Tab. 4-2: Mögliche Kombinationen von Verhaltenskriterien der Quellen

Antwortzeit		Verfügbarkeit		Dynamik		Umfang		Zwischenspeicher
gering	0	gering	+	gering	0	gering	0	Ja
gering	0	gering	+	gering	0	hoch	-	Trade-off
gering	0	gering	+	hoch	-	gering	0	Trade-off
gering	0	gering	+	hoch	-	hoch	-	Trade-off
gering	0	hoch	0	gering	0	gering	0	Neutral
gering	0	hoch	0	gering	0	hoch	-	Nein
gering	0	hoch	0	hoch	-	gering	0	Nein
gering	0	hoch	0	hoch	-	hoch	-	Nein
hoch	+	gering	+	gering	0	gering	0	Ja
hoch	+	gering	+	gering	0	hoch	-	Trade-off
hoch	+	gering	+	hoch	-	gering	0	Trade-off
hoch	+	gering	+	hoch	-	hoch	-	Trade-off
hoch	+	hoch	0	gering	0	gering	0	Ja
hoch	+	hoch	0	gering	0	hoch	-	Trade-off
hoch	+	hoch	0	hoch	-	gering	0	Trade-off
hoch	+	hoch	0	hoch	-	hoch	-	Trade-off

+: positiv -: negativ 0: neutral

In solchen problematischen Kombinationen ist ein **Trade-off** nötig. Die in Abschnitt 4.1.3 beschriebene hybride Integration ermöglicht solche Trade-offs.

So würde man für das oben beschriebene Beispiel der Wetterprognosen grundsätzlich eine virtuelle Integration anwenden, da der Umfang der Wetterprognosen für alle Orte der Welt zu hoch wäre, um diese Daten regelmäßig zu replizieren. Den hohen Antwortzeiten könnte man allerdings entgegenwirken, indem man z.B. die Ergebnisse jeder Abfrage für einen gewissen Zeitraum zwischenspeichert. So wäre die Menge der replizierten Daten vergleichsweise gering und man profitiert bei späteren Abfragen von der Geschwindigkeit des Zwischenspeichers.[53] Zusätzlich könnte man für die häufigsten Abfragen die Daten im Zwischenspeicher regelmäßig aktualisieren, um die Wartezeit bei einem Abruf aus der Quelle von der Benutzerabfrage zu entkoppeln. Diese Daten wären somit permanent aktuell und die Benutzerabfragen könnten in diesen Fällen stets aus dem Zwischenspeicher beantwortet werden.

[53] Natürlich geht dieser Vorteil verloren, wenn alle Abfragen die Wetterprognose für unterschiedliche Orte fordern. Allerdings ist durchaus damit zu rechnen, dass einige Orte sehr häufig abgefragt werden.

Der Trade-off besteht also bei diesem Beispiel darin, dass man nur einen kleinen Ausschnitt der Daten zwischenspeichert. So profitiert man in vielen Fällen von der Zwischenspeicherung und muss trotzdem nur eine begrenzte Menge an Daten replizieren. Diesen Vorteil des geringeren Umfangs der zwischengespeicherten Daten erkauft man sich allerdings damit, dass einige Abfragen nicht aus dem Zwischenspeicher beantwortet werden können. Den Vorteil der Geschwindigkeit erkauft man sich mit einer etwas geringeren Aktualität der Daten. Die Dauer der Zwischenspeicherung sollte also so gewählt werden, dass die Daten ausreichend aktuell sind.

Durch den geschickten Einsatz eines Zwischenspeichers können die Probleme aus dem Verhalten der Quellen also wesentlich reduziert oder gar umgangen werden. Für den Benutzer des Integrationssystems werden die Probleme im Idealfalle vollkommen transparent.

4.3 Abfragen an das Integrationssystem

Wie schon weiter oben festgestellt, erschwert die Vielzahl an zu berücksichtigenden Fällen den Zugriff auf die Quellen. Wünschenswert wäre eine einzige Schnittstelle, die einen einfachen Zugriff auf die Daten erlaubt und dabei die technischen Probleme der unterschiedlichen Zugriffsmöglichkeiten vollkommen transparent werden lässt.

Eine für ein Integrationssystem in einem IID geeignete Abfrageschnittstelle wird in diesem Unterkapitel erarbeitet. Dazu wird zunächst untersucht, welche Abfragen möglich und welche davon tatsächlich erforderlich sind. Nachdem eine Entscheidung über die Art der Abfrageschnittstelle getroffen wurde, wird die Schnittstelle dann genau definiert und anhand von Beispielen illustriert.

4.3.1 Mögliche Abfragen

Es gibt sehr vielfältige Möglichkeiten, Daten abzufragen. Diese Möglichkeiten lassen sich wie folgt klassifizieren: Selektion, Kombination, Aggregation, Gruppierung und Sortierung. Keine Abfragen im engeren Sinne stellen Modifizierungen des Datenbestandes dar, sie sind jedoch ein fester Bestandteil vieler Abfragesprachen. Dies ist eine pragmatische Klassifikation, die sich nicht strikt an eine bestimmte Terminologie (z.B. die der Relationalen Algebra) hält und zudem keinen Anspruch auf Vollständigkeit erhebt. Sie soll bloß zur Beurteilung der Abfragemöglichkeiten im Rahmen eines IIDs dienen.

Selektion

Mit der Selektion ist die Auswahl und Einschränkung der gelieferten Daten gemeint. Sie umfasst hier die die Selektion und Projektion der relationalen Algebra (RA). Selektionen können auf mehreren Ebenen vorgenommen werden.

Zunächst kann man die Datei (SQL: Tabelle, RA: Relation, XQuery: Dokument/-Kollektion) bestimmen, die die Daten enthält. In SQL geschieht das mit der `FROM`-Klausel, in XQuery in der `for`-Klausel.

Darauf aufbauend kann man die Sätze (Zeilen, Tupel, Elemente), die aus dieser Datei ausgewählt werden sollen, durch Bedingungen bestimmen. Diese Bedingungen beschreiben i.d.R., welche Werte bestimmte Felder (Spalten, Attribute, Attribute/-Elemente) haben müssen, um den jeweiligen Satz auszuwählen. In SQL und XQuery werden diese Bedingungen in der `WHERE`- bzw. `where`-Klausel beschrieben.

Schließlich kann man innerhalb der selektierten Sätze einzelne Felder (Spalten, Attribute, Elemente) auswählen. Das entspricht der Projektion der Relationalen Algebra. In SQL erfolgt diese Auswahl innerhalb der `SELECT`-Klausel, in XQuery in der `return`-Klausel.

Kombination

Die Kombination von Daten entspricht der Zusammenführung von Daten aus unterschiedlichen Dateien.

So kann man z.B. für relationale Daten mehrere Relationen (Tabellen) miteinander kombinieren, wobei die kombinierte Relation alle Attribute (Spalten) beider Basisrelationen in sich vereint. In der Relationalen Algebra spricht man von einem Produkt oder der Join-Operation. In SQL kann man eine solche Kombination über die `FROM`- und `JOIN`-Klauseln definieren.

Auch XML-Daten können aus unterschiedlichen Dateien zusammengeführt werden. In XQuery bestimmt man die Dateien in der `for`-Klausel.

Aggregation und Gruppierung

Auf die Daten können Aggregationsfunktionen angewendet werden. Üblich ist die Berechnung von Durchschnitts-, Maximal- und Minimalwerten. Es gibt aber eine sehr breite Palette an Aggregationsfunktionen, die weit darüber hinaus geht. Die aggregierten Werte können auch zur Selektion der Daten herangezogen werden.

Da Aggregationen, wie der Name schon sagt, Daten zusammenfassen, muss man bestimmen können, welche Daten zusammengefasst werden sollen. Ohne eine explizite Angabe werden alle Sätze einer Auswahl zu einem Satz aggregiert. Man kann die Sätze jedoch auch anhand der Werte bestimmter Felder gruppieren. So geschieht die Aggregation nur innerhalb der Sätze, für die diese Felder den gleichen Wert haben und es werden mehrere Sätze zurückgegeben.

Sortierung

Schließlich kann man das Ergebnis nach bestimmten Feldern auf- oder absteigend sortieren. Die Sortierung kann auch über mehrere Felder durchgeführt werden (z.B. zuerst nach Feld A, zusätzlich nach Feld B).

Modifizierung

Zusätzlich zu den oben beschriebenen Möglichkeiten zur Auswahl von Daten bieten die gängigen existierenden Abfragesprachen Operationen zur Modifikation des Datenbestandes an. Dabei handelt es sich im Wesentlichen um das Einfügen, Ändern und Löschen von Daten.

4.3.2 Erforderliche Abfragen

Nachdem die Möglichkeiten der Abfrage geklärt wurden, kann nun untersucht werden, welche Abfragemöglichkeiten in einem IID denn auch tatsächlich benötigt werden.

Selektionsmöglichkeiten werden sicherlich nötig sein. Man könnte die Auswahl z.B. auf eine bestimmte Quelle, auf eine bestimmte Art von Informationen (z.B. Artikel, Bilder) oder bzgl. des Veröffentlichungsdatums einschränken. Die Projektion der Daten auf einzelner Felder wird allerdings von untergeordneter Bedeutung sein. Die verarbeitenden Komponenten werden meistens die vollständigen Daten benötigen, anstatt einzelne Felder wie z.B. nur den Autor auszuwählen. Selbst wenn sie nur Ausschnitte benötigen, können sie diese problemlos aus dem gesamten Satz extrahieren.

Kombinationen mehrerer Sätze werden auf Datenebene nicht benötigt. Man wird den Titel eines Artikels A nicht mit dem Inhalt eines anderen Artikels B kombinieren. Sehr wohl wird man dem Abonnenten jedoch unterschiedliche Inhalte in einem Dokument präsentieren wollen. Dazu ist es sinnvoll, Informationen aus unterschiedlichen Dateien auswählen zu können. Die Zusammenstellung der Informationen ist jedoch Aufgabe anderer Komponenten (Matching, Layout) und erfordert keine Kombinationen der Daten.

Aggregationen werden vor allem auf numerische Daten angewendet. Die Inhalte in einem IID sind jedoch textueller Art. Aggregationen (und die damit verbundenen Gruppierungen) werden bei der Abfrage der Daten i.d.R. nicht auftreten. Sie könnten höchstens für Adminstrations- oder Statistikzwecke interessant sein, um z.B. herauszufinden, wie viele Artikel innerhalb der letzten Woche erschienen sind. Auf solchen Abfragen liegt jedoch nicht das Hauptaugenmerk beim Abruf der Daten aus dem Integrationssystem. Sie können daher in diesem Kontext vernachlässigt werden.

Eine **Sortierung** der Daten ist ebenfalls nicht nötig. Die Daten werden nach dem Abruf ohnehin durch die Matching-Komponente bzgl. ihrer individuellen Relevanz bewertet und anschließend durch die Layout-Komponente hinsichtlich Relevanz und optischen Kriterien angeordnet. Dadurch geht eine evtl. vorhandene Sortierung ohnehin verloren und ist von keiner Bedeutung.

Modifizierungen des Datenbestandes sind nicht vorgesehen. Der Grund findet sich hauptsächlich in der Natur der in einem IID zu verarbeitenden Informationen. Es handelt sich zumeist um redaktionelle Inhalte oder durch bestimmte Dienste gelieferte Informationen, für die die Quellen i.d.R. keine Modifikationsmöglichkeiten

vorsehen. Falls eine Quelle solche Möglichkeiten anbieten sollte, wird sie dafür eigene Schnittstellen anbieten, deren Einbindung jedoch außerhalb des Anwendungsbereichs des zu entwickelnden Integrationssystems liegt.

Zusammenfassend wird klar, dass lediglich Selektionen in Abfragen an das Integrationssystem berücksichtigt werden müssen.

4.3.3 Eignung unterschiedlicher Arten der Abfrage

In Abschnitt 2.3.3 wurden drei Arten von Abfrageschnittstellen dargestellt: Abfragesprachen, Funktionsaufrufe und einfacher Abruf. Im Folgenden sollen diese Arten der Datenabfrage mit ihren Ausprägungen bzgl. ihrer Eignung zum Abruf der Daten aus einem Integrationssystem in einem IID untersucht werden.

Abfragesprachen

Abfragesprachen bieten sehr umfangreiche Möglichkeiten, die gewünschten Daten auszuwählen. Sie stellen die mächtigste Möglichkeit zur Abfrage von Daten dar. Beispiele sind SQL und XQuery. SQL dient dem Zugriff auf relationale Datenbanken. Neben der Abfrage von Daten kann man mittels SQL auch die Datenschemata und Einschränkungen definieren, sowie Datensätze erstellen, aktualisieren und löschen. XQuery[54] ist eine noch relativ junge Abfragesprache, die gezielte Abfragen auf XML-Daten ermöglicht.

Abfragesprachen bieten weitreichende Möglichkeiten zur Selektion von Daten und sind somit generell für Abfragen an das Integrationssystem geeignet. Sie erlauben auch komplexere Abfragen, die über einfache Selektionen hinaus gehen.

Würde man eine Abfragesprache zum Zugriff auf die Daten nutzen, ergäben sich drei Möglichkeiten: Die Nutzung einer vorhandenen Abfragesprache, die Definition einer eigenen Abfragesprache und eine Kombination aus beidem.

Die **Nutzung einer vorhandenen Abfragesprache** liegt zunächst nahe. SQL ist standardisiert, hat einen hohe Reifegrad und ist den meisten Entwicklern bekannt. XQuery ist ebenfalls eine standardisierte Sprache und gewinnt in letzter Zeit deutlich an Popularität.

Allerdings muss man, falls man diese Sprachen nicht selbst implementieren möchte, seine Daten in einer[55] Datenbank verwalten, die die gewünschte Abfragesprache

[54] Für die Spezifikation von XQuery 1.0 siehe World Wide Web Consortium (W3C) (2007e).

[55] Tatsächlich müssten die Daten in genau einer Datenbank liegen, da die üblichen Abfragesprachen keine datenbankübergreifenden Abfragen vorsehen. Um diesen Anforderungen gerecht zu werden, wurden Erweiterungen für diese Sprachen entwickelt. Ein Beispiel ist SchemaSQL (siehe Lakshmanan u. a. (2001)), das den Zugriff auf sowie die Manipulation von Daten in mehreren Datenbanken erlaubt.

unterstützt. Jedoch ist es nicht möglich, die Daten aller Quellen in Datenbanken zu halten. Hochdynamische oder sehr umfangreiche Informationsquellen wie z.B. Börsen- oder Wetterinformationen können nicht vollständig in internen Datenbanken repliziert werden. Diese Quellen müssen direkt abgefragt werden, wobei man nicht davon ausgehen kann, dass dies über die gewünschte Abfragesprache geschieht. Also müssen die Abfragen an das Integrationssystem in Abfragen an diese Quellen übersetzt werden. Dies ist vor allem aufgrund der Flexibilität und Komplexität der gängigen Abfragesprachen mit einem sehr hohen Aufwand verbunden. Zwar gibt es kommerzielle Integrationssysteme[56], die beim Zugriff auf solche Quellen behilflich sind, dennoch erscheint dieser Aufwand unangemessen, wenn man, wie im Falle eines IIDs, tatsächlich nur eine kleine Untermenge der Funktionalität – nämlich die Selektion – der Abfragesprachen benötigt.

Eine Alternative ist die **Definition einer eigenen Abfragesprache**. Diese Sprache kann recht simpel gehalten werden, da lediglich Selektionen möglich sein sollen. Somit ist die Verarbeitung dieser Abfragesprache relativ einfach möglich. Die Abfragen können mit moderatem Aufwand in Abfragen an die Quellen übersetzt werden. Dennoch muss dieser Aufwand berücksichtigt werden. Außerdem muss der Entwickler, der Abfragen an das Integrationssystem stellen möchte, eine neue Sprache lernen, was aufgrund der Einfachheit dieser Sprache jedoch nicht besonders ins Gewicht fallen sollte.

Zusätzlich ist eine **Kombination dieser Möglichkeiten** denkbar. Für Quellen, deren Daten in einer Datenbank gehalten werden, kann man eine vorhandene Abfragesprache nutzen. Für alle anderen Quellen kann man eine einfache Abfragesprache selbst definieren. Vorteilhaft an dieser Variante ist, dass sich die Übersetzung der Abfragen für solche Daten erübrigt, die in Datenbanken liegen. Dort kann man außerdem den vollen Funktionsumfang der vorhandenen Abfragesprache nutzen. Ein sehr wichtiger Nachteil ist allerdings, dass man somit schon an der Abfrageschnittstelle eine Heterogenität schafft, indem man unterschiedliche Abfragesprachen nutzt und nach den Quellen differenzieren muss. Das widerspricht jedoch der Zielsetzung eines Integrationssystems: Transparenz und Minimierung der Heterogenität.

Funktionsaufrufe

Funktionsaufrufe bieten im Vergleich zu Abfragesprachen wesentlich beschränktere Möglichkeiten zur Auswahl von Informationen. Die Abruffunktionen haben i.d.R. eine bestimmte Menge an Parametern, für die Werte bestimmt werden können, die die Auswahl einschränken.

---

[56] Beispiele für kommerzielle Integrationssysteme sind IBM Information Server oder BEA Liquid Data for WebLogic.

Sie bieten somit wesentlich weniger Flexibilität als Abfragesprachen und sind oft auf ganz spezifische Abfragen beschränkt. Sie sind in ihrer Grundform einfach zu definieren und durch ihre geringen Freiheitsgrade leicht zu implementieren. Insbesondere für sehr einfache Abfragen sind sie gut geeignet.

Allerdings müssen für neue Abfragen neue Funktionen bereitgestellt werden, was in einem IID häufig der Fall sein kann, da die Menge der anzubindenen Quellen nicht fest vorgegeben ist. Es können jederzeit neue Quellen hinzukommen. Das führt entweder zu einer großen Anzahl an Funktionen oder zu Parametern, die selbst komplexe Abfragen entgegennehmen.

Funktionsaufrufe werden der nötigen Flexibilität durch die Heterogenität der in einem IID anzubindenden Quellen also nicht gerecht. Vor allem ist das Ziel des Integrationssystems eine einheitliche, einfache Schnittstelle zu den Daten zu bieten. Die Schaffung einer Vielzahl an Funktionsaufrufen widerspricht diesem Ziel.

Einfacher Abruf

Der einfache Abruf von Daten erlaubt keine gezielten Abfragen. Im Kontext eines IIDs würde das bedeuten, dass man mit jedem Abruf den gesamten Datenbestand (bis zu einem bestimmten Datum) geliefert bekommt.

Da jedoch Selektionen möglich sein sollen, disqualifiziert sich der einfache Abruf schon über die Nichterfüllung dieses einfachen Kriteriums.

Schlussfolgerung

Funktionsaufrufe sowie der einfache Abruf kommen für den Zugriff auf die Daten nicht in Frage. Ebenso erscheint die Nutzung unterschiedlicher Abfragesprachen nicht zweckmäßig. Übrig bleiben die Nutzung einer vorhandenen Abfragesprache sowie die Definition einer eigenen Abfragesprache. Tabelle 4-3 (S. 50) gibt eine Übersicht über die Vor- und Nachteile dieser Alternativen.

Generell sind beide Alternativen geeignet, weitere Vorteile sind daher eher ein Bonus und keine Notwendigkeit. Daher müssen insbesondere die Nachteile der beiden Möglichkeiten abgewogen werden.

Es stellt sich die Frage, ob der Aufwand für die Übersetzung der Abfragen einer Standardsprache in die Abfragen an die Quellen größer oder kleiner ist als der Aufwand für die Definition und Implementierung einer eigenen Sprache.

Die Definition einer einfachen, anwendungsspezifischen Sprache – im Englischen *Domain Specific Language* (DSL) genannt – die ausschließlich Selektionen erlaubt, ist recht einfach. Der Aufwand der Übersetzung einer einfachen Sprache ist wesentlich

Tab. 4-3: Vor- und Nachteile der Alternativen zur Datenabfrage

	Vorteile	Nachteile
Vorhandene Sprache	• Standardisiert • Erprobt • Den meisten Entwicklern bekannt • Sehr ausdrucksstark	• Nicht alle Daten können in Datenbanken repliziert werden • Hoher Aufwand für Übersetzung für einige Quellen
Eigene Sprache	• Erfüllt genau die Anforderungen • Relativ leichte Übersetzung in Abfragen an die Quellen • Einfach zu verstehen	• Definition nötig • Aufwand für Übersetzung für alle Quellen

geringer als für die Übersetzung einer umfangreichen Standardsprache. Allein die Syntax üblicher Standardsprachen ist oft sehr umfangreich, die dahinter steckende Logik erhöht die Komplexität weiter.

Unter Betrachtung dieser Aspekte ist die Definition einer einfachen Abfragesprache am sinnvollsten. Der Umfang dieser Sprache wird im folgenden Abschnitt beschrieben und anhand einiger Beispiele illustriert.

4.3.4 Abfragen an das Integrationssystem

Wie schon in Abschnitt 4.3.2 festgestellt wurde, sind bei der Auswahl der Daten lediglich Selektionen von Bedeutung. Die Selektionen können sich auf ganze Quellen, auf die Art der Informationen oder auf einzelne Eigenschaften (Datum, Autor, etc.) einer Informationseinheit beziehen.

Sprachelemente

Die Selektionen lassen sich über einfache **Vergleiche** der Art `Eigenschaft = "Wert"` formulieren. Ein solcher Vergleich besteht praktisch immer aus zwei Operanden und einem Vergleichsoperator. Die **Operanden** können Eigenschaften (bzw. Felder oder Attribute) der Informationseinheiten oder Werte (Zeichenketten und Zahlen) sein. Mögliche **Operatoren** sind `=`, `<`, `<=`, `>`, `>=`, `!=` und `contains`. Arithmetische Ausdrücke (Addition, Multiplikation, etc.) werden bei der Auswahl der Daten von untergeordneter Bedeutung sein und sind daher nicht Bestandteil der Abfragesprache.

Ebenso sind **und**- sowie **oder**-Verknüpfungen (bzw. `and` und `or`) mehrerer Vergleiche möglich. Die und-Verknüpfung bindet stärker als die oder-Verknüpfung. Um

die Rangordnung der Verknüpfungen zu beeinflussen, werden **Klammerungen** um einzelne Teilausdrücke genutzt. Im Term `a=1 and b=2 or b=3` werden zuerst die ersten beiden Vergleiche verknüpft, im Term `a=1 and (b=2 or b=3)` werden die letzten beiden Vergleiche zuerst verknüpft. Ebenfalls existiert ein **Negationsoperator** (`not`).

Möglicherweise unterstützen einige Quellen bestimmte Operatoren bei der Abfrage nicht. Bei der Übersetzung muss dieses Verhalten dann entweder durch den Übersetzer emuliert werden oder es wird eine Fehlermeldung ausgegeben. Für nicht existierende **Eigenschaften** bzw. Variablen kann man entweder einen unbekannten Wert (in SQL wäre das der Wert `NULL`) annehmen oder einen Fehler und eine leere Menge an Resultaten ausgeben.

Sprachdefinition

Zur Definition von formalen Sprachen eignet sich die **Erweiterte Backus-Naur-Form**[57] (EBNF), in der die Abfragesprache im Folgenden definiert wird:

```
Abfrage      = [ Oder ] ;
Oder         = Und { "or" Und } ;
Und          = Bedingung { "and" Und } ;
Bedingung    = Nicht { Operator Nicht } ;
Nicht        = Gruppe | "not" Nicht ;
Gruppe       = Operand | "(" Oder ")" ;
Operand      = Literal | Bezeichner ;
Operator     = "=" | "!=" | ">" | ">=" | "<" | "<=" | "contains" ;
Literal      = Boolean | Zahl | Zeichenfolge ;
Bezeichner   = BUCHSTABE { BUCHSTABE | ZIFFER | "_" | '-' | '.' } ;
Boolean      = "true" | "false" ;
Zahl         = [ "-" ] ZIFFER { ZIFFER } ;
Zeichenfolge = '"' { ZEICHEN } '"' ;
ZEICHEN      = ? alle Zeichen außer dem Anführungszeichen ? ;
ZIFFER       =  "0" | ... | "9" ;
BUCHSTABE    = "a" | ... | "z" | "A" | ... | "Z" ;
```

Die Grammatik gibt folgende Operatoren-Präzedenz vor (in absteigender Präzedenz):

1. Explizit geklammerte Gruppe
2. Negation (`not`)
3. Bedingung (z.B. `a = b`)

---

[57] Die Erweiterte Backus-Naur-Form wurde von der ISO/IEC standardisiert. Für diesen Standard siehe ISO/IEC (1996).

4. Und-Verknüpfung (`and`)
5. Oder-Verknüpfung (`or`)

Datumsangaben werden nicht separat betrachtet. Sie werden wie in SQL einfach als Zeichenfolgenliteral im ISO-8601-Format[58] `"YYYY-MM-DDThh:mm:ssTZD"` ausgezeichnet. Außerdem werden nur ganze Zahlen berücksichtigt – reelle Zahlen sind vorerst nicht vorgesehen.

Zum Vergleich: Eine BNF-Definition von SQL-2003[59] ist mit über 200KB reinem Text (inklusive Kommentaren) bzgl. der Zeichenanzahl mehr als 330 mal so umfangreich wie die hier vorgestellte Grammatik.

Beispiele

Um die Ausführungen zur Abfragesprache abzuschließen, wird sie im Folgenden anhand einiger typischer Beispiele demonstriert.

- Selektion aller Nachrichtenartikel und Newsfeed-Einträge vom 6. Juli 2007:
  `(type="news-article" or type="feed-entry") and date contains "2007-07-06"`
- Selektion aller Nachrichtenartikel, die „Köln“ als Ortsangabe enthalten:
  `type="news-article" and location="Köln"`
- Selektion der aktuellen Wetterinformationen für einen bestimmten Ort:
  `type="weather" and country="de" and city="cologne"`
- Selektion des aktuellen Börsenkurses einer Aktie:
  `type="stock-quote" and exchange="nasdaq" and symbol="GOOG"`
- Selektion einer eindeutig identifizierbaren Informationseinheit:
  `id="some.source.234213374711"`

Die genannten Eigenschaften und Werte, nach denen ausgewählt wird, sind nur beispielhaft zu verstehen. Sie müssen in einer konkreten Implementierung nicht in dieser Form bzw. mit diesem Namen existieren. Die genaue Ausprägung der Eigenschaften hängt von den Quellen und von der Implementierung der jeweiligen Übersetzung der Abfragen ab.

---

[58] Das ISO-8601-Format wurde gewählt, da es (mit leichten Modifikationen) dem XML-Schema-Datentypen `xsd:dateTime` entspricht. Es wird entsprechend häufig in XML-Dokumenten genutzt, die wiederum den Großteil der Quelldaten darstellen (siehe Unterkapitel 9.1). Für eine Beschreibung von `xsd:dateTime` siehe World Wide Web Consortium (W3C) (2004).

[59] Siehe Leffler (2007).

4.4 Transformationen zwischen den Austauschformaten

Die vom Integrationssystem gelieferten Informationen sollen durch die restlichen Komponenten eines IIDs möglichst leicht zu verarbeiten sein. Jedoch wird die Verarbeitung durch die Heterogenität der Informationen deutlich erschwert.

Das Ziel der Transformationen ist dementsprechend die in Anforderung 4 (S. 26) geforderte Minimierung dieses Verarbeitungsaufwandes. Im Folgenden wird erörtert, welche Probleme sich bei der Erreichung dieses Ziels ergeben und wie sie gelöst werden können.

4.4.1 Aufwand durch hohe Anzahl an Transformationen

Ohne eine Integration der Formate ergibt sich bei der Verarbeitung der Daten, die in einer Vielzahl von Austauschformaten vorliegen können, eine hohe Anzahl nötiger Transformationen: Die Informationen liegen in e Eingabeformaten vor. Sie müssen von mehreren Komponenten verarbeitet werden. Jede Komponente muss also jedes Eingabeformat verarbeiten können. Ebenso muss eine Layout-Komponente die Daten aus jedem Eingabeformat in jedes Ausgabeformat überführen. Zählt man die Komponenten zu den a Ausgabeformaten[60], so ergeben sich $e \cdot a$ Transformationen. Abbildung 4-9 (S. 54) veranschaulicht diesen Sachverhalt.

Bei 10 Eingabeformaten ($e = 10$), 5 verarbeitenden Komponenten und 3 Ausgabeformaten ($a = 5 + 3 = 8$) ergeben sich so schon $10 \cdot 8 = 80$ nötige Transformationen.

Im Vergleich mit den unterschiedlichen Möglichkeiten des Zugriffs auf die Daten aus den Quellen (siehe Abschnitt 4.2.1) ergeben sich durch die unterschiedlichen Austauschformate wesentlich mehr zu berücksichtigende Fälle, da sich die Anzahl hier multiplikativ und nicht additiv wie bei den Unterscheidungskriterien der Quellen ergibt.

4.4.2 Reduktion der Anzahl durch Integration der Formate

Um die Komplexität der Verarbeitung der Daten zu reduzieren, muss man die Anzahl der durch die Komponenten zu verarbeitenden Formate durch Generalisierungen auf möglichst wenige Formate verringern.

Die Eingabeformate werden zunächst in ein **Zwischenformat** überführt, das dann von den Komponenten verarbeitet wird. Der theoretische Idealfall wäre eine Reduktion auf ein einziges Zwischenformat. Die Komponenten müssten dann anstatt e Formaten nur noch ein einziges Format – das Zwischenformat – verarbeiten können, wie Abbildung 4-10 (S. 54) es veranschaulicht.

[60] Die Eingabedaten müssen für jede Komponente in eine interne Datenstrukturen überführt werden, die durch die in dieser Komponente eingesetzten Algorithmen verarbeitet werden kann.

Abb. 4-9: Transformationen zwischen Eingabeformaten, Komponenten und Ausgabeformaten

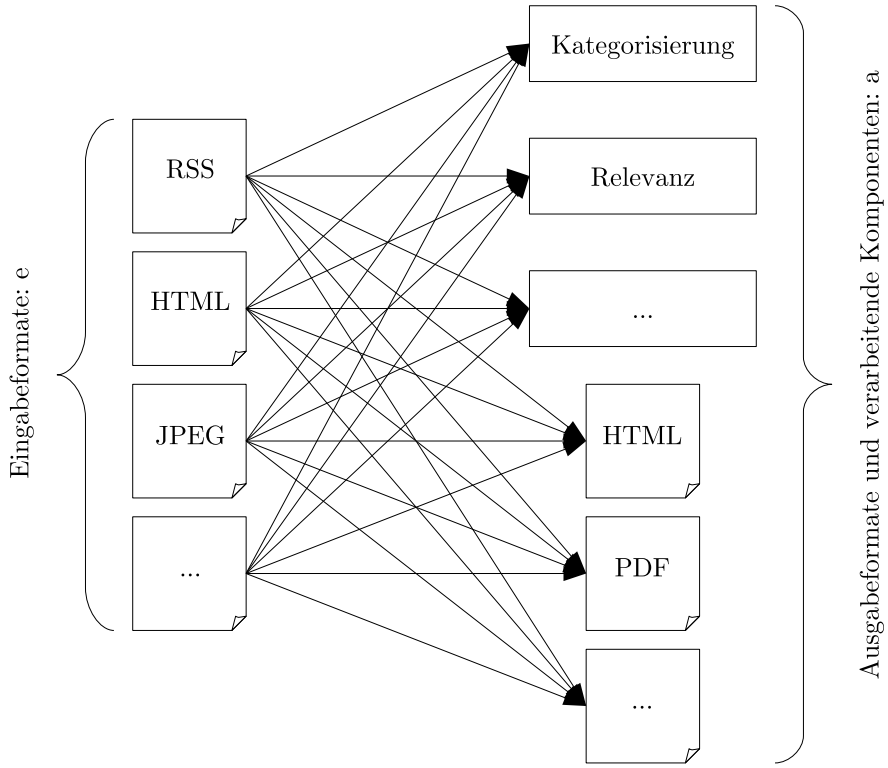
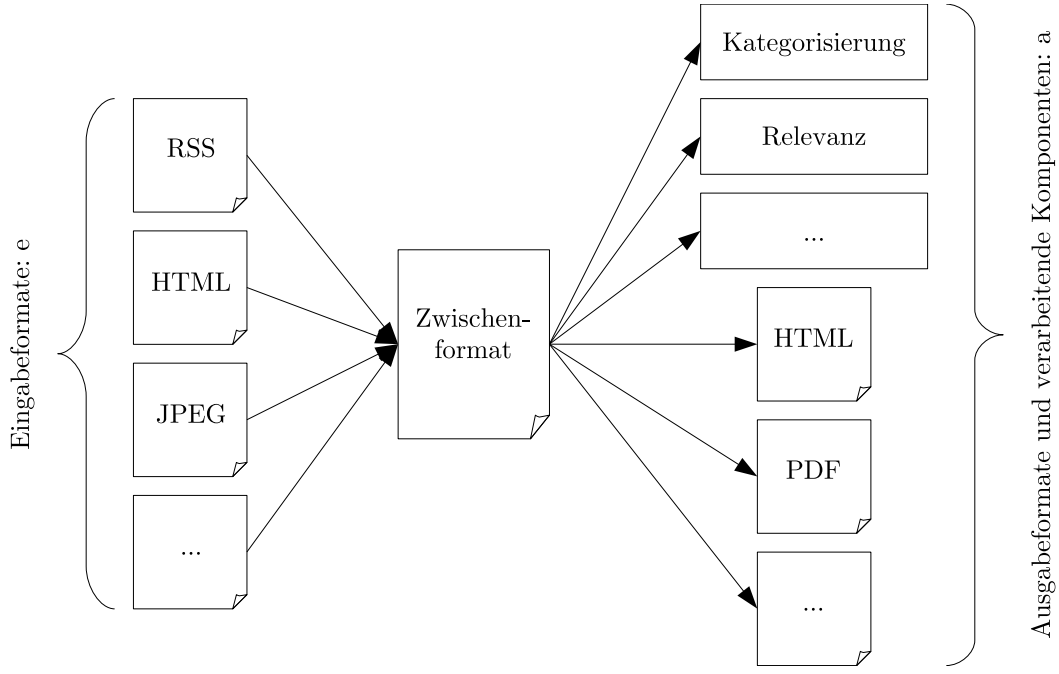


Abb. 4-10: Komplexitätsreduktion durch ein Zwischenformat



Dazu müssen die e Eingabeformate zunächst in das Zwischenformat überführt werden, was zu $e \cdot 1 = e$ Transformationen führt. Da die a Komponenten und Ausgabeformate nur noch ein Format verarbeiten müssen, ergeben sich $1 \cdot a = a$ Transformationen für die Komponenten und in die Ausgabeformate. Die Anzahl an Transformationen würde sich so von den in Abschnitt 4.4.1 identifizierten $e \cdot a$ Transformationen auf $e + a$ reduzieren. Für das oben angeführte Beispiel ($e = 10$, $a = 8$) ergeben sich anstatt den 80 nötigen Transformationen nur noch $10 + 8 = 18$ Transformationen.

4.4.3 Aufwand durch Komplexität der Zwischenformate

Jedoch wird es nicht möglich oder sinnvoll sein, alle Daten in ein einziges Format zu überführen. Dieses Format müsste eine Obermenge aller in Frage kommender Eingabeformate sein. Will man keine mit semantischen Verlusten behaftete Vereinfachungen machen, so hätte das Zwischenformat einen sehr großen Umfang.

Das widerspricht jedoch einem wichtigen Prinzip der Informatik: Teile und Herrsche. Diesem Prinzip entsprechend wird die Verarbeitung eines sehr umfangreichen Formats i.d.R. aufwändiger sein als die Verarbeitung mehrerer kleinerer Formate. Die Summe der Komplexität[61] der Verarbeitung mehrerer Einzelformate ist dann also größer als die Komplexität eines gemeinsamen Formats. Da gerade ein möglichst geringer Aufwand bei der Weiterverarbeitung der Daten das Ziel der Transformationen ist, muss die **Komplexität der Formate** unbedingt beachtet werden.

4.4.4 Nutzung mehrerer Zwischenformate

Es ist also sinnvoller, **mehrere Zwischenformate** zu definieren, die jeweils eine Generalisierung mehrerer ähnlicher Formate darstellen. Abbildung 4-11 (S. 56) illustriert diese Möglichkeit. So können die einzelnen Zwischenformate relativ einfach gehalten werden. Obwohl die Komponenten nun mit mehreren anstatt nur einem Zwischenformat arbeiten müssen, kann der Gesamtaufwand durch eine verringerte Komplexität der Formate reduziert werden.

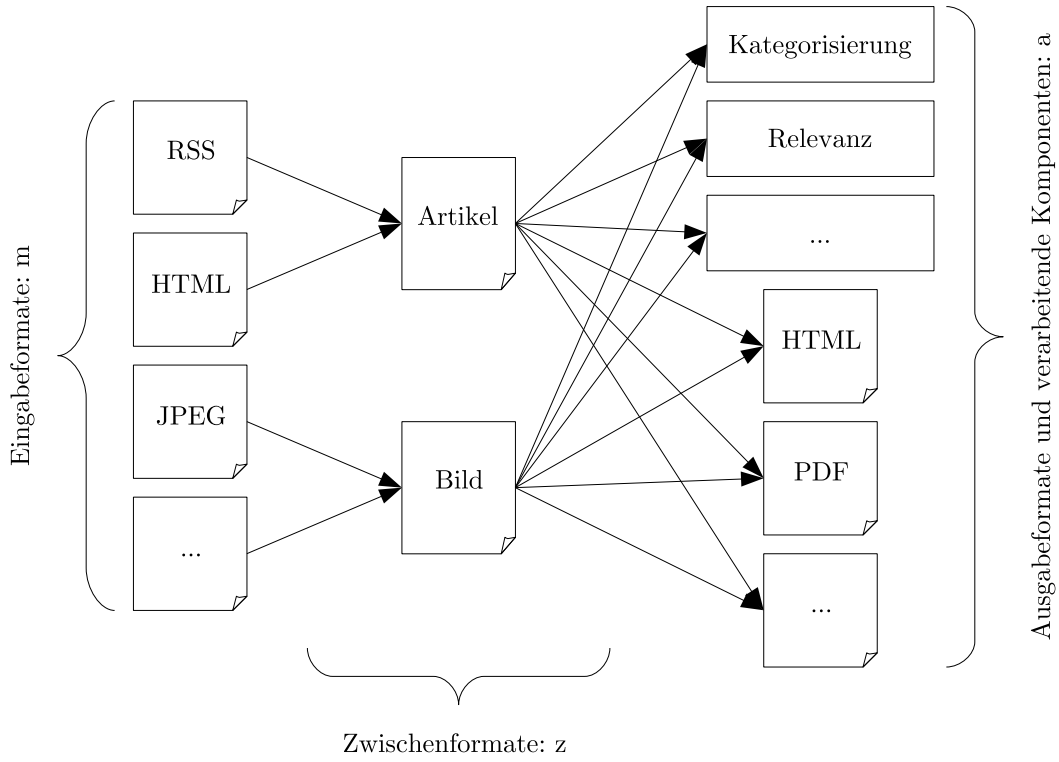
Da die Zusammenfassung disjunkter Formate sehr komplexe Zwischenformate ergibt, liegt es nahe, ähnliche Formate zusammenzufassen. Hat man mehrere **ähnliche Formate** (z.B. unterschiedliche Formate, die jeweils einen Nachrichtenartikel darstellen), so kann man sie in einem Zwischenformat aggregieren, das nur unwesentlich größer[62] ist als das umfangreichste der betrachteten Eingabeformate. Gleiche

---

[61] Mit Komplexität ist in diesem Kontext nicht die aus der Theoretischen Informatik bekannte Komplexitätsklasse gemeint. Ebensowenig handelt es sich um die Ausführungszeit der Implementierung. Gemeint ist die Komplexität, die ein Entwickler bei der Implementierung der Verarbeitung bewältigen muss. Mit anderen Worten: Der Entwicklungsaufwand.

[62] Im Idealfall kann es auch gleich groß sein.

Abb. 4-11: Komplexitätsreduktion durch mehrere Zwischenformate



Strukturelemente, die in jedem Eingabeformat separat enthalten sind, existieren im Zwischenformat nur noch ein mal.

Nimmt man semantische Verluste in Kauf, fasst man also mehrere Strukturelemente zu einem zusammen oder verwirft sie, so kann das Zwischenformat sogar kleiner als die Summe der Eingabeformate sein.

Der entscheidende Faktor bei der Entscheidung zur Zusammenfassung mehrerer Formate ist die Ähnlichkeit zwischen den Formaten. Redundanzen zwischen unterschiedlichen Formaten werden so vermieden. Ebenfalls entstehen keine übermäßig komplexen Formate, die die weitere Verarbeitung erschweren würden.

5 Entwurf

5.1 Vorgehensweise

Der im Folgenden beschriebene Entwurf soll den Aufbau des Systems verdeutlichen und die getroffenen Architekturentscheidungen beschreiben. Es ist nicht das Ziel, das System bis ins kleinste Detail mit allen Klassen, Methoden und Attributen wiederzugeben. Der dargestellte Entwurf ist keine vollständige Dokumentation des Systemaufbaus, vielmehr soll er die wesentlichen Zusammenhänge verdeutlichen. Oft werden nur solche Teile der Klassen gezeigt, die für die gerade im Text diskutierten Sachverhalte relevant sind. Für eine vollständige Beschreibung aller Klassen und Schnittstellen sei auf die Quelltext-Dokumentation verwiesen.[63]

Trotz der teilweise recht abstrakten Darstellung handelt es sich jedoch auch nicht um einen reinen Grobentwurf. An bestimmten Stellen werden selektiv ganz konkrete Aspekte detailliert dargestellt, um das Bild an diesen Stellen zu schärfen.

Bevor der eigentliche Entwurf des Systems beschrieben wird, sollen im folgenden Unterkapitel zunächst einige grundsätzliche Architekturentscheidungen beschrieben werden. Das System wird dann im Wesentlichen *Top-Down* – von der Abfrageschnittstelle über die Quell-Komponenten bis hin zu den Informationseinheiten – beschrieben.

[63] Für weitere Hinweise zur Dokumentation des Systems siehe Unterkapitel 6.4.

5.2 Grundlegende Architekturentscheidungen

Lose Kopplung

Mit Kopplung ist hier die Kopplung zwischen den Klassen innerhalb des Systems gemeint, also die Kopplung im Sinne der OOP, und nicht die systemübergreifende Kopplung zwischen unterschiedlichen Diensten. Die **Kopplung** entspricht also in diesem Zusammenhang dem Grad der Abhängigkeiten zwischen zwei Klassen.

Generell ist eine möglichst lose Kopplung erstrebenswert.[64] Es sollen also möglichst wenige Abhängigkeiten zwischen den Klassen existieren. Viele Abhängigkeiten führen dazu, dass bei der Änderung einer Klasse im schlimmsten Falle alle von dieser Klasse abhängigen Klassen modifiziert werden müssen, damit sie mit dem neuen Verhalten zurecht kommen.

Nicht vermeidbare Abhängigkeiten zwischen Klassen sollen über möglichst schmale und stabile Schnittstellen realisiert sein, um Problemen durch Veränderungen an bestehenden Klassen zu minimieren. Eine Klasse, die mit einer anderen Klasse kommuniziert, soll möglichst keine Annahmen über die Implementierung dieser Klasse machen müssen. Idealerweise ist die Kenntnis der genutzten Schnittstelle ausreichend – das Verhalten der Klassen sollte vollständig über die Schnittstelle gekapselt werden.

Eine lose Kopplung trägt der Flexibilität, der Erweiterbarkeit und der Einfachheit zu.

Offene Architektur

Unter einer **offenen Architektur** wird hier verstanden, dass sie möglichst wenige Einschränkungen bezüglich möglicher Erweiterungen macht. Das beinhaltet, dass die Architektur mit möglichst wenig Vorgaben für zukünftige Erweiterungen auskommt.

Insbesondere sollen im zu entwickelnden System nur sehr wenige Annahmen über die potenziell anzubindende Quellen gemacht werden. Das ist nötig, da sie äußerst heterogen sein können. Sie haben möglicherweise so gut wie gar keine Gemeinsamkeiten, die für alle Quellen angenommen werden können. Zu eng definierte Annahmen können zu einem Ausschluss bestimmter Quellen führen.

Eine offene Architektur erhöht die Flexibilität des Systems. Sie ermöglicht es insbesondere, auf die Heterogenität der Quellen einzugehen. Ebenso kann sie aufgrund möglichst weniger Annahmen und schmaler Schnittstellen leicht und einfach erweitert werden.

---

[64] Eine mögliche Definition der Losen Kopplung findet man in McConnell (2004), S. 100 ff. Im gleichen Kapitel dieses Buches findet man ebenfalls weitere, erstrebenswerte Entwurfsziele.

Autonome Quell-Komponenten

Die oben beschriebene offene Architektur führt dazu, dass die Komponenten für die Anbindung der Quellen recht autonom agieren. Die einzige verpflichtende Vorgabe ist die Schnittstelle, über die sie in das System eingebunden werden.

Konkret bedeutet das für den Fall der Quellen, dass alle Annahmen über die Quellen in einer schmalen Schnittstelle festgehalten werden, die definiert, welche Abfragen eine Quelle verstehen muss und welche Ergebnisse von ihr erwartet werden. Wie eine Quelle diese Vorgaben realisiert, ist ihr überlassen. Es werden sinnvoll erscheinende Vorgehensweisen vorgeschlagen und auch implementiert, sie sind jedoch nicht verbindlich und je nach Quelle kann vollständig davon abgewichen werden.

Solche autonome Quell-Komponenten werden in der Literatur auch als **Fat Wrapper**[65] bezeichnet. Sie bewerkstelligen die komplette quellenspezifische Funktionalität inkl. Übersetzung der Abfrage und der Ergebnisse.

Sie haben den Vorteil, dass die Schnittstelle sehr dünn ist und dass das Integrationssystem nur minimale Annahmen über die Quellen machen muss. Außerdem können sie effizient implementiert werden, da der Wrapper sehr genau auf die Besonderheiten einer Quelle eingehen kann.

Der Nachteil ist allerdings, dass für jede neue Quelle recht viel Funktionalität implementiert werden muss, da das Integrationssystem vergleichsweise wenig Arbeit übernimmt. Häufig auftretende Aufgaben können jedoch durch die im nächsten Unterabschnitt beschriebene Hilfsklassen bewerkstelligt werden, was die Implementierung eines Fat Wrapper stark erleichtert. Im Idealfall kann ein Großteil der Aufgaben durch eine Kombination dieser Klassen gelöst werden.

Hilfsklassen

Um die doppelte Implementierung gleicher Funktionalität zu vermeiden, soll diese Funktionalität in Hilfsklassen ausgelagert werden. Es ist durchaus zu erwarten, dass gewisse Funktionalitäten für mehrere unterschiedliche Quellen identisch sind.

Da diese Funktionalität jedoch nicht im Kern der Integrationssystems implementiert werden soll, bietet es sich an, sie in Hilfsklassen auszulagern. Somit kann man redundanten Quelltext weitgehend vermeiden, obwohl der Kern des Integrationssystems recht einfach gehalten wird.

---

[65] Zur Beschreibung eines *Fat Wrappers* siehe Domenig u. Dittrich (1999), S. 69–70.

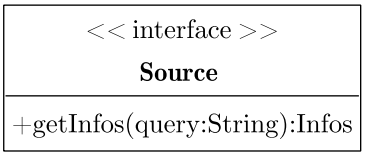
5.3 Abfrageschnittstelle

Die Abfragen an das Integrationssystem werden, wie in Abschnitt 4.3.4 herausgearbeitet wurde, in Form einer einfachen, auf die Zwecke dieses Systems abgestimmten Abfragesprache formuliert. Sie sind aus technischer Sicht also nichts weiter als eine Zeichenfolge, die Befehle zur Auswahl der Daten enthält.

Als Antwort erwartet der Benutzer eine Menge von Informationseinheiten, die seine Abfragebedingungen erfüllen. Welche Quellen die Informationseinheiten liefern, soll für ihn vollkommen transparent[66] sein – er stellt seine Abfrage an eine einzige, übergeordnete Quelle.

Die in Abbildung 5-12 dargestellte Schnittstelle beschreibt genau diesen Sachverhalt und verdeutlicht die Einfachheit dieser Schnittstelle. Die Abfrage wird als `String` übermittelt. Als Antwort wird eine Menge von Informationen geliefert.

Abb. 5-12: UML-Klassendiagramm für die Abfrageschnittstelle



Die Schnittstelle bietet ganz bewusst nur einen lesenden Zugriff auf die Daten. Ein schreibender Zugriff wurde in Abschnitt 4.3.2 ausgeschlossen.

Wie diese Schnittstelle von den Quellen umgesetzt wird, beschreiben die folgenden Unterkapitel. Die technische Realisierung des Zugriffs auf diese Schnittstelle in Form einer Client/Server-Architektur wird in Abschnitt 6.2.4 beschrieben.

[66] Tatsächlich könnte er die Auswahl mit entsprechenden Bedingungen auf bestimmte Quellen einschränken. Ebenso könnte er aus den Metadaten der gelieferten Resultate ableiten, aus welchen Quellen die Daten stammen.

5.4 Quell-Komponenten

5.4.1 Pool

Wie im vorigen Unterkapitel beschrieben, verhält sich das Integrationssystem dem Benutzer gegenüber wie eine einzige Quelle. Tatsächlich werden aber viele unterschiedliche Quellen angebunden. Sie werden von einem **Pool** aggregiert, der die Abfrage entgegennimmt und an die einzelnen Quell-Komponenten weiterleitet. Die Ergebnisse aller Quell-Komponenten werden zusammengefasst und an den Benutzer weitergeleitet.

Abb. 5-13: UML-Komponentendiagramm für die Organisation der Quellen

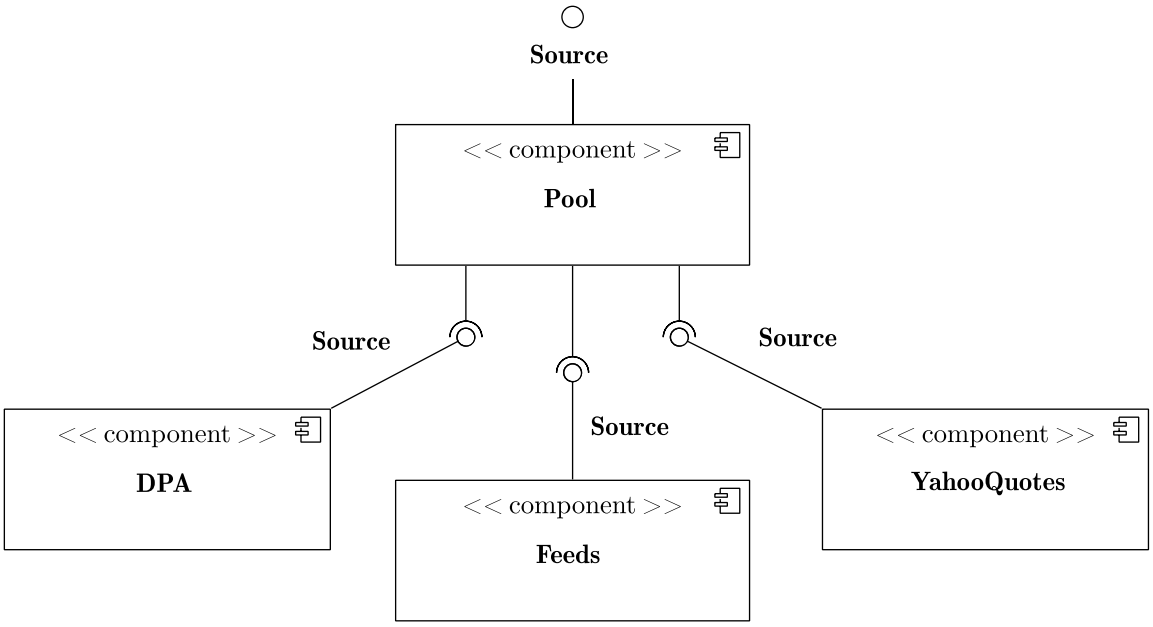


Abbildung 5-13 verdeutlicht die Beziehung zwischen dem Pool und den einzelnen Quell-Komponenten. Der Pool stellt nach außen die im vorigen Unterkapitel dargestellte `Source`-Schnittstelle bereit. Er ist selber eine Quelle, die jedoch nicht auf externe Daten zugreift, sondern die Daten aus den übrigen Quell-Komponenten aggregiert. Dazu greift er ebenfalls über die `Source`-Schnittstelle auf die Quell-Komponenten zu, die im folgenden Abschnitt genauer betrachtet werden.

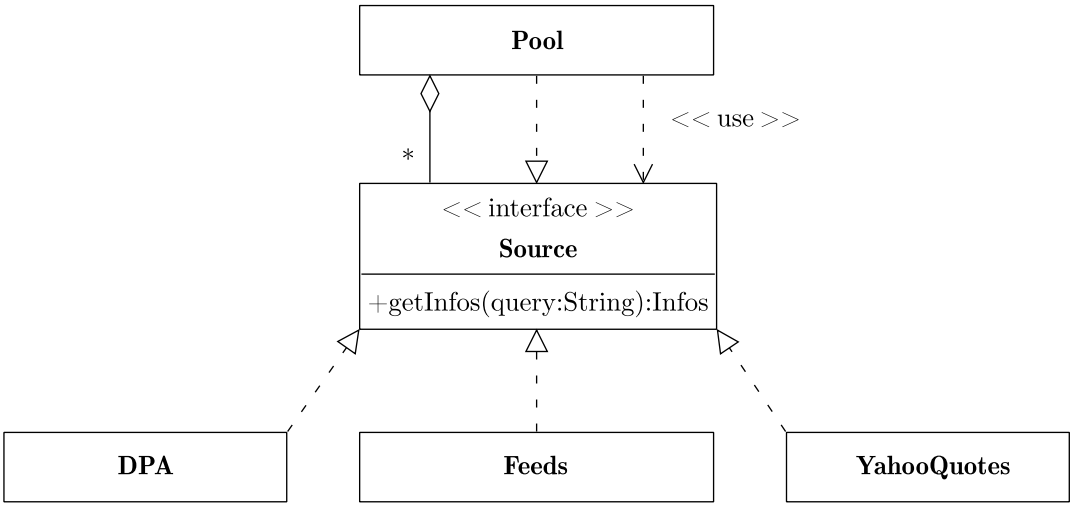
5.4.2 Quellen

Implementierung der Source-Schnittstelle

Wie schon in Unterkapitel 5.2 festgehalten wurde, wird für jede Quelle eine eigene Komponente erstellt. In Abbildung 5-13 (S. 61) werden diese Komponenten anhand dreier Beispiele[67] im unteren Teil des Diagramms dargestellt.

Jede Quell-Komponente muss die `Source`-Schnittstelle anbieten, über die der Pool die Daten aus der Quelle abrufen kann. Der Pool aggregiert somit die übrigen Quell-Komponenten über die `Source`-Schnittstelle. Abbildung 5-14 stellt diesen Zusammenhang beispielhaft dar.

Abb. 5-14: UML-Klassendiagramm für die Source-Schnittstelle



Üblicher Aufbau einer Quell-Komponente

Eine Quell-Komponente ist, wie in Unterkapitel 5.2 beschrieben, sehr autonom bei der Bewerkstelligung ihrer Aufgaben. Die einzige Verpflichtung, die sie eingeht, ist die Implementierung der `Source`-Schnittstelle.

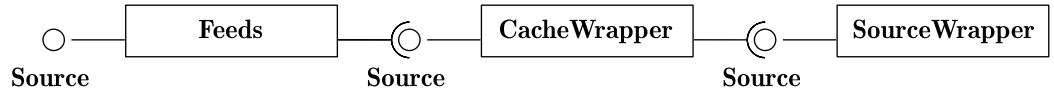
Dennoch wurde in Abschnitt 4.1.3 gezeigt, dass eine hybride Integration eine gute Grundlage zum Zugriff auf die Quellen bildet. Für sehr viele Quellen wird diese Vorgehensweise sehr nützlich sein. Im Wesentlichen sieht die hybride Integration den Einsatz eines optionalen Zwischenspeichers vor, der vor die externe Quelle geschaltet wird.

Zum Zugriff auf die Quelle und den Zwischenspeicher werden jeweils entsprechende Wrapper benötigt, die die Abfrage entgegennehmen, für die Quelle bzw. für den

[67] Im Folgenden werden stets die drei Beispiel-Quellen DPA, Feeds und Yahoo Quotes verwendet. In der konkreten Implementierung des Systems existiert zusätzlich eine Komponente AFP. Tatsächlich können natürlich fast beliebige Quellen angebunden werden.

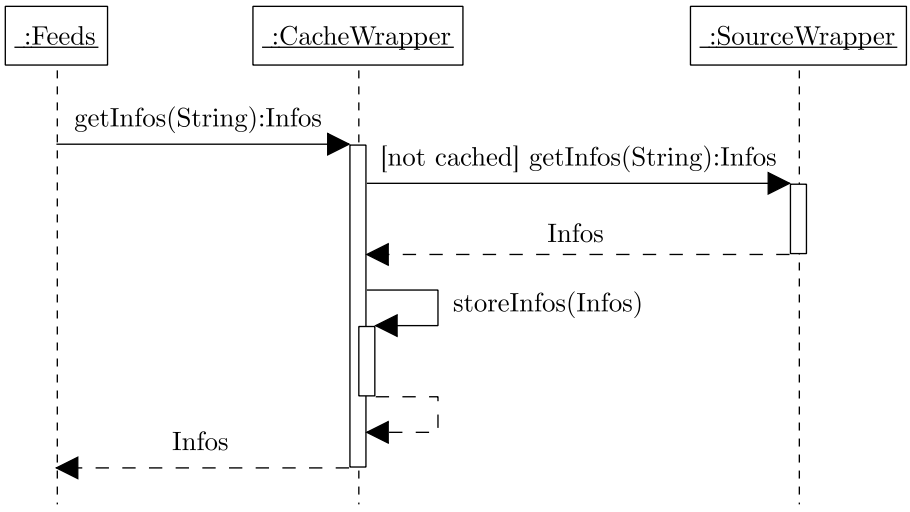
Zwischenspeicher übersetzen und die Ergebnisse in das gewünschte Ausgabeformat transformieren. Für diese Wrapper werden eigene Klassen erstellt. In Abbildung 5-15 wird dieser Sachverhalt beispielhaft für eine Quelle dargestellt.

Abb. 5-15: UML-Klassendiagramm für eine Quelle mit ihren Wrappern



Dabei greift die Klasse, die eine Quell-Komponente repräsentiert, zunächst über einen `CacheWrapper` auf den Zwischenspeicher zu. Falls die Abfrage nicht aus dem Zwischenspeicher beantworten werden kann, so stellt der `CacheWrapper` wiederum eine Abfrage an den `SourceWrapper`. Der `SourceWrapper` übersetzt die Abfrage in eine Abfrage an die externe Quelle, nimmt die Rohergebnisse entgegen, extrahiert ggf. die relevanten Daten und transformiert sie in das das gewünschte Format. Der `CacheWrapper` nimmt die Ergebnisse entgegen, speichert sie möglicherweise im Zwischenspeicher und liefert das Ergebnis an die übergeordnete Quell-Komponente aus. Dieser Ablauf wird in Abbildung 5-16 dargestellt.

Abb. 5-16: UML-Sequenzdiagramm für die Interaktion zwischen der Quelle und ihren Wrappern



Auch hier wird wieder die `Source`-Schnittstelle genutzt, um zwischen der Quelle und ihren Wrappern zu kommunizieren.

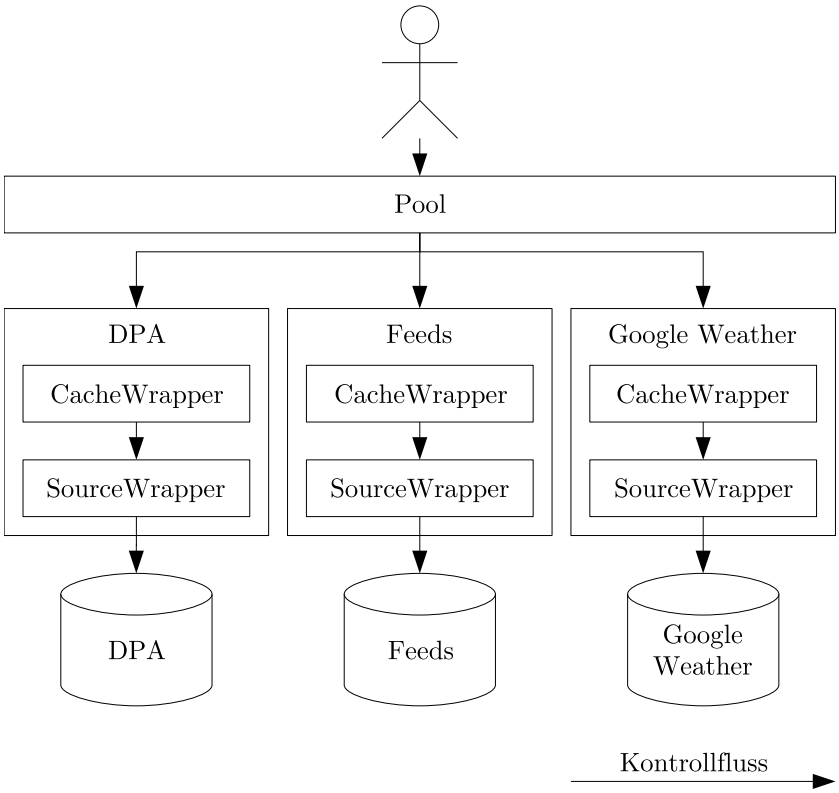
Diese Organisation, die sich an der hybriden Integration orientiert, ist entsprechend dem Prinzip autonomer Quell-Komponenten nur ein möglicher Vorschlag, der jedoch sehr flexibel ist und daher für eine Vielzahl an Quellen geeignet sein wird. Für

Quellen, für die die hybride Integration nicht geeignet ist, können beliebige andere Verfahren genutzt werden, solange die Quell-Komponente die `Source`-Schnittstelle implementiert.

Schichtung

Eine Besonderheit der Kommunikation zwischen den Komponente bzw. deren Klassenrepräsentationen sowie deren untergeordnete Klassen ist, dass die Methodenaufrufe stets von einer Klasse zu einer untergeordneten Klasse erfolgen und nicht umgekehrt. Es ergibt sich eine Schichtung der Komponenten bzw. Klassen, wie man Abbildung 5-17 entnehmen kann.

Abb. 5-17: Schichtung der Komponenten bzw. Klassen



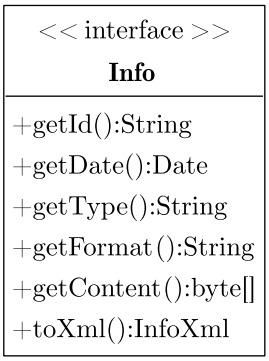
Die Schichtung führt zu unidirektionale Abhängigkeiten, die wiederum zu einer Verringerung der Komplexität, einer potenziell einfacheren Testbarkeit und einer besseren Verständlichkeit des Systems.

5.5 Verwaltung der Informationseinheiten

5.5.1 Schnittstellen

Die verarbeiteten Daten können strukturell höchst unterschiedlich sein (siehe Anforderung 2 (S. 26)). Um sie dennoch systemweit nutzbar zu machen, wird eine Schnittstelle definiert, die den kleinsten gemeinsamen Nenner für alle Informationseinheiten vorschreibt. Diese Schnittstelle (`Info`) zeigt Abbildung 5-18.

Abb. 5-18: UML-Klassendiagramm für die Schnittstelle `Info`



Jede konkrete Klasse, die eine Informationseinheit repräsentiert, muss diese Schnittstelle implementieren. Sie schreibt vor, dass jede Informationseinheit einen eindeutigen Bezeichner (*ID*), ein Datum, einen Typ, ein Format und einen Inhalt haben muss.

Da die Quellen autonom agieren, soll der Bezeichner per Konvention mit dem Paketnamen der Quell-Komponente beginnen, um Kollisionen zu vermeiden. Der Typ ist frei wählbar und soll die Informationsart beschreiben. Beispiele wären „newsarticle“ oder „image“. Das Format ist ein *MIME Media Type*[68], der beschreibt, wie der Inhalt formatiert ist.

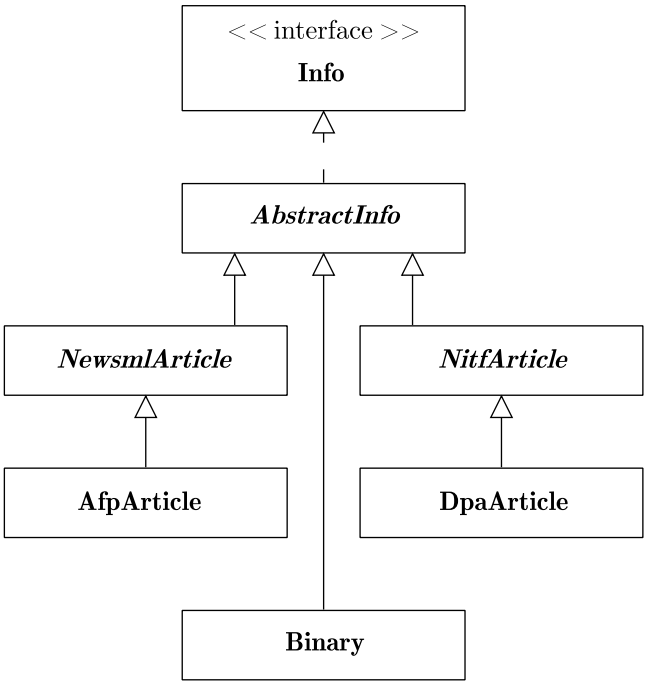
Der Inhalt selbst wird auf dieser Abstraktionsebene bloß als Byte-Array betrachtet. Eine genauere Interpretation wird von der Schnittstelle zunächst nicht verlangt. Die konkreten Informationsklassen sind dafür verantwortlich, ihren Inhalt zu interpretieren und evtl. umzuformen.

[68] Der Aufbau eines *MIME Media Type*s wird in Freed u. Borenstein (1996) beschrieben. Eine Liste der registrierten MIME Media Types kann man unter `http://www.iana.org/assignments/media-types/` (Abruf: 2007-09-21) einsehen.

5.5.2 Informationsklassen

Jede Informationseinheit wird durch eine Instanz einer dem Austauschformat entsprechenden Klasse repräsentiert. Für jedes Austauschformat wird eine eigene Klasse erstellt, wie es Abbildung 5-19 beispielhaft veranschaulicht.

Abb. 5-19: UML-Klassendiagramm für Beispiele von Informationsklassen



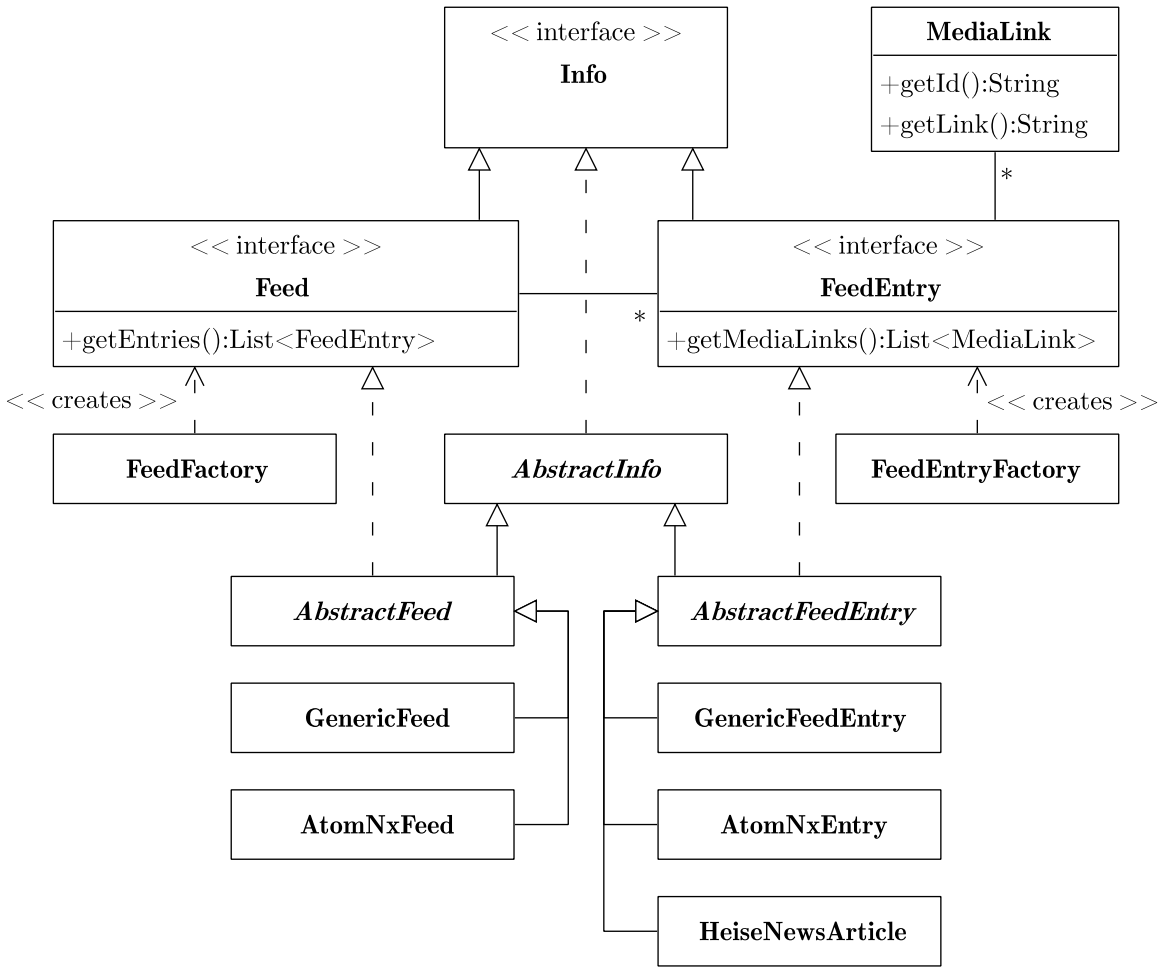
Wie schon erwähnt, müssen alle Informationsklassen die Schnittstelle `Info` implementieren. Diese Schnittstelle steht daher an der Spitze des Diagramms. Sie wird i.d.R. zunächst von abstrakten Klassen (z.B. `AbstractInfo`) implementiert, die gemeinsame Funktionalität für ähnliche Erweiterungsklassen beinhalten. Die abstrakten Klassen wiederum werden durch konkrete Klassen (z.B. `Binary`) erweitert, die die konkreten Informationseinheiten darstellen.

Wie man an der Hierarchie `DpaArticle` → `NitfArticle` → `AbstractInfo` erkennen kann, sind auch durchaus mehrere Abstraktionsebenen denkbar.

Ebenfalls kann es sinnvoll sein, die Schnittstelle `Infos` durch weitere Schnittstellen zu erweitern, was in Abbildung 5-20 (S. 67) dargestellt wird.

Die Schnittstellen `Feed` und `FeedEntry` sind Erweiterungen der Schnittstelle `Info`. Sie enthalten zusätzliche Methoden, die von jeder implementierenden Klasse erwartet werden. So muss jeder `Feed` eine Methode `getEntries():List<FeedEntry>` bereitstellen, die alle in diesem Feed enthaltenen Einträge zurückgibt. Ähnlich muss jeder `FeedEntry` eine Methode `getMediaLinks():List<MediaLink>` bereitstellen,

Abb. 5-20: UML-Klassendiagramm für die Organisation von Feeds und ihren Einträgen



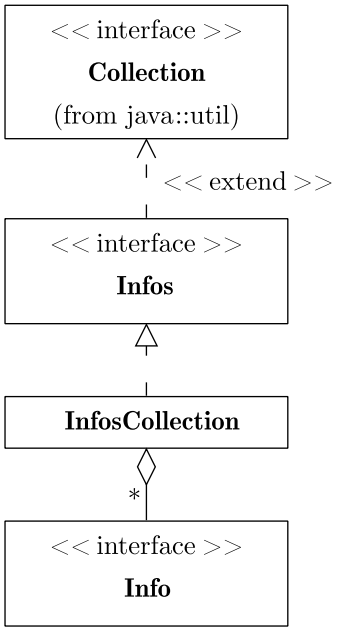
die alle in diesem Eintrag referenzierten Medienobjekte (z.B. Fotos) zurückgibt.[69] Abbildung 5-20 (S. 67) verdeutlicht u.a. diesen Zusammenhang.

Ein weiteres Entwurfsdetail, das man diesem Diagramm entnehmen kann, sind die Klassen `FeedFactory` und `FeedEntryFactory`. Sie realisieren jeweils das Entwurfsmuster einer einfachen Fabrik[70]. Die `FeedFactory` erzeugt anhand einer URL Instanzen von Klassen, die die `Feed`-Schnittstelle implementieren. Analog erzeugt die `FeedEntryFactory` anhand einer URL Instanzen von Klassen, die die `FeedEntry`-Schnittstelle implementieren.

5.5.3 Container-Klasse

Wie weiter oben beschreiben, gibt die Methode `getInfos` der Schnittstelle `Source` die Ergebnisse der Abfrage in einer Instanz des Typs `Infos` zurück. Es handelt sich um eine Schnittstelle, die mehrere `Info`-Instanzen beinhalten kann. Abbildung 5-21 veranschaulicht diese Schnittstelle und eine implementierende Klasse.

Abb. 5-21: UML-Klassendiagramm für einen `Info`-Container



Die Schnittstelle `Infos` erweitert eine allgemeine Schnittstelle für eine Menge an Objekten (in diesem Falle die Java-Schnittstelle `java.util.Collection`). Sie wird von

[69] Jede Medien-Referenz besteht aus ihrer Zieladresse und einem eindeutigen Bezeichner, unter dem das Medienobjekt intern abgelegt werden soll. Diese Referenz wird dazu genutzt, das Medienobjekt in einem internen Zwischenspeicher abzulegen, um nicht auf einen entfernten Server angewiesen zu sein.

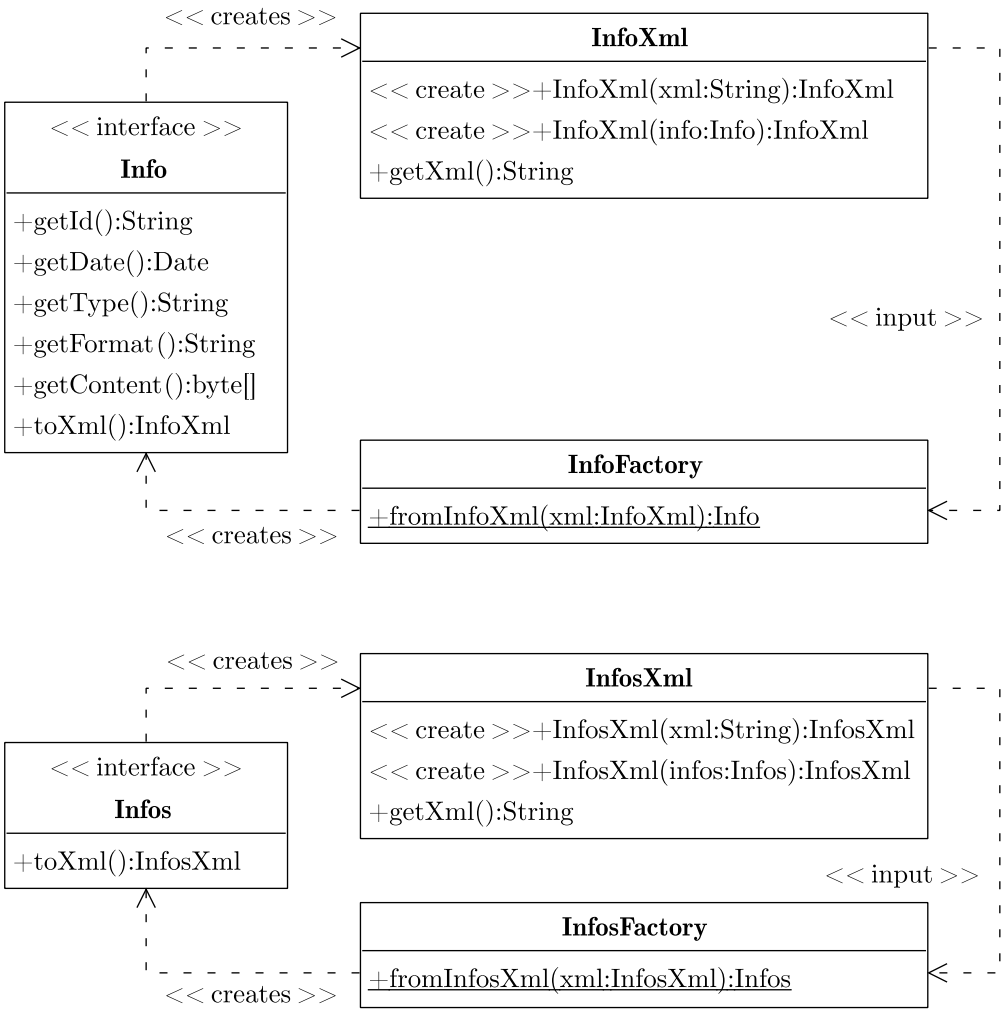
[70] Für eine Beschreibung der Fabrikmethode (*Factory Method*) siehe Gamma u. a. (1995), S. 107 ff. oder speziell für die einfache Fabrik (*Simple Factory*) Freeman u. a. (2004), S. 117.

der konkreten Klasse `InfosCollection` implementiert, die mehrere Informationseinheiten aggregiert. Die Informationseinheiten können Instanzen unterschiedlicher Klassen sein, die jedoch alle die Schnittstelle `Info` implementieren.

5.5.4 Serialisierung

Um die Informationseinheiten abspeichern und zwischen unterschiedlichen Programmen übertragen zu können, müssen die Objekte in ein portables Format serialisiert werden. Insbesondere zum Austausch von Daten zwischen unterschiedlichen Systemen ist XML[71] gut geeignet. Daher sollen alle Informationseinheiten, sowie alle Sammlungen von Informationseinheiten eine Methode zur Serialisierung nach XML bieten. Abbildung 5-22 veranschaulicht, wie sich diese Forderung auf die Schnittstellen und Klassen auswirkt.

Abb. 5-22: UML-Klassendiagramm zur Verdeutlichung der Serialisierung von Objekten nach XML.



[71] Für die Spezifikation von XML 1.0 siehe World Wide Web Consortium (W3C) (2002).

Die Schnittstellen `Info` und `Infos` definieren eine Methode `toXml()`, die eine serialisierte Version (`InfoXml` bzw. `InfosXml`) der Instanz erstellt und zurückgibt. Die Deserialisierung erfolgt durch die `InfoFactory` bzw. die `InfosFactory`. Diese einfachen Fabriken erzeugen eine Instanz einer Klasse, die der Serialisierung entspricht. Welche Klasse konkret instanziert wird, entscheidet die Fabrik anhand des Formats und evtl. des Bezeichners.

Die XML-Serialisierung folgt einem einfachen Schema, das die Struktur der `Info`- bzw. `Infos`-Schnittstelle auf XML-Elemente abbildet. Die Schema-Definition in Form einer RELAX-NG-Compact[72]-Grammatik (RNC) befindet sich im Anhang in Abschnitt 9.2.1.

Die Grammatik definiert sowohl die `Info`- als die `Infos`-Serialisierung. Binärdaten sollten Base64-kodiert werden. Ein einfaches, gültiges[73] Beispiel einer `Infos`-Serialisierung könnte wie folgt aussehen:

```
<infos count="4" xmlns="http://gedankenkonstrukt.de/infopool/schema/info">
    <info>
        <id>unique234823749</id>
        <date>2007-09-12T22:49:35</date>
        <type>some binary text</type>
        <format>text/plain</format>
        <content size="3">Zm9v</content>
    </info>
    <!--
    <info>...</info>
    <info>...</info>
    <info>...</info>
    -->
</infos>
```

[72] Für die Spezifikation der RELAX NG Compact Syntax siehe The Organization for the Advancement of Structured Information Standards (OASIS) (2002), für eine Einführung siehe The Organization for the Advancement of Structured Information Standards (OASIS) (2003).

[73] Die Validierung von XML-Dateien gegen ein RELAX-NG-Schema kann man beispielsweise mit Jing durchführen. Jing kann man von der Seite `http://www.thaiopensource.com/relaxng/` (Abruf: 2007-09-22) herunterladen.

5.6 Transformation der Informationseinheiten

In den Anforderungen wurde festgehalten, dass das Integrationssystem Daten in unterschiedlichen Austauschformaten verarbeiten (siehe Anforderung 2 (S. 26)) und sie zur Verringerung des Verarbeitungsaufwandes beim Benutzer durch Transformationen vereinheitlichen (siehe Anforderung 4 (S. 26)) soll.

Die Eingabeformate ergeben sich aus den anzubindenen Quellen. Sie wurden bereits in Unterkapitel 2.4 aus fachlicher Sicht betrachtet.

Im Folgenden werden Zwischenformate vorgeschlagen, in die die Eingabedaten transformiert werden. Anschließend werden die Transformationen in den Software-Entwurf eingeordnet.

5.6.1 Untersuchung der Eingabeformate

Die Daten können in sehr unterschiedlichen Austauschformaten geliefert werden. In Unterkapitel 4.4 wurde gezeigt, dass es zur Verringerung des Aufwands der Verarbeitung der Daten sinnvoll ist, die Anzahl der Austauschformate zu reduzieren. Dabei sollen Informationen mit ähnlichen Formaten in ein möglichst einfaches Zwischenformat transformiert werden. Semantische Verluste können (und sollten) in Kauf genommen werden, wenn dies aus fachlicher Sicht keine nennenswerte Beeinträchtigung darstellt.

Zum Austausch von **Nachrichtenartikeln** sind insb. die durch das IPTC spezifizierten Formate NITF[74] und NewsML[75] von Bedeutung. Wie man der Übersicht über Quellen für Nachrichtenartikel in Tabelle 9-6 (S. 113) entnehmen kann, liefern AFP, ddp und Reuters ihre Daten im NewsML-Format aus. Die Artikel der dpa werden im NITF-Format angeboten.

Tatsächlich haben beide Formate gewisse Ähnlichkeiten. Bei beiden Formaten handelt es sich XML-Dialekte, die Nachrichtenartikel und mit ihnen verbundene Metadaten enthalten. Sie sind beide den semi-strukturierten Daten unterzuordnen, da sie gewisse strukturelle Freiheiten im Dokumentenaufbau erlauben. Obwohl NewsML nach NITF entstanden ist und viele Bereiche des NITF abdeckt, sind die Formate komplementär zu betrachten.

NITF erfüllt zwei Aufgaben: **Strukturierung** von Nachrichteninhalten (Titel, Untertitel, angereicherter Text) und **Definition** von Metadaten für die Inhalte. Der

[74] Die Spezifikation des *News Interchange Text Formats* (NITF) kann unter International Press Telecommunications Council (2007b) eingesehen werden.

[75] Für die Spezifikation der *News Markup Language* (NewsML) siehe International Press Telecommunications Council (2007a).

Text kann neben den üblichen strukturellen Auszeichnungen[76] auch semantische Auszeichnungen wie die Kennzeichnung einer Organisation, einer Person oder eines Ortes enthalten. Ebenso können externe Medienobjekte eingebunden werden. Die Metadaten enthalten neben Datum, Herausgeber, Kategorien und Titel auch Angaben zum Workflow, wie z.B. die Dringlichkeit oder den aktuellen Status.

NewsML ist ein Container-Format für Nachrichteninhalte. Ein NewsML-Dokument besteht im Wesentlichen aus Metadaten sowie mehreren Inhalts-Komponenten. Diese Komponenten können Inhalt beliebigen Typs enthalten. Mehrere Text-Komponenten (evtl. in unterschiedlichen Sprachen), Fotos oder Videos können in einer NewsML-Instanz auftreten. Sie können beliebig verschachtelt werden und sich untereinander referenzieren. Abgesehen von bestimmten Informationen über die Komponenten wie Titel und Datum, beinhaltet NewsML keine Elemente zur Strukturierung von Textinhalten. Text-Komponenten werden i.d.R. in NITF[77] oder XHTML ausgezeichnet.

Insbesondere Online-Nachrichtendienste liefern ihre Artikel über **Feeds** aus. Im Wesentlichen handelt es sich um RSS-Varianten und Atom-Feeds.[78] Die Feed-Einträge sind ähnlich zu NewsML auch bloß als Container zu betrachten. Sie definieren ein Schema für Metadaten wie Titel und Datum. Die Inhalte können in beliebigen Formaten vorliegen. Sie können referenziert oder eingebettet werden. Eingebettete Textinhalte liegen üblicherweise im HTML- oder XHTML-Format vor.

Vergleicht man diese Formate, so stellt NewsML das flexibelste, aber auch komplexeste Format dar. Insbesondere die beliebige Verschachtelung der Komponenten führt zu recht komplizierten Dokumenten. Ein NITF-Dokument enthält hingegen genau eine Text-Komponente, die evtl. mehrere Medienobjekte referenziert. Ähnlich wie ein Feed-Eintrag, der auch eine Text-Komponente vorsieht, die evtl. mehrere Medienobjekte referenziert.[79]

---

[76] Konkret können NITF-Textblöcke Absätze, Überschriften zweiter Ordnung, Tabellen, Listen, Medienobjekte, Zitate, Fußnoten, Trennlinien, vorformatierten Text und Kommentare enthalten. Sie entsprechen damit in weiten Teilen einer Untermenge von XHTML 1.0 Strict.

[77] Genauer genommen wird hier nur der `body.content`-Teil eines NITF-Dokuments – auch bc-NITF genannt – genutzt. Die Metadaten werden im NewsML-Dokument ausgezeichnet. Das ist der Weg, den das IPTC empfiehlt.

[78] Für eine Liste alle üblichen RSS- und Atom-Varianten sowie Verweisen zu ihrer Spezifikation siehe Abschnitt 2.4.2.

[79] Feed-Einträge können zusätzlich sog. *Enclosures* enthalten. Dabei handelt es sich meistens um Audio- oder Video-Dateien.

5.6.2 Gemeinsames Zwischenformat für Nachrichtenartikel

Da sich die oben beschriebenen Formate in gewissen Teilen ähneln, liegt es nahe, sie in ein gemeinsames Zwischenformat zu überführen. Da die Formate für die Metadaten weitgehend unabhängig von denen für die Inhalte sind, müssen effektiv zwei Entscheidungen getroffen werden.

Zunächst erscheint es sinnvoll, NewsML als Container-Format zu nutzen. Es ist das mächtigste der untersuchten Austauschformate und sollte somit problemlos die anderen Formate abdecken können. Allerdings hätte diese Vorgehensweise einen gravierenden Nachteil: NewsML ist, wie schon oben festgestellt wurde, ein sehr komplexes Format. Die Nutzung von NewsML als Zwischenformat würde dem Entwurfsziel der Einfachheit sowie dem eigentlichen Ziel der Transformation – nämlich die Erleichterung der Verarbeitung der Daten – zuwiderlaufen. Ein schlankeres Format wäre hier erstrebenswert.

Ermittlung tatsächlich benötigter Strukturelemente

Also wurden zur Ermittlung der tatsächlich benötigten Strukturelemente reale Beispieldaten untersucht. Die Untersuchung umfasste gut 300 NewsML-Instanzen aus dem AFP Live-Katalog[80], gut 470 NewsML-Instanzen aus einem Beispielpaket von Reuters[81] sowie über 3300 NITF-Instanzen aus einem Beispielpaket der dpa[82]. Zur Untersuchung der Beispieldaten wurde ein einfaches Tool in Python geschrieben, das die XML-Bäume mehrerer Dateien lädt und für jedes Element im Baum die Häufigkeit des Auftretens dieses Elements grafisch darstellt. Die Ausgabe für die NewsML- und NITF-Beispieldaten befinden sich im Anhang in Unterkapitel 9.3.

Es wurde deutlich, dass die NewsML-Instanzen, wie erwartet, wesentlich komplexer sind als die NITF-Instanzen. Das liegt zum Teil auch sicherlich daran, dass Instanzen von zwei verschiedenen Anbietern untersucht wurden, die ihre Dokumente unterschiedlich strukturieren. So liegt bei den Instanzen von Reuters der Textinhalt in dreifach verschachtelten `NewsComponent`s, bei den AFP-Instanzen liegen sie in der zweiten Ebene. Ebenfalls kann schließen, dass die benutzten Metadaten für das Gesamtdokument hauptsächlich die Kategorisierung, Identifikation und das Datum betreffen. Textinhalte sind eingebettet und Medienobjekte liegen in externen Dateien. Die Inhaltskomponenten haben, neben einem eventuellen Verweis auf eine

---

[80] Den AFP Live-Katalog erreicht man unter der Adresse `http://www.united-news.de/` (Abruf: 2007-09-22). Man kann sich dort für einen kostenlosen Testzugang anmelden.

[81] Mehrere NewsML-Beispielpakete von Reuters kann man sich von der Seite `http://about.reuters.com/newsml/newsmldemo.asp` (Abruf: 2007-09-22) herunterladen.

[82] Das NITF-Beispielpaket der dpa wurde auf Anfrage per E-Mail zur Verfügung gestellt.

externe Datei, Überschriften unterschiedlicher Ebenen sowie Angaben zur Herkunft (Verfasser, Anbieter) des Inhalts.

Die NITF-Instanzen sind wesentlich einfacher obwohl sie Inhalte gleichen Typs – nämlich um Bilder angereicherte Nachrichtenartikel – enthalten. Tatsächlich enthalten sie neben dem Inhalt praktisch die selben Metadaten wie die NewsML-Instanzen. Aufgrund des beschränkteren Einsatzgebiets von NITF fallen die Dokumente jedoch wesentlich knapper aus.

Anhand dieser Ergebnisse und einer genaueren Untersuchung einzelner Exemplare kann man schlussfolgern, dass gängige Nachrichtenartikel (abgesehen vom Inhalt) folgende, im Kontext eines IIDs benötigte Strukturelemente benötigen:

- Eindeutiger Bezeichner (ID)
- Veröffentlichungsdatum
- Aktualisierungsdatum
- Anbieter / Agentur
- Copyright
- Sprache
- Titel, Untertitel, Unteruntertitel
- Zusammenfassung
- Autoren
- Orte
- Kategorien
- Inhalt
- Medienobjekte

Obwohl NewsML all diese Elemente aufweist, soll es aufgrund der hohen Komplexität nicht als Zwischenformat genutzt werden. NITF ist wesentlich einfacher, erlaubt jedoch nur NITF zur Auszeichnung der Inhalte und ist relativ beschränkt bei der Angabe von Metadaten zu referenzierten Medienobjekten, dafür bietet es viele Elemente, die tatsächlich nicht benötigt werden.

Atom/NX als geeignetes Zwischenformat

Erstaunlich nah an die erforderliche Struktur kommt das Atom-Format[83], das zudem durch seine Einfachheit bevorteiligt ist. Außerdem ist es als erweiterbares Format konzipiert: Zusätzliche Elemente aus anderen XML-Namensräumen können an praktisch allen Stellen des Dokuments eingefügt werden und es kann Textinhalte in beliebigen Formaten transportieren. Zudem definiert es nicht nur Feeds, sondern

[83] Für die Spezifikation des Atom Syndication Format siehe Nottingham u. Sayre (2005).

erlaubt auch Dokumente, die nur aus einem alleinstehenden `<entry>` bestehen. Aufgrund der Einfachheit und Erweiterbarkeit wurde das Atom-Format als Grundlage für ein Zwischenformat für Nachrichtenartikel gewählt.

Die vom Atom-Format nicht unterstützten Elemente sind der Untertitel, Unteruntertitel, Anbieter, Orte sowie um Metadaten angereicherte Medienobjekte. Um das Format auf für weitere Informationsarten nutzen zu können, soll zusätzlich ein frei wählbarer Typ der Informationseinheit (z.B. „news-article“, „blog-entry“) definierbar sein. Ein Atom-Eintrag lässt leicht um diese Elemente erweitern. Das entsprechende RELAX-NG-Schema befindet sich im Anhang in Abschnitt 9.2.2.

Das Schema erlaubt es, einen Atom-Feed oder -Eintrag mit Elementen für Typ, Untertitel, Unteruntertitel, Anbieter und Ort auszuzeichnen. Zusätzlich kann ein Atom-Eintrag mehrere Medienobjekte enthalten, die wiederum Titel, Untertitel usw. enthalten können. Ein Minimalbeispiel könnte wie folgt aussehen:

```
<entry xmlns="http://www.w3.org/2005/Atom"
 xmlns:nx="http://gedankenkonstrukt.de/infopool/schema/nx">
    <id>urn:uuid:234213374711</id>
    <nx:type>blog-entry</nx:type>
    <title>Look! Atom with news extensions!</title>
    <updated>2007-09-13T12:12:52Z</updated>
    <content type="xhtml">
        <div xmlns="http://www.w3.org/1999/xhtml">
            Hello World!
            <object data="#photo0" />
        </div>
    </content>
    <nx:media>
        <id>photo0</id>
        <nx:type>image</nx:type>
        <content type="image/jpeg" src="foobar.jpg" />
    </nx:media>
</entry>
```

5.6.3 Zwischenformat für strukturierten Text

Bislang wurde nur die Struktur eines Dokuments mit seinen Metadaten untersucht. Wie der eigentliche Textinhalt strukturiert sein soll, wurde noch nicht betrachtet.

Wie schon erwähnt wurde, ist es in NewsML-Dokumenten üblich, den Text mittels XHTML oder NITF zu formatieren. In NITF-Dokumenten muss der Text ebenfalls mit NITF-Elementen ausgezeichnet werden. In Feeds oder auf Web-Seiten sind HTML oder seltener XHTML üblich.

Will man bei einem XML-Dialekt bleiben, stellt sich also im Wesentlichen die Frage, ob man XHTML oder NITF zur Textformatierung nutzen möchte. Tatsächlich haben sie eine relativ große Schnittmenge an Elementen, da sich NITF stark an XHTML orientiert. NITF hat jedoch einige zusätzliche Elemente, die eine Anreicherung des Textes um semantische Informationen[84] ermöglicht. So kann man bspw. Orte, Organisationen oder Personen im Text als solche kennzeichnen. XHTML sieht solche semantischen Auszeichnungen in diesem Ausmaß nicht vor, man würde also einen semantischen Verlust in Kauf nehmen müssen, wenn man Texte aus dem NITF-Format nach XHTML transformieren würde[85].

Wie die Analyse der Beispieldaten ergeben hat, wird von diesen semantischen Auszeichnungen in der Praxis jedoch kein Gebrauch gemacht. Es werden ausschließlich Elemente für Absätze, Überschriften, Hyperlinks, Fettdruck, Listen, und Tabellen benutzt. Eine zusätzliche Untersuchung einer Ausgabe der FAZ[86] und des Kölner Stadt-Anzeigers[87] hat bestätigt, dass diese Auswahl an Strukturelementen zur Textauszeichnung ausreichend ist. Somit sollte XHTML 1.0 Strict zur Textauszeichnung geeignet sein. Die Bekanntheit und große Werkzeugunterstützung spricht ebenfalls für den Einsatz von XHTML.

5.6.4 Zwischenformate für anderweitige Informationsarten

Auch wenn die Eignung des Formats Atom/NX+XHTML bislang nur für Nachrichtenartikel gezeigt wurde, so ist es ebenso problemlos auf **Blog-Einträge** anwendbar, da sie der eigentliche Hauptanwendungsbereich von Atom sind.

Das Atom-Format ist aber auch darüber hinaus sehr vielseitig einsetzbar. So existieren u.a. bereits Erweiterungen zur Erfassung von Diskussionsstrukturen[88] oder der

---

[84] In der NITF-Spezifikation spricht man von *Enriched Text*.

[85] Tatsächlich könnte man einige dieser semantischen Informationen auch in XHTML übertragen. Dazu setzt man diese Informationen in generische XHTML-Tags wie `span` oder `div` und versieht sie mit CSS-Klassen, die eine bestimmte Semantik implizieren. Beispiel: `Preis: <span class='currency euro'>23,42</span>`. Eine Fortführung dieser Technik mit standardisierten Klassen ist unter dem Begriff **Microformats** bekannt. Mehr Informationen zu Microformats findet man unter `http://microformats.org/about/` (Abruf: 2007-09-22).

[86] Es handelte sich um die Donnerstagsausgabe vom 02.08.2007.

[87] Es handelte sich um die Donnerstagsausgabe vom 09.08.2007.

[88] Die Spezifikation der *Atom Threading Extensions* findet man unter `http://www.ietf.org/rfc/rfc4685.txt` (Abruf: 2007-09-22).

Historie eines Eintrags[89]. Google Mail[90] bietet sogar eine Funktion zum Abholen der E-Mails[91] über einen Atom-Feed.

Trotzdem kann man natürlich nicht alle Informationsarten sinnvoll in einem einzigen Format unterbringen, was in Abschnitt 4.4.3 festgestellt wurde. Für Informationenarten, die kaum oder keine strukturellen Ähnlichkeiten mit dem Atom-Format haben, macht es also Sinn, **eigene Formate** zu definieren. Insbesondere anwendungsspezifische Formate von Diensten, die als persönliche Assistenz genutzt werden, erfordern möglicherweise neue Zwischenformate. Falls bestimmte Dienste ähnliche Formate nutzen sollten, so ist die Vorgehensweise jedoch vollkommen analog zur Vorgehensweise für Nachrichtenartikel: Ähnliche Formate lassen sich gut zu einem gemeinsamen Zwischenformat zusammenfassen.

Für die Entwicklung der Quell-Komponente zum Abruf von **Aktienkursen** wurde ein eigenes, einfaches XML-Format entwickelt, das Strukturelemente für die üblichen Werte wie den aktuellen Kurs, die Kursveränderung, das Datum, etc. enthält.

Nicht betrachtet wurden bisher **Medienobjekte** wie Foto, Audio und Video. Es handelt sich um unstrukturierte Daten. Sie werden einfach der Klasse `Binary` untergeordnet und unverändert gespeichert. Etwaige Metainformationen wie Titel oder Urheber werden im referenzierenden Dokument abgelegt[92]. Transformationen würden die Medienobjekte bloß von einer Binärdarstellung in eine andere überführen. Diese leicht durch entsprechende Software-Bibliotheken lösbare Aufgabe sei der Ausgabetransformation überlassen.

Ebenso wurden Werbeanzeigen zunächst nicht betrachtet. In Abschnitt 2.4.4 wurde festgestellt, dass eine Integration voraussichtlich nicht nötig ist. Trotzdem wäre auch hier die Schaffung eines einheitlichen Zwischenformats denkbar. Eine gute Grundlage dafür könnte das von der Ifra vorgeschlagene Format AdsML[93] bieten.

---

[89] Den Entwurf „Feed History: Enabling Incremental Syndication“ findet man unter `http://tools.ietf.org/html/draft-nottingham-atompub-feed-history-04` (Abruf: 2007-09-22).

[90] Den Dienst Google Mail (auch: Gmail) findet man unter `http://mail.google.com/` (Abruf:2007-09-22).

[91] Google Mail bietet die Möglichkeit an, seine E-Mails per Atom-Feed über die Adresse `https://mail.google.com/mail/feed/atom` (Abruf: 2007-09-22) abzuholen.

[92] Z.B. in einem `nx:media`-Abschnitt in einem Atom/NX-Dokument.

[93] Mehr Informationen zu AdsML findet man auf der offiziellen Homepage unter `http://www.adsml.org/` (Abruf: 2007-09-22).

5.6.5 Ausgabeformate

Bisher wurden mögliche Eingabeformate und geeignete Zwischenformate diskutiert. Der Zweck der Zwischenformate ist die möglichst einfache Weiterverarbeitung in der Integration nachgelagerten Prozessen.

Einen der wesentlichen nachgelagerten Prozesse stellt die Ausgabe der Daten dar. Sie ist nicht Teil des Integrationssystems, das lediglich zur Aufgabe hat, eine Ausgabe in unterschiedlichen Formate zu ermöglichen. Die Wahl einfacher, bekannter, auf XML basierender Formate wie z.B. Atom und XHTML zur Speicherung der Daten sollte eine Ausgabe in unterschiedliche Formate leicht möglich machen. Zumal es eine gute Werkzeugunterstützung[94] zur Transformation von XML-Daten in unterschiedliche Ausgabeformate gibt.

Aber auch andere nachgelagerte Prozesse, wie z.B. eine automatisierte Kategorisierung, profitieren von der Reduktion auf wenige einheitliche und einfache Formate.

5.6.6 Einordnung der Transformationen in den Software-Entwurf

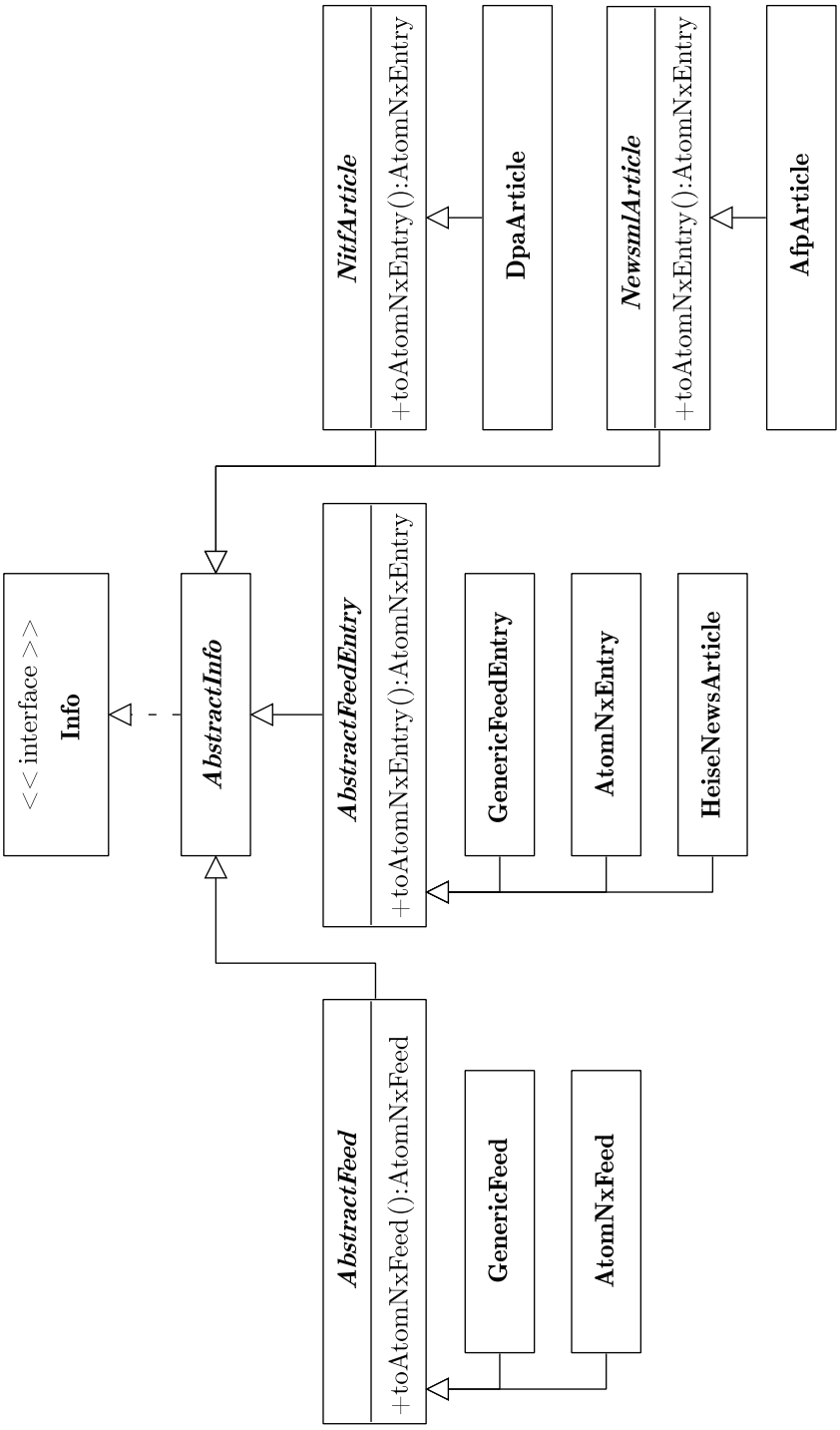
Zum Abschluss dieses Unterkapitels soll verdeutlicht werden, wo die Transformationen ihren Platz im Software-Entwurf haben. Wie in Abschnitt 5.5.2 beschrieben wurde, existiert für jedes Austauschformat eine eigene Klasse. Diese Klassen definieren nun neben den Methoden, die die `Info`-Schnittstelle vorgibt, Transformationsmethoden der Form `toFormatXY()`, die die jeweiligen Informationseinheiten in ein anderes Austauschformat transformieren. Abbildung 5-23 (S. 79) verdeutlicht das.

Wie man erkennen kann, können diese Transformationsmethoden auch durchaus aus übergeordneten, evtl. abstrakten Klassen geerbt werden, wenn sie für mehrere Unterklassen identisch sind.

Theoretisch könnte man auf gleiche Art und Weise auch Methoden zur Ausgabetransformation deklarieren. So könnte man z.B. eine Methode `toHtml()` definieren, die eine Informationseinheit in eine HTML-Repräsentation überführt. Die Implementierung einer solchen Funktionalität liegt aber außerhalb der Rahmens des zu entwickelnden Prototyps.

[94] So kann man XML-Dokumente über XSLT-Stylesheets recht einfach in HTML- oder XSL-FO-Dokumente (und dann in PDF- oder Postscript-Dokumente) überführen. Werkzeuge, die bei der Erstellung solcher Stylesheets behilflich sind, sind z.B. Stylus Studio oder Altova XMLSpy und Altova StyleVision.

Abb. 5-23: UML-Klassendiagramm zur Verdeutlichung der Transformationsmethoden der Informationsklassen



5.7 Systemschnittstellen

Das zu entwickelnde Integrationssystem ist primär zur Benutzung als Teil in einem Gesamtsystem gedacht. Daher muss es Schnittstellen anbieten, um von anderen Systemen genutzt zu werden. Diese Schnittstellen werden im Folgenden beschrieben.

5.7.1 Native Schnittstelle

Wie schon in Abschnitt 5.4.1 beschrieben wurde, stellt die `Pool`-Komponente die zentrale Stelle zum Zugriff auf alle Daten dar.

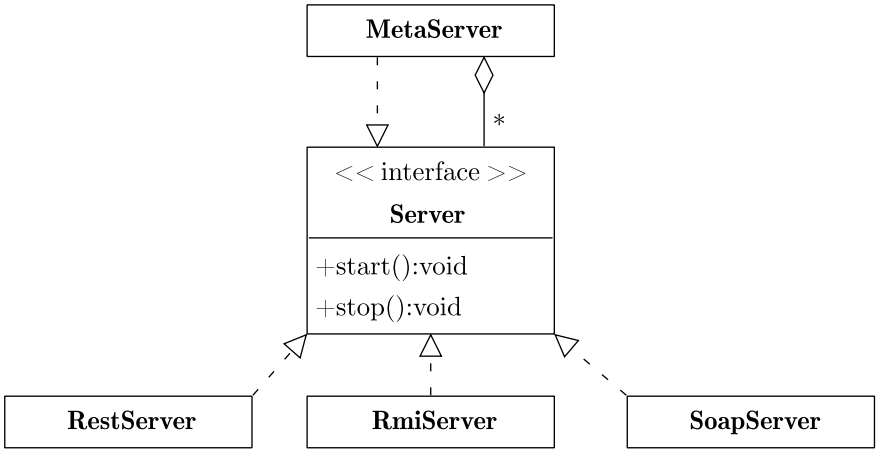
Wenn das Integrationssystem also nur in einem einzigen Programm genutzt wird, kann dieses Programm eine Instanz der `Pool`-Komponente[95] erstellen und hat somit einen direkten Zugriff auf die Abfrageschnittstelle.

5.7.2 Server-Schnittstellen

In der Praxis wird es wohl üblicher sein, dass mehrere Programme auf das Integrationssystem zugreifen. Daher wird eine Server-Komponente entwickelt, die den Kern des Integrationssystems einbettet und den Zugriff durch mehrere Clients erlaubt.

Um den Zugriff möglichst vielen externen Programmen zu ermöglichen, werden gleich mehrere Server-Schnittstellen angeboten: REST, RMI und SOAP. Abbildung 5-24 veranschaulicht den Entwurf für die Server-Schnittstellen. Ein übergeordneter `MetaServer` instanziert die konkreten Implementierungen der einzelnen Schnittstellen und startet bzw. stoppt sie, wenn er selbst gestartet bzw. gestoppt wird.

Abb. 5-24: UML-Klassendiagramm für die Server-Schnittstellen



[95] Wie in Abschnitt 6.1.1 beschrieben, wird das Integrationssystem in Java implementiert werden. Das instanzierende Programm müsste also auch in Java implementiert sein.

REST

Der **REST**[96]-Architekturstil basiert auf dem HTTP-Protokoll[97]. Die Kommunikation mit einem entfernten System geschieht über Verben, die auf Ressourcen angewendet werden. Die Verben entsprechen den HTTP-Methoden `PUT`, `GET`, `POST` und `DELETE` und können als eine Abbildung auf das gängige Schema Create, Read, Update, Delete (CRUD) verstanden werden. Die Ressourcen werden durch URLs gekennzeichnet, auf die die Verben angewendet werden.

Die Schnittstelle des Integrationssystem sieht lediglich die Abfrage von Daten vor (siehe Unterkapitel 5.3). Daher wird bloß das Verb `GET` zur Abfrage benötigt. Da die Abfrageschnittstelle keinen direkten Zugriff auf die Informationseinheiten (bzw. Ressourcen) vorsieht, müssen die Ressourcen stets über eine Abfrage abgerufen werden. Die REST-Schnittstelle entspricht dann dem folgendem einfachen Schema `http://server:port/query/<Abfrage>`.

Abfragen können mit jeden beliebigen HTTP-Client[98] abgesetzt werden. Die Ergebnisse werden als XML-Serialisierung geliefert. Für evtl. auftretende Fehler werden entsprechende HTTP-Status-Codes[99] verwendet.

Java RMI

Zusätzlich wird ein Zugriff auf die Abfrageschnittstelle über **Java RMI**[100] ermöglicht. RMI steht für *Remote Method Invocation* und erlaubt den Aufruf von Methoden von Objekten, die sich auf entfernten Systemen befinden können. RMI übernimmt dabei die (De-)Serialisierung der am Aufruf beteiligten Objekte und beinhaltet zusätzlich einen Mechanismus, über den sich die beteiligten Systeme die Java-Klassen herunterladen können, falls sie lokal nicht existieren.

Die Abfrageschnittstelle wird unter dem Objektnamen `infopool.server.rmi.RemoteSource` angeboten und implementiert die in Unterkapitel 5.3 vorgestellte Abfrageschnittstelle. In einem Java-Programm verhält sich ein Aufruf einer Methode mittels RMI praktisch identisch zum Methodenaufruf auf einem lokalen Objekt.

---

[96] Für die ursprüngliche Definition des REST-Architekturstils siehe Fielding (2000), S. 76 ff. Eine gute deutsche Einführung findet man in Mintert (2005), S. 63 ff.

[97] Für die Spezifikation des HTTP-Protokolls siehe The Internet Society (1999).

[98] Das kann auch ein Web-Browser sein.

[99] Für eine ungültige Abfrage würde bspw. der HTTP-Status-Code `400 Bad Request` mit einer entsprechenden Fehlermeldung zurückgegeben.

[100] Die Dokumentation zu RMI findet man neben weiteren Informationen unter `http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp` (Abruf: 2007-09-22).

Auch evtl. auftretende Ausnahmen (*Exceptions*) treten im Fehlerfalle[101] genau wie bei einem lokalen Aufruf auf.

SOAP

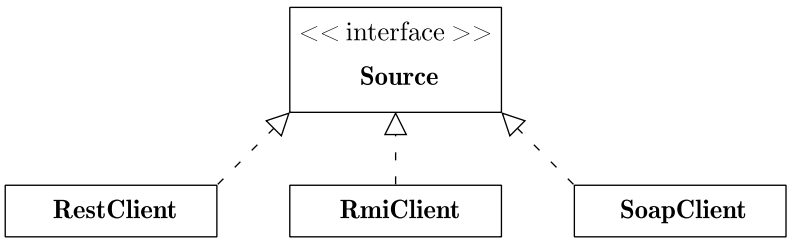
Schließlich wird die Abfrage über eine SOAP[102]-Schnittstelle angeboten. SOAP dient dem Austausch von Dokumenten (Document Style) zwischen Systemen oder dem Aufruf von entfernten Methoden (RPC Style). Die Schnittstellen werden in WSDL[103] definiert, die Aufrufe werden in SOAP-Nachrichten verpackt und i.d.R. per HTTP übermittelt[104], die Nutzdaten werden in XML-Form übertragen.

Die Abfrageschnittstelle ist unter `http://server:port/infopool/pool` als WebService im RPC-Stil zu erreichen. Die WSDL-Definition findet man unter `http://server:port/infopool/pool?wdsl`.

5.7.3 Generischer Client

Um den Zugriff auf die Schnittstellen des Integrationssystems zu vereinfachen, existiert eine Bibliothek, die alle drei Schnittstellen ansprechen kann und in einfachen Klassen kapselt. In Abbildung 5-25 werden diese Client-Klassen dargestellt.

Abb. 5-25: UML-Klassendiagramm der Client-Klassen



Alle Clients implementieren die bekannte `Source`-Schnittstelle und können somit genau wie jede andere Quelle angesprochen werden. Die Benutzung der Abfrageschnittstellen wird somit auf die Instanzierung einer Client-Klasse und den Aufruf der Schnittstellen-Methoden reduziert. Sie ist somit sehr einfach.

---

[101] Falls bei der Abfragebearbeitung ein Fehler auftritt, wird eine `QueryException` produziert.

[102] Für die Spezifikationen von SOAP 1.1 bzw. SOAP 1.2 siehe World Wide Web Consortium (W3C) (2007a) bzw. World Wide Web Consortium (W3C) (2007b).

[103] Für die Spezifikation von WSDL siehe World Wide Web Consortium (W3C) (2007c).

[104] SOAP sieht auch andere Transportmechanismen, wie z.B. E-Mails, vor. In der Praxis wird jedoch fast ausschließlich HTTP zur Übertragung genutzt.

6 Implementierung

6.1 Grundlegende Implementierungsentscheidungen

Bevor in den folgenden Unterkapiteln die Lösungen zu spezifischen Problemen diskutiert werden, soll zunächst ein Überblick über die für die gesamte Implementierung grundlegenden Entscheidungen gegeben werden.

6.1.1 Entwicklungsplattform

Die Entwicklungsplattform wird im Wesentlichen durch die eingesetzte Programmiersprache sowie die Systemplattform (Betriebssystem, Hardware) definiert.

Als Entwicklungsplattform wurde Java ausgewählt. Java ist derzeit eine der populärsten[105] Programmiersprachen. Die Java-Entwicklungsplattform ist zudem kostenlos erhältlich und genießt eine breite Unterstützung durch Werkzeuge und verfügbare Bibliotheken. Außerdem ist Java weitgehend plattformunabhängig, man muss sich also nicht auf ein bestimmtes Betriebssystem oder eine bestimmte Hardware festlegen. Aufgrund dieser Vorteile erscheint die Wahl von Java als Entwicklungsplattform sinnvoll.

Konkret wurde die Java 6.0 Standard Edition[106] verwendet. Der in dieser Arbeit erstellte Quelltext orientiert sich weitgehend an den Java Code Conventions[107]. Als Betriebssystem wurde Windows XP SP2 eingesetzt.

6.1.2 Werkzeuge

Zur produktiveren Software-Entwicklung ist der Einsatz von Werkzeugen ratsam. Das umfasst Entwicklungsumgebungen sowie unterstützende Tools.

Als Java-Entwicklungsumgebung wurde Eclipse 3.3[108] eingesetzt. Für die Erstellung von XML-Dateien und XSLT-Stylesheets diente Altova XMLSpy 2007[109]. Zur Validierung der RELAX-NG-Schemas wurde Jing[110] genutzt.

---

[105] Ein Popularitäts-Ranking von Programmiersprachen kann man unter `http://www.tiobe.com/tpci.htm` (Abruf: 2007-09-23) finden.

[106] Die Java 6.0 Standard Edition kann unter `http://java.sun.com/javase/downloads/` (Abruf: 2007-09-23) heruntergeladen werden.

[107] Die Java Code Conventions kann man unter `http://java.sun.com/docs/codeconv/CodeConventions.pdf` (Abruf: 2007-09-23) herunterladen.

[108] Die Entwicklungsumgebung Eclipse 3.3 kann unter `http://www.eclipse.org/downloads/` (Abruf: 2007-09-23) heruntergeladen werden.

[109] Testversionen der Altova-Produkte können unter `http://www.altova.com/download.html` heruntergeladen werden.

[110] Jing kann man von der Seite `http://www.thaiopensource.com/relaxng/` (Abruf: 2007-09-23) herunterladen.

6.1.3 Nutzung vorhandener Softwarekomponenten

Um die für einen Prototypen wichtige schnelle und günstige Entwicklung zu erreichen, sollte man möglichst viel Funktionalität durch die Nutzung vorhandener Softwarekomponenten abdecken. Neben dem Aspekt der Zeitersparnis haben vorhandene Komponenten oft eine höhere Qualität, da sie schon von einer großen Nutzerschaft erprobt wurden.

Konkret wurden mehrere Open-Source-Bibliotheken[111] genutzt. Sie sind für die Java-Plattform äußerst zahlreich vorhanden, was wiederum einen weiteren wichtigen Grund für die Wahl von Java als Plattform darstellt.

[111] Eine Übersicht der genutzten Werkzeuge und Bibliotheken findet man im Anhang in Unterkapitel 9.4.

6.2 Implementierungsdetails

Die folgenden Abschnitte beschreiben Lösungen für die spezifischen Probleme, die sich bei der Umsetzung der in dieser Arbeit vorgeschlagenen Lösungsansätze ergeben haben. Da nicht jedes Detail erwähnt werden kann, werden bloß bestimmte Kernaspekte erläutert.

6.2.1 Abfragesprache

In Abschnitt 4.3.4 wurde eine eigene Abfragesprache zum Abruf der Daten aus dem Integrationssystem vorgestellt. Damit die Quell-Komponenten die Abfragen dieser Sprache verarbeiten können, müssen sie geparst[112] und ausgewertet bzw. übersetzt werden.

Zum Parsen der Abfragesprache wurde der Parser-Generator ANTLR[113] genutzt. ANTLR nutzt (ähnlich wie z.B. Yacc) eine um Programmcode angereicherte, BNF-ähnliche Grammatik, um daraus einen Parser zu generieren. Im Gegensatz zu z.B. Yacc und JavaCC, die LR-Parser generieren, generiert ANTLR einen sog. LL(*)-Parser. Der Asterisk in LL(*) sagt aus, dass die generierten Parser einen beliebig großen Look-Ahead durchführen können, was für den Menschen leichter zu schreibende (und zu lesende) Grammatiken ermöglicht.[114]

Zunächst wurde die in Abschnitt 4.3.4 vorgestellte Grammatik an die in ANTLR verwendete Sprache angepasst. [115] Die entwickelte Grammatik überführt eine konkrete Abfrage in einen Abstrakten Syntaxbaum[116]. Für die einfache Abfrage `a = 1 and b > 2 or c < 3` hat er den in Abbildung 6-26 (S. 86) dargestellten Aufbau.

Um den Baum in Java-Objekte zu überführen, wurde eine weitere ANTLR-Grammatik, eine sog. Baum-Grammatik(*Tree Grammar*), geschrieben. Diese Grammatik verarbeitet nicht mehr einen Zeichenstrom als Eingabe, sondern liest einen Baum ein und durchläuft ihn. Die Grammatik erlaubt die Definition von Java-Code, der ausgeführt wird, wenn bestimmte Baum-Elemente verarbeitet werden. Somit

---

[112] Abgeleitet vom englischen Begriff „*to parse*“.

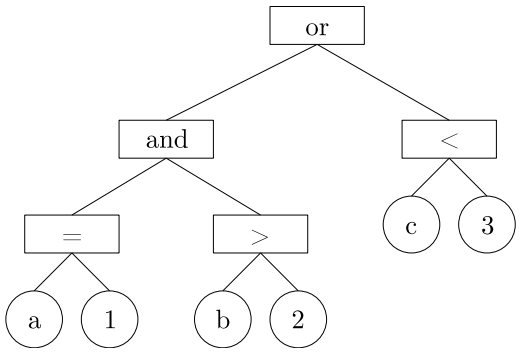
[113] ANTLR steht für *ANother Tool for Language Recognition*. Mehr Informationen sowie Downloadmöglichkeiten zu ANTLR findet man unter `http://www.antlr.org/` (Abruf: 2007-09-23).

[114] Eine genaue Beschreibung von LL(*)-Parsern findet man in Parr (2007), S. 262 ff.

[115] Bei der Entwicklung der ANTLR-Grammatik war das Open-Source-Werkzeug ANTLRWorks sehr hilfreich. ANTLRWorks bietet neben der Überprüfung der Syntax auch eine grafische Ausgabe des Parse-Baums und einen Debugger, mit dem man den Parsing-Vorgang genau untersuchen kann. ANTLRWorks kann unter `http://www.antlr.org/works/index.html` (Abruf: 2007-09-23) heruntergeladen werden. Ebenfalls hilfreich war das ANTLR-Plugin für Eclipse, das unter `http://www.javadude.com/tools/antlr3-eclipse/index.html` (Abruf: 2007-09-23) heruntergeladen werden kann.

[116] Auch: *Abstract Syntax Tree*. Kurz: AST. Ein AST repräsentiert die logischen Zusammenhänge eines Dokuments (in diesem Falle eine Abfrage) in einem Baum.

Abb. 6-26: Abstrakter Syntaxbaum für eine einfache Beispielabfrage



können für jeden Knoten des Baums Instanzen korrespondierender Java-Klassen erstellt werden. Der Baum wird so in eine Hierarchie von Java-Objekten überführt.

Für jeden Knotentypen existiert eine eigene Klasse. Konkret sind das die Klassen `AndExpression`, `Condition`, `Identifier`, `Literal`, `Negation`, `Operator` und `OrExpression`. Alle Klassen implementieren die Schnittstelle `Expression`. Diese Schnittstelle definiert die Methoden, die jede Klasse zur Auswertung oder Übersetzung eines Abfragebaums implementieren muss. Abbildung 6-27 stellt diese Schnittstelle dar.

Abb. 6-27: Schnittstelle für einen Ausdruck der Abfragesprache

<< interface >> **Expression**
+evaluate(identifiers:Map):Literal +toString():String +toSql(identifiers:Map):String +toXPath(identifiers:Map):String +extractConditions():List<Condition> +extractConditions(identifier:String):List<Condition> +prune(identifiers:Map):Expression

Die Methode `evaluate(identifiers)` wertet die gesamte Abfrage zu einem Wahrheitswert aus. Bei einer rekursiven Verarbeitung des Baumes werden die Operationen automatisch in der richtigen Reihenfolge (gemäß den Präzedenzen der Operatoren) ausgeführt. Die Methode `toString()` überführt einen Abfragebaum wieder in eine äquivalente String-Repräsentation. Die Methoden `toSql(identifiers)` und `toXPath(identifiers)` übersetzen den Abfragebaum in Ausdrücke der entsprechenden Sprachen.

Bis auf `toString()` nehmen die oben beschriebenen Methoden eine assoziative Liste[117] namens `identifiers` entgegen, die die Variablennamen der Abfrage auf entsprechende Ausdrücke bzw. Werte der Zielrepräsentation abbildet. Nutzt man in der Abfragesprache z.B. eine Variable `title`, die in einer SQL-Datenbank auf eine Spalte `entrytitle` der Tabelle `blogentries` abgebildet werden soll, so erstellt man in der assoziativen Liste einfach eine Zuordnung `title` ⇒ `blogentries.entrytitle`. Analog würde man für einen XPath-Ausdruck vorgehen. Will man den Ausdruck mittels `evaluate` auswerten, so setzt man die konkreten Werte der Variablen in die Liste.

Falls die genannten Auswertungs- bzw. Umwandlungsmethoden nicht ausreichen, so kann man recht einfach eigene Methoden definieren. Dazu fügt man die Methode zunächst in der Schnittstelle `Expression` hinzu und setzt sie dann für alle implementierenden Klassen um. Die Umsetzung ist in verhältnismäßig wenigen Quelltext-Zeilen möglich.

Zusätzlich zu diesen Methoden existieren weitere Hilfestellungen für die Verarbeitung der Abfragen. So extrahiert `extractConditions()` alle Bedingungen aus dem Abfragebaum. `extractConditions(identifiers)` extrahiert alle Bedingungen, die den angegebenen Bezeichner enthalten. Die Methode `prune(identifiers)` wertet alle Teile des Baums zu `false` aus, die einen Bezeichner benötigen, der nicht in `identifiers` definiert ist. Das ist nützlich, um Abfrageteile aus dem Baum zu entfernen, die eine Quelle nicht verwerten kann.

6.2.2 Extraktion durch die Quell-Komponenten

Im Rahmen des in dieser Arbeit zu entwickelnden Prototypen wurden Komponenten für folgende Quellen[118] implementiert: AFP Multimedia, dpa-Webline, beliebige News-Feeds, sowie Aktienkurse.

Die Komponenten wurden alle entsprechend dem Muster aus Abschnitt 5.4.2 entwickelt, d.h. sie bestehen neben der Klasse für die Quelle aus einem `CacheWrapper` und einem `SourceWrapper`.

Für wiederholt auftretende Aufgaben wurden Hilfsklassen angelegt. So wurde eine Klasse zum Herunterladen neuer Dateien von einem FTP-Server entwickelt. Benutzt wird sie für Dateien der AFP und der dpa, die per FTP ausgeliefert werden.[119] Sie

[117] Eine assiziative Liste (auch: *Hash-Map*) speichert mehrere (Schlüssel, Wert)-Paare. Man kann zu einem Schlüssel einen Wert speichern, und ihn über diesen Schlüssel abrufen.

[118] Für eine genauere Beschreibung dieser Quellen siehe Unterkapitel 2.4.

[119] Zu Testzwecken wurde ein eigener FTP-Server aufgesetzt, auf den die Beispieldaten der Anbieter hochgeladen wurden.

basiert auf der FTP-Client-Bibliothek edtFTPj/Free[120] und durchsucht rekursiv ein angegebenes Verzeichnis auf einem FTP-Server. Es werden nur die Dateien heruntergeladen, die seit dem letzten Abruf neu hinzugekommen sind, um die mehrfache Verarbeitung der gleichen Dateien zu vermeiden. Ebenfalls wurde eine Wrapper-Klasse um die Jakarta Commons HttpClient-Bibliothek[121] geschrieben, der einen sehr einfachen Zugriff auf HTTP-Ressourcen ermöglicht.

Der Verarbeitungsablauf für AFP Multimedia, dpa-Webline sowie für die Feeds ist recht ähnlich: Die Quellen werden zu regelmäßigen Zeitpunkten[122] nach neuen Artikeln abgefragt. Falls neue Artikel eingetroffen sind, so werden sie mittels XSLT-Stylesheets[123,124] in das Atom/NX-Format übertragen und (samt Fotos) in einer Datenbank abgelegt. Werden die Daten über eine Abfrage abgerufen, so übersetzt der `CacheWrapper` die Abfrage in eine Abfrage an die Datenbank und liefert die Ergebnisse zurück.

Die Feed-Komponente kann beliebige Feeds herunterladen. Zum Verarbeiten der Feeds wird die Bibliothek ROME[125] genutzt. ROME kann praktisch alle üblichen Feed-Formate[126] lesen und überführt sie in leicht zu verarbeitende Java-Objekte. Die Ergebnisse werden normalisiert, evtl. ergänzt und in das Atom/NX-Format übertragen.

Für den Nachrichtenticker von Heise Online[127] wurde zusätzlich eine Funktion implementiert, um für jede Meldung den kompletten Inhalt inklusive Abbildungen von der verlinkten Artikel-Web-Seite zu extrahieren und mit abzuspeichern. Dazu wird ein recht einfacher, aber robuster und für andere Quellen adaptierbarer Ansatz verfolgt: Die Quelldaten werden gesäubert und in einen Baum überführt. Aus dem Baum

---

[120] Die Bibliothek edtFTPj/Free kann unter `http://www.enterprisedt.com/downloads/ftp.html` (Abruf: 2007-09-23) heruntergeladen werden.

[121] Die Jakarta Commons HttpClient-Bibliothek kann unter `http://jakarta.apache.org/httpcomponents/httpclient-3.x/downloads.html` (Abruf: 2007-09-23) heruntergeladen werden.

[122] Die Abrufintervalle sind pro Quelle frei konfigurierbar.

[123] Für die Spezifikation zu XSLT 2.0 siehe World Wide Web Consortium (W3C) (2007f).

[124] Für die Ausführung der XSLT-Stylesheets wird die Open-Source-Bibliothek Saxon-B benutzt. Saxon-B kann unter `http://saxon.sourceforge.net/` (Abruf: 2007-09-23) heruntergeladen werden.

[125] Weitere Informationen zu ROME sowie eine Downloadmöglichkeit findet man unter `https://rome.dev.java.net/` (Abruf: 2007-09-23).

[126] ROME kann RSS 0.90, RSS 0.91 Netscape, RSS 0.91 Userland, RSS 0.92, RSS 0.93, RSS 0.94, RSS 1.0, RSS 2.0, Atom 0.3, sowie Atom 1.0 verarbeiten.

[127] Der Heise-Online-Nachrichtenticker ist unter `http://www.heise.de/newsticker/` (Abruf: 2007-09-23) zu erreichen.

werden dann die relevanten Elemente extrahiert. Diese Grundsätzliche Vorgehensweise wird auch oft in der Literatur zur Extraktion aus semistrukturierten Daten verfolgt. Viele Systeme definieren dafür eigene Extraktionssprachen.[128] Da sich die vom *World Wide Web Consortium* (W3C) entwickelten Standardsprachen XPath[129] und XSLT[130] zur Extraktion und Manipulation von XML-Daten durchgesetzt haben, gewinnen sie auch an Popularität für die Extraktion von Web-Inhalten.[131]

Konkret geschieht die Säuberung der heruntergeladenen Seite mit JTidy[132], das die Seite in gültiges XHTML 1.0 Strict überträgt. Die Extraktion der relevanten Daten erfolgt durch ein vergleichweise einfaches XSLT-Stylesheet, das als Ergebnis einen Atom/NX-Entry liefert. Die XPath-Ausdrücke zur Selektion relevanter Ausschnitte aus dem HTML-Baum kann man z.B. mit dem Firefox-Add-On Firebug[133] für eine geöffnete Web-Seite bestimmen.[134]

Die Komponente zum Abruf von Aktienkursen aus dem Dienst Yahoo Finance verfolgt einen anderen Ansatz. Da es sich um eine recht dynamische Quelle handelt, ist eine regelmäßige Replizierung aller Daten nicht sinnvoll. Stattdessen werden die Daten erst vom Server heruntergeladen, wenn der Benutzer eine entsprechende Abfrage absetzt. Die Quelle liefert das Ergebnis in Form einer CSV-Datei, die in eine XML-Struktur überführt und an den Benutzer weitergereicht wird. Das transformierte Ergebnis wird für einen bestimmten Zeitraum zwischengespeichert, um häufige Abfragen schneller beantworten zu können.

Abschließend sei angemerkt, dass die Quell-Komponenten dynamisch geladen werden. D.h. für eine neue Quell-Komponente muss der Quelltext des Kernsystems nicht modifiziert werden. Die Komponente muss bloß die `Source`-Schnittstelle implemen-

---

[128] Das in Liu u. a. (2000), S. 614 ff., vorgestelle System XWRAP generiert Abfragen in einem eigenen XML-Dialekt. Einen ähnlichen Ansatz verfolgt die WysiWyg Web Wrapper Factory (W4F). W4F wird in Sahuguet u. Azavant (2001) vorgestellt und nutzt die auf S. 290 ff. beschriebene eigene Extraktionssprache HTML Extraction Language.

[129] Die Spezifikation zu XPath 2.0 findet man unter World Wide Web Consortium (W3C) (2007d).

[130] Die Spezifikation zu XSLT 2.0 findet man unter World Wide Web Consortium (W3C) (2007f).

[131] Das am IBM Almaden Research Center entwickelte Web-Extraktions-System ANDES nutzt Tidy, XHTML, XPath und XSLT. Eine Beschreibung von ANDES findet man in Myllymaki (2001), S. 285 ff.

[132] JTidy ist eine Java-Umsetzung des bekannten Werkzeugs HTML-Tidy. JTidy kann unter `http://jtidy.sourceforge.net/` (Abruf: 2007-09-23) heruntergeladen werden. Für diese Arbeit wurde die aktuelle Version aus dem Subversion-Repository heruntergeladen und kompiliert.

[133] Firebug kann man unter `https://addons.mozilla.org/de/firefox/addon/1843` (Abruf: 2007-09-23) herunterladen.

[134] Dazu untersucht man die Seite mit Firebug, klickt auf den Text-Bereich auf der Seite und wählt die Option *Copy XPath* aus dem Kontextmenü im ausgewählten Knoten.

tieren, im Java-Classpath liegen und in der Konfiguration mit ihrem Klassennamen eingetragen werden. Sie wird dann während der Laufzeit durch den `Pool` instanziert.

6.2.3 Speicherung der Daten

Zur Speicherung der Daten wird eine XML-Datenbank genutzt. Die Nutzung einer XML-Datenbank[135] liegt nahe, da sie ohnehin schon im XML-Format vorliegen. Man kann über XQuery praktisch jedes Element der Daten in die Abfrage einbeziehen, ohne zuvor ein explizites Datenschema definieren zu müssen. Zum einfacheren Zugriff auf die XML-Datenbank wurde eine entsprechende Hilfsklasse geschrieben.

Die Speicherung in einer XML-Datenbank ist allerdings keineswegs verpflichtend. Der `CacheWrapper` entscheidet autonom, wie (und ob überhaupt) die Daten gespeichert werden sollen. Die Speicherung in einer relationalen Datenbank könnte möglicherweise performanter sein[136], erfordert jedoch die Formulierung eines expliziten Schemas, das für die abzufragenden Felder separate Spalten definieren muss.

Die Binärdaten (z.B. Fotos) werden ebenfalls als XML-Serialisierung[137] in der XML-Datenbank abgelegt. Die Speicherung in der XML-Datenbank hat den Vorteil, dass keine separate Datenbank-Abfrage für die Binärdaten nötig ist. Sie hat allerdings den Nachteil, dass die Datenbank möglicherweise sehr umfangreich wird. Dies müsste im Produktiveinsatz des Systems beobachtet werden und ggf. müssen die Binärdaten extern gespeichert werden.

6.2.4 Systemschnittstellen

Wie schon in Unterkapitel 5.7 beschrieben wurde, werden drei Server-Schnittstellen zum Abruf der Daten bereitgestellt.

Die REST-Schnittstelle wurde mit dem im JDK enthaltenen HTTP-Server[138] realisiert. Dazu wurde ein einfacher Handler geschrieben, der auf GET-Anfragen unterhalb des Pfades `/query/` antwortet. Die Abfrage muss einfach an diesen Pfad angehangen werden.

---

[135] Konkret wurde die Open-Source XML-Datenbank eXist eingesetzt. Weitere Informationen sowie eine Downloadmöglichkeit findet man unter `http://exist.sourceforge.net/` (Abruf: 2007-09-23).

[136] Wobei die eingesetzte XML-Datenbank eXist auch Abfragen über einen Bestand aus mehreren Tausend Artikel in wenigen Millisekunden beantworten kann.

[137] Die XML-Serialisierung wurde in Abschnitt 5.5.4 beschrieben.

[138] Der HTTP-Server befindet sich im Paket `com.sun.net.httpserver`.

Der RMI-Server basiert auf der Java-RMI-API[139] und startet eine integrierte RMI-Registry[140], falls keine Registry gefunden werden konnte.

Die SOAP-Schnittstelle wurde über die Java 6 Web-Services-Annotationen[141] realisiert. Sie bieten eine einfache Möglichkeit, Klassen und Methoden als Web-Service zu deklarieren. Die wesentlichen Annotationen sind `@WebService` und `@WebMethod`. Sie werden einfach vor die Klasse bzw. die Methode gesetzt. Die nötigen Web-Service-Wrapper werden dann automatisch[142] generiert.

6.2.5 Client-GUI

Obwohl sie nicht Teil der Anforderungen ist, wurde eine einfache Client-GUI entwickelt. Sie nutzt die Client-Bibliothek und kann entsprechend Abfragen über alle drei Schnittstellen absetzen. Die Abfragen werden während der Eingabe validiert und die Ergebnisse in Text-Form angezeigt. Falls das Ergebnis der Abfrage ein Foto ist, so wird es geladen und als Bild angezeigt. Abbildung 6-28 (S. 92) zeigt einen Screenshot der Client-GUI.

Die Client-GUI nutzt SWT[143] als GUI-Toolkit und ist somit weitgehend plattformunabhängig.

6.2.6 Hilfsklassen

Neben den schon erwähnten Hilfsklassen zum Herunterladen von Dateien von FTP- und HTTP-Quellen, sowie zum Umwandeln von HTML- in XHTML-Dateien, existieren noch weitere Hilfsklassen, die die Erstellung von weiteren Quell-Komponenten erleichtern.

---

[139] Eine sehr gute Einführung in Java RMI bietet das Tutorial, das unter `http://java.sun.com/docs/books/tutorial/rmi/index.html` (Abruf: 2007-09-23) zu finden ist.

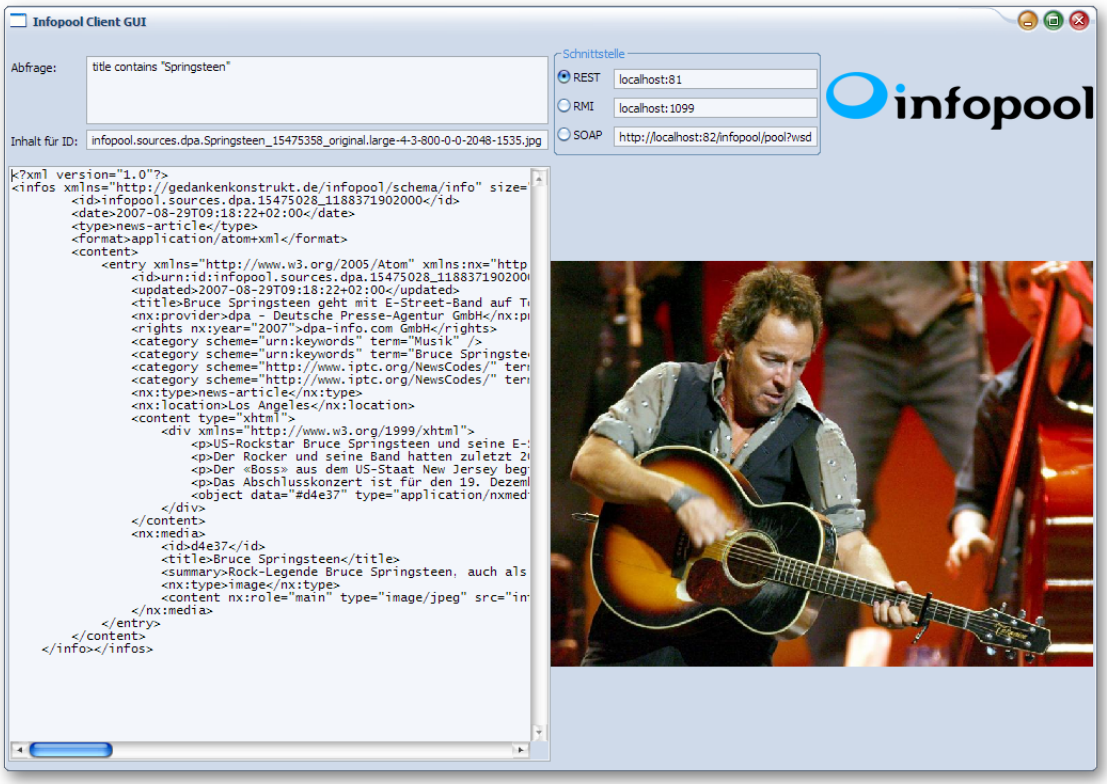
[140] Leider hat sich die integrierte Registry nur als bedingt zuverlässig erwiesen. Falls man tatsächlich RMI einsetzen möchte, sollte man möglicherweise eine dedizierte RMI-Registry starten.

[141] Die Spezifikation zur *Java API for XML-Based Web Services (JAX-WS) 2.0* findet man unter `http://jcp.org/aboutJava/communityprocess/final/jsr224/index.html` (Abruf: 2007-09-23). Für die Annotationen siehe S. 77ff im Spezifikationsdokument.

[142] Tatsächlich musste in der konkreten Anwendung ein Generator-Tool manuell ausgeführt werden, um die Exception-Klasse in die Generierung mit einzubeziehen. Der nötige Aufruf ist im Quelltext dokumentiert.

[143] Mehr Informationen zu SWT (*Standard Widget Toolkit*) sowie die Quell- und Binärpakete findet man unter `http://www.eclipse.org/swt/` (Abruf: 2007-09-23).

Abb. 6-28: Grafischer Client zum Stellen von Abfragen an das Integrationssystem



So wurden Hilfsklassen zum Lesen und Speichern von Konfigurationsdateien, zum Konvertieren zwischen Datumsformaten, zum Parsen[144] und Transformieren von XML-Daten und zum Zugriff auf XML-Datenbanken über die XML:DB-Schnittstelle[145] entwickelt. Diese Hilfsklassen nutzen wiederum andere Bibliotheken, bieten aber einfache Methoden für häufig benötigte Funktionen an und vermeiden somit redundanten Quelltext.

[144] Der Zugriff auf Elemente der XML-Dokumente wurde über das Prinzip des Document Object Models (DOM) realisiert. Das Document Object Model (DOM) überführt den XML-Baum in eine Hierarchie aus Objekten. Der Baum kann über die Methoden der Objekte verarbeitet werden. Als Bibliothek wurde dazu die Bibliothek XOM benutzt. XOM kann unter `http://www.cafeconleche.org/XOM/` (Abruf: 2007-09-23) heruntergeladen werden.

[145] Informationen zur XML:DB-Schnittstelle findet man unter `http://xmldb-org.sourceforge.net/` (Abruf: 2007-09-23).

6.3 Leistungsbewertung

6.3.1 Erfüllung der Anforderungen

Das wesentliche Leistungskriterium stellt sicherlich die Erfüllung der gesetzten Anforderungen dar. Eine Übersicht der Anforderungen[146] und eine kurze Beschreibung der Erfüllung mit Verweisen auf die entsprechenden Textstellen dieser Arbeit finden sich in Tabelle 6-4 (S. 95) und Tabelle 6-5 (S. 96).

Bis auf Anforderung 3.2 (S. 30) wurden alle mit **muss** priorisierten Anforderungen erfüllt. Die nicht-erfüllte Anforderung verlangt alternative Abrufschnittstellen für Quell-Komponenten, die nicht über die in Anforderung 3.1 (S. 30) geforderte Schnittstelle abfragbar sind. Bei der Implementierung hat sich gezeigt, dass alle implementierten Quell-Komponenten über die generische Abfrageschnittstelle ansprechbar sind. Eine alternative Schnittstelle war nicht nötig und wurde entsprechend nicht implementiert. Es wäre sogar denkbar, andere Abfragesprachen in eine Variable der generischen Sprache einzubetten: `customquery="Spezifische Abfrage"`. Die Quelle würde den Wert der Variablen extrahieren und die spezifische Abfrage ausführen. Alternativ könnte man die `Source`-Schnittstelle um weitere Methoden zur Abfrage erweitern.

Zusätzlich wurden alle mit **soll** priorisierten Anforderungen erfüllt. Die **kann**-Anforderung 1.3 (S. 28) wurde nicht direkt realisiert, da sie für die implementierten Quell-Komponenten nicht benötigt wurde. Indirekt wurde sie jedoch in den `CacheWrapper`n implementiert, die über Abfragesprachen auf eine Datenbank als Zwischenspeicher zugreifen. Analog würde man in einem `SourceWrapper` vorgehen. Wie in Abschnitt 6.2.1 gezeigt wurde, sind Übersetzungen der Abfragen nach SQL und XPath schon implementiert. Weitere Übersetzungen sind leicht zu realisieren.

6.3.2 Geschwindigkeit

Auch wenn eine hohe Geschwindigkeit nicht das primäre Ziel eines Prototypen ist, so soll sie hier dennoch betrachtet werden.[147]

Das Laden, Transformieren und Speichern neuer Informationseinheiten geschieht relativ schnell. Die ca. 1800 Beispieldateien des Dienstes dpa-Infoline konnten in rund 30 Sekunden von einem lokalen FTP-Server geladen, gefiltert und transformiert werden. Nach der Filterung[148] wurden 450 Dateien in die Datenbank geschrieben, was

[146] Eine Ausformulierung der Anforderungen findet man in Kapitel 3.

[147] Die Leistungstests wurden auf einem Desktop-PC mit einem Athlon-64 3000+ als CPU und 2GB RAM unter Windows XP SP2 durchgeführt.

[148] Es wird nur das Foto mit der höchsten Auflösung gespeichert.

Tab. 6-4: Übersicht der Anforderungen und deren Erfüllung

Anforderung	Kurzbeschreibung	Priorität	Erfüllung
1 (S. 26)	Abruf aus heterogenen Quellen	Muss	Für mehrere Quellen implementiert (siehe Abschnitt 6.2.2).
1.1 (S. 27)	Abruf über untersch. Protokolle	Muss	Nutzung von FTP und HTTP (siehe Abschnitt 6.2.2)
1.2 (S. 27)	Abruf über untersch. Schnittstellen	Soll	Implementiert für HTTP-Get und FTP-Get (siehe Abschnitt 6.2.2)
1.3 (S. 28)	Abruf über Abfragesprachen	Kann	Für implementierte Quellen nicht nötig, jedoch einfach über Bibliotheken realisierbar.
1.4 (S. 28)	Abruf über Funktionsaufrufe	Muss	Implementiert für Yahoo Finance
1.5 (S. 28)	Periodischer Abruf	Muss	Implementiert für HTTP und FTP (siehe Abschnitt 6.2.2).
1.6 (S. 28)	Dynamik der Quellen	Muss	Durch hybride Integrationsarchitektur berücksichtigt (siehe Abschnitt 4.1.3).
1.7 (S. 28)	Mangelnde Verfügbarkeit	Soll	Durch Cache berücksichtigt (siehe Abschnitt 4.1.3).
1.8 (S. 29)	Hohe Antwortzeiten	Soll	Durch Cache berücksichtigt (siehe Abschnitt 4.1.3).
1.9 (S. 29)	Umfangreiche Datenbestände	Muss	Durch hybride Integrationsarchitektur berücksichtigt (siehe Abschnitt 4.1.3).
2 (S. 26)	Verarbeitung heterogener Austauschformate	Muss	Implementiert für NewsML, NITF, RSS, Atom, (X)HTML, beliebige Binärdaten (siehe Abschnitt 6.2.2).
2.1 (S. 29)	Strukturierte Daten	Muss	Implementiert für CSV (Yahoo Finance)
2.2 (S. 29)	Semi-strukturierte Daten	Soll	Implementiert für NewsML, NITF, RSS, Atom und (X)HTML (siehe Abschnitt 6.2.2).
2.3 (S. 29)	Unstrukturierte Daten	Muss	Implementiert für beliebige Binärdaten (siehe Abschnitt 6.2.2).

Tab. 6-5: Fortsetzung: Übersicht der Anforderungen und deren Erfüllung

Anforderung	Kurzbeschreibung	Priorität	Erfüllung
3 (S. 26)	Entwicklung einer einfachen Abfrageschnittstelle	Muss	Definiert in Abschnitt 4.3.4. Für Implementierung siehe Abschnitt 6.2.1.
3.1 (S. 30)	Generische Abfrageschnittstelle	Soll	Siehe übergeordnete Anforderung.
3.2 (S. 30)	Alternative Abfrageschnittstelle	Muss	Für eingebundene Quellen nicht erforderlich.
4 (S. 26)	Vereinheitlichung der Austauschformate	Muss	Transformation von NewsML, NITF, RSS, Atom, (X)HTML nach Atom/NX+XHTML (siehe Unterkapitel 4.4, Unterkapitel 5.6 und Abschnitt 6.2.2).
4.1 (S. 30)	Minimierung syntaktischer und struktureller Heterogenität	Muss	Siehe übergeordnete Anforderung.
4.2 (S. 31)	Erhaltung der Semantik	Soll	Nur relevante Semantik bleibt erhalten (siehe Abschnitt 5.6.2).
4.3 (S. 31)	Erweiterung für neue Austauschformate	Muss	Einfach durch neue Klassenmethoden möglich (siehe Abschnitt 5.6.6).
5 (S. 27)	Option zur Einbindung zusätzlicher Quell-Komponenten	Muss	Lediglich Implementierung einer einfachen Schnittstelle nötig (siehe Abschnitt 5.4.2).
5.1 (S. 31)	Schnittstellen zur Einbindung zusiehe Quell-Komponenten	Muss	Für Schnittstellendefinition siehe Abschnitt 5.4.2.
5.2 (S. 31)	Quell-Komponenten ohne Modif. am Kernsystem einbindbar	Soll	Komponenten werden dynamisch geladen (siehe Ende Abschnitt 6.2.2).
6 (S. 27)	Quellen-Verwaltung	Muss	Siehe Unteranforderungen.
6.1 (S. 32)	Installierbarkeit neuer Quell-Komponenten ohne nötige Quelltextmodif.	Soll	Komponenten werden dynamisch geladen (siehe Ende Abschnitt 6.2.2).
6.2 (S. 32)	Konfigurierbarkeit ohne nötige Quelltextmodif.	Soll	Konfiguration über Datei `infopool.properties` möglich.

knapp 24 Sekunden dauerte. Zur Verarbeitung und Speicherung von 450 Informationseinheiten (nach Filterung) brauchte das System insgesamt 54 Sekunden. Das entspricht über 8 Dateien pro Sekunde.

Die ca. 780 Beispieldateien des AFP-Multimedia-Dienstes konnten in ca. 38 Sekunden heruntergeladen und transformiert werden. Dabei wurden bis auf zwei fehlerhafte alle Dateien gespeichert. Die Speicherung dauerte ungefähr 35 Sekunden. In der Summe entspricht das über 10 Dateien pro Sekunde.

Für den Abruf der Daten wurden folgende Zahlen ermittelt: Die Abfrage `title contains "Merkel"` über die 1200 gespeicherten Informationseinheiten lieferte acht Artikel und benötigte dafür im Durchschnitt ca. 0,10 Sekunden. Die Abfrage `type = "news-article"` lieferte 478 Informationseinheiten und wurde in ca. 3 Sekunden beantwortet. Der Abruf einer Informationseinheit über ihre ID benötigte durchschnittlich 0,05 Sekunden.

Beim Abruf von Informationen aus Web-Quellen (wie Feeds oder den Aktienkursen von Yahoo Finance) ergab sich, dass die Übertragung von den Web-Servern den Flaschenhals darstellte.

Die oben beschriebenen einfachen Leistungstests können eine genaue Untersuchung unter Produktiv-Bedingungen nicht ersetzen. Der erste Eindruck der Leistung ist jedoch sehr positiv, obwohl im Rahmen des Prototypen kaum Optimierungen vorgenommen wurden. Vorschläge zu möglichen Optimierungen werden in Abschnitt 7.2.2 gegeben.

6.3.3 Unit-Tests

Klassen, die eine zentrale Stellung im Quelltext einnehmen, wurden mit Unit-Tests auf die erwartete Funktionalität hin überprüft. Als Test-Framework wurde JUnit 4[149] eingesetzt.

Die Tests decken im Wesentlichen die Informationsklassen `CommonInfo`, `InfosCollection`, den Abfrage-Parser, die Auswertung und Übersetzung der Abfragen sowie die Datumskonvertierungen. Insgesamt existieren 300 Testfälle, die die korrekte Funktionalität dieser Kernkomponenten sichern sollen. Natürlich kann man nur die Anwesenheit, nicht aber die Abwesenheit von Fehlern testen. Aber allein der Test auf Anwesenheit von Fehlern hat während der Entwicklung zur Entdeckung vieler Probleme geführt.

[149] Informationen sowie Downloadmöglichkeiten zu JUnit findet man unter `http://www.junit.org/` (Abruf: 2007-09-24).

6.4 Dokumentation

Die entwickelte Software wurde umfangreich dokumentiert. Zusätzlich zum schriftlichen Teil dieser Arbeit existiert eine Quelltext-Dokumentation, ein Benutzerhandbuch, eine Anleitung zur Erstellung einer einfachen Quell-Komponente sowie eine Übersicht der verwendeten Bibliotheken und Werkzeuge.

Die **Quelltext-Dokumentation** wurde per Javadoc[150] aus den Quelltexten generiert. Es wurde Wert darauf gelegt, den Quelltext ausreichend zu dokumentieren. Die generierte Dokumentation befindet sich auf der dieser Arbeit beiliegenden CD. Sie ist für jedes Projekt im Unterverzeichnis `doc` zu finden.

Das **Benutzerhandbuch** beschreibt die Installation und Konfiguration der Software. Ebenfalls wird beschrieben, wie man mit einem Web-Browser oder mit der Client-GUI Abfragen an das System senden kann. Sie liegt im Wurzelverzeichnis der CD. Die Anleitung zur Erstellung einer einfachen Quell-Komponente ist ebenfalls im Wurzelverzeichnis auf der CD zu finden.

Eine **Übersicht der verwendeten Bibliotheken** befindet sich im Anhang in Abschnitt Unterkapitel 9.4.

[150] Informationen zu Javadoc und Downloads findet man unter `http://java.sun.com/j2se/javadoc/` (Abruf: 2007-09-24).

7 Schlussfolgerungen

7.1 Zusammenfassung

Ziel der Arbeit

Diese Arbeit hatte sich zum Ziel gesetzt, eine Lösung auf die Probleme zu finden, die sich bei der Verarbeitung heterogener Inhalte in individualisierbaren Informationsdiensten ergeben. Insbesondere sollten der entstehende Aufwand bei der Verarbeitung der Daten minimiert, sowie technische Probleme so weit wie möglich gekapselt werden. Es sollte ein prototypisches Softwaresystem entwickelt werden, das die vorgeschlagenen Lösungen umsetzt.

Grundlagen

Nachdem in Kapitel 2 u.a. eine Konkretisierung der Problemstellung[151] gegeben wurde, folgte eine Übersicht über geeignete Informationsquellen[152]. Die Quellen wurden klassifiziert, auf ihre technischen Eigenschaften hin untersucht und hinsichtlich der Eignung zur Verwendung in einem IID beurteilt. Konkret wurden Komponenten zur Integration von Nachrichten aus den Diensten AFP Multimedia, dpa-Webline und Heise-Online implementiert. Zusätzlich wurde eine Komponente entwickelt, die für eine persönliche Assistenz den Abruf von Aktienkursen aus dem Dienst Yahoo Finance ermöglicht.

Anforderungen

Die Erwartungen an das zu entwickelnde System wurden in Kapitel 3 in Form strukturierter Anforderungslisten präzisiert. Eine wesentliche Kernanforderungen beinhaltete den Zugriff auf heterogene Quellen unter Beachtung unterschiedlicher Zugriffsschnittstellen und möglicher Probleme durch das unterschiedliche Verhalten (Dynamik, Verfügbarkeit, Antwortzeiten, Umfang) der Quellen. Ebenso musste die Verarbeitung von Daten mit unterschiedlichem Strukturierungsgrad und in heterogenen Austauschformaten möglich sein. Die Entwicklung einer möglichst einfachen, aber hinreichenden Abfrageschnittstelle wurde ebenfalls gefordert. Des Weiteren sollten die Austauschformate zur Reduktion des Aufwands der Verarbeitung der Daten vereinheitlicht werden. Es sollten Möglichkeiten zur nachträglichen Einbindung zusätzlicher Informationsquellen durch Erweiterungskomponenten sowie zur möglichst einfachen Installation und Konfiguration der Komponenten geschaffen werden.

---

[151] Für die Konkretisierung der Problemstellung siehe Unterkapitel 2.3.

[152] Mögliche Informationsquellen werden in Unterkapitel 2.4. Eine tabellarische Übersicht konkreter Beispiele findet man im Anhang in Unterkapitel 9.1.

Lösungsansatz

In Kapitel 4 wurden die im Folgenden kurz beschriebenen Lösungen zur Erfüllung der Anforderungen vorgeschlagen. Als grundlegender Lösungsansatz wurden die Konzepte der Datenintegration untersucht.[153] Als wesentliche Integrationsansätze wurden die virtuelle und die materialisierte Integration identifiziert und mit ihren Vor- und Nachteilen gegenübergestellt. Die Datenintegration wurde im Kontext individualisierbarer Informationsdienste kritisch durchleuchtet. Es wurde festgestellt, dass eine Adaption der Konzepte an die spezifische Problemstellung erforderlich ist. Das Ergebnis ist ein hybrider Integrationsansatz, der die virtuelle mit der materialisierten Integration vereint.

Aufbauend auf diesem grundlegenden Lösungsansatz wurden die Teilprobleme genauer untersucht. Das unterschiedliche Verhalten der Quellen kann durch einen gezielten Einsatz der hybriden Integration gut behandelt werden.[154] Die Komplexität durch die vielen unterschiedlichen Zugriffsmöglichkeiten auf die Quellen wird vollständig durch das System gekapselt und ist für den Benutzer des Systems transparent. Der Benutzer greift über eine einfache Abfragesprache auf das System zu.[155] Die zum Abruf der Daten nötigen Sprachelemente wurden identifiziert und formal in Form einer EBNF-Grammatik festgehalten.

Schließlich wurde der Aufwand, der durch die heterogenen Austauschformate der Informationen entsteht, genau analysiert.[156] Eine Vielzahl an Austauschformaten führt zu einer hohen Anzahl nötiger Transformationen, die durch den Einsatz von Zwischenformaten reduziert werden kann. Es wurde zusätzlich festgestellt, dass die Anzahl der Transformationen nicht allein entscheidend ist. Vielmehr muss auch die Komplexität möglicher Zwischenformate berücksichtigt werden. Die Schlussfolgerung war, dass eine Zusammenfassung mehrerer Austauschformate zu einem Zwischenformat genau dann sinnvoll ist, falls sich dadurch Synergien ergeben, also wenn die Ursprungsformate ähnlich sind.

Entwurf

Die identifizierten Lösungen wurden daraufhin in Kapitel 5 in einen entsprechenden Software-Entwurf übertragen. Wesentliche Entwurfsziele waren eine lose Kopplung der Komponenten und eine offene Architektur, die möglichst wenige Ein-

---

[153] Die Diskussion der Datenintegration als Lösungsansatz findet man in Unterkapitel 4.1.

[154] Der Umgang mit dem unterschiedlichen Verhalten der Quellen stellt den Kernaspekt von Unterkapitel 4.2 dar.

[155] Die Abfrageschnittstelle wurde in Unterkapitel 4.3 erarbeitet.

[156] Für die Analyse der Probleme durch heterogene Austauschformate sowie die gezogene Schlussfolgerung siehe Unterkapitel 4.4.

schränkungen für Erweiterungskomponenten enthält.[157] Zusätzlich sollten die Quell-Komponenten möglichst autonom sein. Sie müssen lediglich eine einfache Schnittstelle implementieren und haben davon abgesehen große Freiheiten bei der Erfüllung ihrer Aufgaben. Diese Autonomie ist nötig, da die Quell-Komponenten mit höchst heterogenen Informationsquellen interagieren sollen und jede Vorgabe den Einsatzbereich einschränken könnte.

Neben der Beschreibung wesentlicher Schnittstellen und Klassen des Systems wurde auch ein geeignetes Zwischenformat für Nachrichtenartikel vorgestellt.[158] Es basiert auf dem *Atom Syndication Format* und erweitert es um zusätzliche, für Nachrichtenartikel übliche Angaben wie z.B. Orte, Untertitel und den Anbieter der Informationen.

Implementierung

Schließlich wurde das System implementiert. In Kapitel 6 werden zunächst grundlegende Implementierungsentscheidungen begründet.[159] So wurde das System in Java entwickelt und macht zur Erreichung einer schnellen und wirtschaftlichen Entwicklung starken Gebrauch von Open-Source-Bibliotheken.

Daraufhin wurden die wesentlichen Details der Implementierung dargestellt.[160] Die Abfragesprache wird mithilfe des Parser Generators ANTLR in einen Baum aus Java-Objekten transformiert. Die Abfrageergebnisse werden je nach Schnittstelle in Form von Java-Objekten oder XML-Daten zurückgegeben. Zur Transformation von Daten, die in XML-Dialekten vorliegen, wurden XSLT-Stylesheets entwickelt. Zur Extraktion von Daten aus Web-Seiten wurde ein recht einfacher, aber leistungsfähiger Ansatz vorgestellt: Der Seiten-Quelltext wird mittels JTidy in gültiges XHTML 1.0 Strict transformiert. Mittels XSLT-Stylesheets können daraufhin die relevanten Teile der Seite extrahiert werden. Die nötigen XPath-Ausdrücke können grafisch mithilfe des Firefox-Add-Ons Firebug ermittelt werden. Demonstriert wurde diese Vorgehensweise am Heise-Online-Newsticker. Für einen Artikel können der Inhalt sowie alle Abbildungen extrahiert werden.

Die transformierten Daten werden in einer XML-Datenbank abgelegt und können über mehrere Systemschnittstellen abgerufen werden. Das Integrationssystem stellt

---

[157] Eine ausführliche Beschreibung der grundlegenden Architekturentscheidungen findet man in Unterkapitel 5.2.

[158] Für den Entwurf der Schnittstellen und Klassen sowie die vorgeschlagenen Zwischenformate siehe Unterkapitel 5.3 bis Unterkapitel 5.6.

[159] Die genaue Beschreibung der grundlegenden Implementierungsentscheidungen befindet sich in Unterkapitel 6.1.

[160] Alle wesentlichen Implementierungsdetails wurden in Unterkapitel 6.2 festgehalten.

dazu eine REST-, eine Java-RMI- und eine SOAP-Schnittstelle zur Verfügung. Ein einfacher grafischer Client nutzt diese Schnittstellen zur Demonstration des Zugriffs auf das System. Die REST-Schnittstelle kann zusätzlich mit einem einfachen Web-Browser ohne zusätzliche Software genutzt werden.

Abgeschlossen wurde das Implementierungskapitel durch eine Leistungsbewertung sowie Hinweise zur Dokumentation.[161]

7.2 Kritische Reflexion und Ausblick

7.2.1 Nicht erreichte Ziele

Wie in Abschnitt 6.3.1 genauer erläutert wurde, konnte eine Unteranforderung nicht erfüllt werden. Sie fordert neben der generischen Abfrageschnittstelle alternative Schnittstellen für Quell-Komponenten, für die die generische Schnittstelle ungeeignet ist. Da die generische Abfrageschnittstelle sich für alle entwickelten Komponenten als geeignet erwiesen hatte, waren alternative Schnittstellen nicht erforderlich. Im Produktiveinsatz wird sich zeigen, ob sich diese Anforderung möglicherweise als unnötig erweist.

7.2.2 Alternative Lösungsmöglichkeiten und Optimierungen

Extraktion aus Web-Seiten

Der in Abschnitt 6.2.2 vorgestellte einfache Ansatz zur Extraktion von Informationen erfordert eine manuelle Entwicklung von XSLT-Stylesheets und ist somit möglicherweise ungeeignet, um ihn auf eine große Anzahl sehr unterschiedlicher Web-Seiten anzuwenden. Außerdem kann die Extraktion fehlschlagen, wenn sich die Struktur der Web-Seite verändert. Auch wenn die Erstellung solcher Stylesheets durch die gute Werkzeugunterstützung deutlich vereinfacht wird, skaliert eine manuelle Definition möglicherweise schlecht. Hier wäre die Nutzung oder Entwicklung leistungsfähigerer Extraktionssysteme denkbar.[162]

Geschwindigkeit

Der in dieser Arbeit entstandene Quelltext ist noch nicht auf eine hohe Verarbeitungsgeschwindigkeit optimiert. Vielmehr soll er einfach und verständlich sein. Ein bekanntes Zitat von Donald Knuth liefert den Grund dafür:

---

[161] Die Leistungsbewertung findet man in Unterkapitel 6.3. Für die Hinweise zur Dokumentation siehe Unterkapitel 6.4.

[162] Beispiele für Extraktionssysteme sind XWRAP (siehe Liu u. a. (2000)), W4F (siehe Sahuguet u. Azavant (2001)) und ANDES (siehe Myllymaki (2001)). Adaptive Extraktionssysteme, also solche, die sich bei Veränderungen in Web-Seiten anpassen sowie weitgehend universell einsetzbar sein sollen, werden in Gregg u. Walczak (2006), S. 80 ff. definiert.

Premature optimization is the root of all evil.[163]

Es wurde allerdings darauf geachtet, unnötige Operationen zu vermeiden.[164] Aber es existieren noch weitere Ansatzpunkte für spätere Optimierungen. Sollte sich die XML-Datenbank unerwarteterweise als Flaschenhals erweisen, so sollte sie – wie in Abschnitt 6.2.3 beschrieben – mit vertretbarem Aufwand durch eine relationale Datenbank austauschbar sein. Ebenfalls könnten die derzeit in der Datenbank gespeicherten Binärdaten (z.B. Fotos) extern gespeichert werden. Schließlich könnte man die Deserialisierung der XML-Repräsentationen der Daten anstatt mit DOM über die schnellere und speicherschonendere SAX-API durchführen.

7.2.3 Erweiterung

Zusätzliche Quell-Komponenten

Wie in Unterkapitel 3.9 beschrieben wurde, implementiert der Prototyp nur einen Ausschnitt des gesamten Systems. Insbesondere wurde nur eine begrenzte Anzahl an Quell-Komponenten entwickelt. Die Möglichkeit der Integration zusätzlicher Komponenten stellt jedoch einen Kernaspekt des Systems dar. Sie wurde durchgängig von Anforderungen bis hin zur Implementierung bedacht. Neben dem Entwurf, der die nötigen Schnittstellen dokumentiert, befindet sich auf der dieser Arbeit beiliegenden CD eine Anleitung zur Entwicklung einer zusätzlichen Quell-Komponente. Für weitere Hinweise zur Dokumentation siehe Unterkapitel 6.4.

Zusätzliche Zwischenformate

Das entwickelte Zwischenformat Atom/NX[165] ist zwar recht flexibel[166] und erweiterbar, jedoch kann es nicht alle denkbaren Informationsarten abdecken. Möglicherweise muss das Schema für ähnliche, aber nicht identische Informationsarten erweitert werden. Evtl. müssen neue Formate genutzt oder entwickelt werden. Das Integrationssystem sieht die Verwendung beliebiger Austauschformate explizit vor, indem alle Daten durch ein generisches Containerformat, das in Abschnitt 5.5.1 beschrieben wurde, gekapselt werden. Innerhalb dieses Containers können sich Daten in beliebigen Austauschformaten befinden.

---

[163] Dieses Zitat findet man in Knuth (1974), S. 268.

[164] Bspw. wurde die Abfrageschnittstelle um eine Methode `getInfosXml(query:String):InfosXml` erweitert. Sie erlaubt es, durchgängig XML-Daten zu verarbeiten – falls alle Komponenten in der Kette von der Quelle bis zum Client XML-Daten nutzen – und vermeidet somit unnötige (De-)Serialisierungen zwischen Java- und XML-Darstellungen.

[165] Atom/NX wird in Abschnitt 5.6.2 beschrieben. Das Erweiterungsschema findet man im Anhang in Abschnitt 9.2.2.

[166] Beispiele für Erweiterungen dieses Formats finden sich in Abschnitt 5.6.4.

Sicherheit

Im Rahmen des Prototypen wurden Aspekte bzgl. der Sicherheit des Systems nicht berücksichtigt. Unter der Annahme, dass das System in einer kontrollierten Umgebung (privates Netzwerk bzw. Schutz durch eine Firewall) eingesetzt wird, sind Vorkehrungen für eine Authentifizierung oder die Verhinderung von Angriffen nicht nötig. Soll das Integrationssystem Teil eines verteilten Gesamtsystems sein, deren Komponenten über öffentliche Netzwerke kommunizieren, so sollten zusätzliche Sicherheitsvorkehrungen implementiert werden.

7.3 Fazit

In dieser Arbeit wurden Lösungen für einen einheitlichen Zugriff auf heterogene Inhalte vorgestellt. Die Eignung dieser Lösungen wurde anhand einer prototypischen Implementierung eines Datenintegrationssystems überprüft und weitgehend bestätigt.

Eine große Herausforderung stellte im Vorfeld der Arbeit die Ermittlung und Berücksichtigung der potenziellen Anforderungen externer Komponenten dar, die das Integrationssystem nutzen sollen. So wurde zunächst die Einbindung dieser Komponenten in den Integrationsprozess der Informationen erwägt. Durch die vielen Abhängigkeiten erreichte die Architektur, die solch einen Prozess ermöglicht, jedoch eine hohe Komplexität. Diese ursprüngliche Vorgehensweise erwies sich als ungeeignet und wurde nicht weiter verfolgt.

Als wesentlich einfacherer hat sich die Idee herausgestellt, die externen Komponenten nicht einzubinden, sondern sie als Dienst-Konsumenten zu betrachten, die über eine schmale Schnittstelle auf das System zugreifen. Diese Variante hielt letztlich Einzug in die Anforderungen, die entsprechenden Lösungsansätze und in das darauf aufbauende System. Es zeigte sich, dass nicht immer die umfassendste, sondern oft die einfachere Lösung besser geeignet ist.

Das entsprechend diesem Prinzip entwickelte System hat sich als grundsätzlich geeignete Lösung auf die Problemstellung erwiesen. Trotz allem muss es sich noch im Produktiveinsatz bewähren. Dort wird sich zeigen, welche konkreten Erweiterungen und Optimierungen erforderlich sind. Entsprechende Anforderungen wurden konsequent berücksichtigt und zukünftige Änderungen werden somit ermöglicht.

8 Literatur

Abiteboul 1997

Abiteboul, Serge: Querying Semi-Structured Data. In: **ICDT '97: Proceedings of the 6th International Conference on Database Theory**. London, UK : Springer-Verlag, Jan 1997. – ISBN 3540622225, 1–18

Blieberger u. a. 2001

Blieberger, Johann ; Klasek, Johann ; Redlein, Alexander: **Informatik**. Springer-Verlag KG, 2001. – ISBN 3211828605

Chaudhuri u. Dayal 1997

Chaudhuri, Surajit ; Dayal, Umeshwar: An Overview of Data Warehousing and OLAP Technology. In: **SIGMOD Rec.** 26 (1997), 03, Nr. 1, S. 65–74

Checkland u. Scholes 1999

Checkland, Peter ; Scholes, Jim: **Soft Systems Methodology in Action**. Wiley & Sons, 1999. – ISBN 0471986054

Domenig u. Dittrich 1999

Domenig, Ruxandra ; Dittrich, Klaus R.: An Overview And Classification of Mediated Query Systems. In: **SIGMOD Rec.** 28 (1999), Nr. 3, S. 63–72

Fielding 2000

Fielding, Roy: **Architectural Styles and the Design of Network-based Software Architectures**, University of California, Irvine, Diss., 2000. http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

Freed u. Borenstein 1996

Freed, N. ; Borenstein, N.: **Multipurpose Internet Mail Extensions Part Two: Media Types**. http://www.ietf.org/rfc/rfc2046.txt, Veröffentlichung: 1996-11, Abruf: 2007-09-21

Freeman u. a. 2004

Freeman, Elisabeth ; Freeman, Eric ; Bates, Bert ; Sierra, Kathy: **Head First Design Patterns**. O'Reilly, 2004. – ISBN 0596007124

Gamma u. a. 1995

Gamma, Erich ; Helm, Richard ; Johnson, Ralph ; Vlissides, John: **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995

Gregg u. Walczak 2006

Gregg, Dawn G. ; Walczak, Steven: Adaptive Web Information Extraction. In: **Communications of the ACM** 49 (2006), 05, Nr. 5, S. 78–84

Hanani u. a. 2001

Hanani, Uri ; Shapira, Bracha ; Shoval, Peretz: Information Filtering: Overview of Issues, Research and Systems. In: **User Modeling and User-Adapted Interaction** 11 (2001), Nr. 3, S. 203–259

Hull u. Zhou 1996

Hull, Richard ; Zhou, Gang: A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches. In: **SIGMOD '96: Proceedings of the 1996 ACM SIGMOD international conference on Management of data**. New York, NY, USA : ACM Press, 1996. – ISBN 0897917944, S. 481–492

International Press Telecommunications Council 2007a

International Press Telecommunications Council: **NewsML 1.2 Specification**. http://www.newsml.org/IPTC/NewsML/1.2/specification/NewsML_1.2-spec-functionalspec_8.html, Veröffentlichung: 2003-10-10, Abruf: 2007-05-11

International Press Telecommunications Council 2007b

International Press Telecommunications Council: **NITF: A Solution for Sharing News**. 2007

ISO/IEC 1993

ISO/IEC: **ISO/IEC 2382-1: Information technology – Vocabulary – Part 1: Fundamental terms**. http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip

ISO/IEC 1994

ISO/IEC: **ISO/IEC 7498-1: Open Systems Interconnection - Basic Reference Model**. http://standards.iso.org/ittf/PubliclyAvailableStandards/s020269_ISO_IEC_7498-1_1994(E).zip. – Geneva 2001

ISO/IEC 1996

ISO/IEC: **ISO/IEC 14977: Extended BNF**. http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip

Knuth 1974

KNUTH, Donald E.: Structured Programming with go to Statements. In: **ACM Comput. Surv.** 6 (1974), Nr. 4, S. 261–301

Kugler 2005

KUGLER, Sebastian: **Individuelle gedruckte Tageszeitung - Konzeption und Implementierung eines Software-Prototypen**, Universität zu Köln, Diplomarbeit, 2005

Lakshmanan u. a. 2001

LAKSHMANAN, Laks V. S. ; SADRI, Fereidoon ; SUBRAMANIAN, Subbu N.: SchemaSQL: An Extension to SQL for Multidatabase Interoperability. In: **ACM Transactions on Database Systems** 26 (2001), Nr. 4, S. 476–519

Leffler 2007

LEFFLER, Jonathan: **BNF Grammars for SQL-92, SQL-99 and SQL-2003**. http://savage.net.au/SQL/, Veröffentlichung: 2005-07-11, Abruf: 2007-07-10

Lenzerini 2002

LENZERINI, Maurizio: Data Integration: A Theoretical Perspective. In: **PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems**. New York, NY, USA : ACM Press, 2002. – ISBN 1581135076, S. 233–246

Leser u. Naumann 2006

LESER, Ulf ; NAUMANN, Felix: **Informationsintegration: Architekturen und Methoden zur Integration verteilter und heterogener Datenquellen.** 1. dpunkt.verlag, 2006. – ISBN 3898644006

Liu u. a. 2000

LIU, Ling ; PU, Calton ; HAN, Wei: XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In: **ICDE** 00 (2000), S. 611–621

Maes 1994

MAES, Pattie: Maes1994 - Agents That Reduce Work and Information Overload. In: **Communications of the ACM** 37 (1994), Nr. 7, S. 30–40

Manolescu u. a. 2001

MANOLESCU, Ioana ; FLORESCU, Daniela ; KOSSMANN, Donald: Answering XML Queries on Heterogeneous Data Sources. In: **VLDB '01: Proceedings of**

the 27th International Conference on Very Large Data Bases. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2001. – ISBN 1558608044, S. 241–250

McConnell 2004
McConnell, Steve: **Code Complete (2nd Ed.)**. Microsoft Press, 2004. – ISBN 0735619670

Mintert 2005
Mintert, Stefan: Implementierung von Webservices - REST vs. SOAP? In: **Wirtschaftsinformatik** 1 (2005), S. 63–65

Myllymaki 2001
Myllymaki, Jussi: Effective Web Data Extraction with Standard XML Technologies. In: **Proceedings of the 10th international conference on World Wide Web**, ACM Press, New York, NY, USA, 2001, S. 635–644

Netscape 1999
Netscape: **RSS 0.90 Specification**. http://www.purplepages.ie/RSS/netscape/rss0.90.html, Veröffentlichung: 1999, Abruf: 2007-08-15

Nottingham u. Sayre 2005
Nottingham, Mark ; Sayre, Robert: **The Atom Syndication Format**. http://www.ietf.org/rfc/rfc4287, Veröffentlichung: 2005-12, Abruf: 2007-02-23

Parr 2007
Parr, Terence: **The Definitive ANTLR Reference: Building Domain-Specific Languages**. Pragmatic Bookshelf, 2007. – ISBN 0978739256

Pomberger u. a. 1991
Pomberger, Gustav ; Bischofberger, Walter R. ; Kolb, Dieter ; Pree, Wolfgang ; Schlemm, Holger: Prototyping-Oriented Software Development - Concepts and Tools. In: **Structured Programming** 12 (1991), Nr. 1, S. 43–60

Popa u. a. 2002
Popa, Lucian ; Velegrakis, Yannis ; Miller, Renee J. ; Hernandez, Mauricio A. ; Fagin, Ronald: Translating Web Data. In: **Proceedings of VLDB 2002, Hong Kong SAR, China**, 2002, 598–609

Rechenberg 2003
Rechenberg, Peter: Zum Informationsbegriff der Informationstheorie. In: **Informatik Spektrum** 26 (2003), Nr. 5, S. 317–326

Rossi u. a. 2001

Rossi, Gustavo ; Schwabe, Daniel ; Guimar, Robson: Designing Personalized Web Applications. In: **WWW '01: Proceedings of the 10th international conference on World Wide Web**. New York, NY, USA : ACM Press, 2001. – ISBN 1581133480, S. 275–284

Roth u. Schwarz 1997

Roth, Mary T. ; Schwarz, Peter M.: Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources. In: **VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases**. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997. – ISBN 1558604707, S. 266–275

RSS Advisory Board 2006

RSS Advisory Board: **RSS 2.0 Specification**. http://www.rssboard.org/rss-specification, Veröffentlichung: 2006-08-12, Abruf: 2007-02-23

RSS-DEV Working Group 2000

RSS-DEV Working Group: **RDF Site Summary (RSS) 1.0**. http://web.resource.org/rss/1.0/, Veröffentlichung: 2000-12-06, Abruf: 2007-08-15

Sahuguet u. Azavant 2001

Sahuguet, Arnaud ; Azavant, Fabien: Building Intelligent Web Applications Using Lightweight Wrappers. In: **Data Knowledge Engineering** 36 (2001), Nr. 3, S. 283–316

Schoder u. Sick 2003

Schoder, Detlef ; Sick, Stefan: **System und Verfahren zur Herstellung eines kundenindividuellen Druckerzeugnisses** (Juni 2003). Internationale Veröffentlichungsnummer: WO 03/052648 A2

Scholz 2007

Scholz, Jens: **Jean-Remy von Matt ist beleidigt (Klowände des Internets)**. http://www.jensscholz.com/2006/01/jean-remy-von-matt-ist-beleidigt.htm, Veröffentlichung: 2006-01-19, Abruf: 2007-09-25

Stahlknecht u. Hasenkamp 2004

Stahlknecht, Peter ; Hasenkamp, Ulrich: **Einführung in die Wirtschaftsinformatik**. 11. Springer, Berlin, 2004

Tanenbaum 2002

Tanenbaum, Andrew S.: **Computer Networks**. 4. Prentice Hall International, 2002. – ISBN 0130661023

The Internet Society 1999

The Internet Society: **Hypertext Transfer Protocol – HTTP/1.1**. http://www.ietf.org/rfc/rfc2616.txt, Veröffentlichung: 1999-06, Abruf: 2007-09-22

The Organization for the Advancement of Structured Information Standards (OASIS) 2002

The Organization for the Advancement of Structured Information Standards (OASIS): **RELAX NG Compact Syntax**. http://relaxng.org/compact-20021121.html, Veröffentlichung: 2002-11-21, Abruf: 2007-09-22

The Organization for the Advancement of Structured Information Standards (OASIS) 2003

The Organization for the Advancement of Structured Information Standards (OASIS): **RELAX NG Compact Syntax Tutorial**. http://relaxng.org/compact-20021121.html, Veröffentlichung: 2003-03-26, Abruf: 2007-09-22

UserLand Software 2000a

UserLand Software: **RSS 0.91 Specification**. http://www.rssboard.org/rss-0-9-1, Veröffentlichung: 2000-06-09, Abruf: 2007-08-15

UserLand Software 2000b

UserLand Software: **RSS 0.92 Specification**. http://www.rssboard.org/rss-0-9-2, Veröffentlichung: 2000-12-25, Abruf: 2007-08-15

Widom 1995

Widom, Jennifer: Research Problems in Data Warehousing. In: **CIKM '95: Proceedings of the fourth international conference on Information and knowledge management**. New York, NY, USA : ACM Press, 1995. – ISBN 0897918126, S. 25–30

World Wide Web Consortium (W3C) 2002

World Wide Web Consortium (W3C): **XHTML 1.0 - The Extensible HyperText Markup Language (Second Edition) (Recommendation)**. http://www.w3.org/TR/xhtml1/, Veröffentlichung: 2002-08-01, Abruf: 2007-09-22

World Wide Web Consortium (W3C) 2004
World Wide Web Consortium (W3C): **XML Schema Part 2: Datatypes Second Edition (Recommendation)**. http://www.w3.org/TR/xmlschema-2/, Veröffentlichung: 2004-10-28, Abruf: 2007-03-08. – David C. Fallside and Priscilla Walmsley

World Wide Web Consortium (W3C) 2007a
World Wide Web Consortium (W3C): **Simple Object Access Protocol (SOAP) 1.1 (Note)**. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/, Veröffentlichung: 2000-05-08, Abruf: 2007-09-22

World Wide Web Consortium (W3C) 2007b
World Wide Web Consortium (W3C): **SOAP Version 1.2 (Recommendation)**. http://www.w3.org/TR/soap12/, Veröffentlichung: 2007-04-27, Abruf: 2007-09-22

World Wide Web Consortium (W3C) 2007c
World Wide Web Consortium (W3C): **Web Services Description Language (WSDL) 1.1 (Note)**. http://www.w3.org/TR/wsdl, Veröffentlichung: 2001-03-15, Abruf: 2007-09-22

World Wide Web Consortium (W3C) 2007d
World Wide Web Consortium (W3C): **XML Path Language (XPath) 2.0 (Recommendation)**. http://www.w3.org/TR/xpath20/, Veröffentlichung: 2007-01-23, Abruf: 2007-09-23

World Wide Web Consortium (W3C) 2007e
World Wide Web Consortium (W3C): **XQuery 1.0: An XML Query Language (Recommendation)**. http://www.w3.org/TR/xquery/, Veröffentlichung: 2007-01-23, Abruf: 2007-07-07. – Scott Boag and Don Chamberlin and Mary F. Fernández and Daniela Florescu and Jonathan Robie and Jérôme Siméon

World Wide Web Consortium (W3C) 2007f
World Wide Web Consortium (W3C): **XSL Transformations (XSLT) Version 2.0 (Recommendation)**. http://www.w3.org/TR/xslt20/, Veröffentlichung: 2007-01-23, Abruf: 2007-09-23. – James Clark

9 Anhang

9.1 Übersicht konkreter Beispiele für mögliche Informationsquellen

Im Folgenden wird eine Übersicht über mögliche Informationsquellen gegeben. Die Angaben umfassen neben dem Namen und der Adresse auch die Arten der Informationen (Text, Bild, Ton oder Video), die Austauschformate sowie die Protokolle, über welche sie abgerufen werden können.

9.1.1 Nachrichten

Beispiele für Nachrichtenquellen werden in Tabelle 9-6 (S. 113) aufgeführt. Wo eine Angabe fehlt, konnten diesbezüglich keine genauen Informationen über den Dienst herausgefunden werden. Die Angabe „XML“ spricht für ein vom Anbieter nicht ferner spezifiziertes XML-Format. Die Austauschformate wurden nur für die Textinhalte bzw. das Containerformat erfasst. Für evtl. eingebettete Bild-, Ton- und Videoinhalte wurden die Formate nicht in der Tabelle erfasst. Die angegebenen Web-Seiten wurden alle am 15.08.2007 abgerufen.

9.1.2 Blogs

Im Gegensatz zu Tabelle 9-6 (S. 113) wird in den Beispielen für Blogs in Tabelle 9-7 (S. 114) nicht weiter auf die Art der Inhalte, das Austauschformat und die Schnittstellen eingegangen. Blogs können prinzipiell alle Inhaltsarten (Text, Bild, Ton, Video) enthalten. Das Austauschformat ist praktisch immer RSS oder Atom mit evtl. eingebettetem HTML. Der Abruf der Feeds erfolgt über HTTP-Pull (bzw. HTTP GET). Die angegebenen Web-Seiten wurden alle am 16.08.2007 abgerufen.

9.1.3 Bilder

Tabelle 9-8 (S. 115) gibt eine Übersicht über Beispiele für populäre Bilderdienste. Die angegebenen Web-Seiten wurden alle am 16.08.2007 abgerufen. Die Bilder werden meistens im JPEG-Format ausgeliefert.

Die angegebenen Beispiele sind bis auf zwei Ausnahmen kommerzielle, kostenpflichtige Dienste bekannter Nachrichtenagenturen. Kostenlose Alternativen stellen flickr und pixelio.de dar.

Tab. 9-6: Beispiele für Quellen, die Nachrichten liefern

Quelle/Dienst	Art der Inhalte	Austauschformat	Schnittstelle
AFP Finanzinfo[a]	Text, Bild	XML	FTP-Push
AFP Multimedia/Internet News[b]	Text, Bild, Video	NewsML, NITF, HTML, WML, ASCII	FTP-Push
AP Online[c]	Text, Bild, Ton, Video	XML	XML-Feed
ddp Webcontent[d]	Text, Bild	NewsML, Textformatierung unbekannt	FTP-Push
dpa-Webline[e]	Text, Bild, Ton, Video	NITF	FTP-Push
Heise Online[f]	Text, Bild	RSS, HTML	HTTP-Pull
Reuters Online Financial News[g]	Text, Bild	NewsML	Kinecta Interact Subscriber[h]
Reuters Online Reports[i]	Text, Bild, Video	NewsML	Kinecta Interact Subscriber
sid Internet-Sport-Dienst[j]	Text, Bild	HTML, ASCII, XML	FTP-Push
Spiegel Online[k]	Text, Bild, Ton, Video	RSS, HTML	HTTP-Pull

[a] Mehr Informationen zum Dienst AFP Finanzinfo findet man unter `http://www.united-news.de/download_file.html?6`.

[b] Eine Übersicht über die AFP Multimedia-Produkte gibt `http://www.afp.com/deutsch/products/?pid=online/internet`. Unter `http://www.united-news.de/` findet man das Portal für die Multimedia-Angebote der AFP.

[c] Der Dienst AP Online wird (neben weiteren Diensten) auf `http://www.apdigitalnews.com/news.html` kurz beschrieben.

[d] Der Dienst ddp Webcontent wird auf `http://www.ddp.de/produkte_dienste/webcontent/` beschrieben.

[e] Der Dienst dpa-Webline wird auf `http://www.dpa.com/de/produkte/multimedia/webline.html` beschrieben.

[f] Den Newsticker des Heise-Verlags findet man unter `http://www.heise.de/newsticker/`.

[g] Der Dienst Reuters Online Financial News wird auf `http://about.reuters.com/media/categories/online/financial/` beschrieben.

[h] Die Anleitung zum Kinecta Interact Subscriber findet man unter `http://www.about.reuters.com/ids/kinecta/textversions/kinecta_sub_guide.pdf`, eine Beschreibung der API unter `http://www.about.reuters.com/ids/kinecta/textversions/api_custom_guide.pdf`.

[i] Der Dienst Reuters Online Reports wird auf `http://about.reuters.com/media/categories/online/reports/` beschrieben. Leider ist er noch nicht für deutsche Nachrichten verfügbar.

[j] Der Internet-Sport-Dienst wird auf `http://www.sid.de/isd/unten_isd.html` beschrieben.

[k] Die Internetpräsenz des SPIEGEL findet man auf `http://www.spiegel.de/`.

Tab. 9-7: Beispiele einiger populärer Blogs

Titel	Adresse
BILDblog.	http://bildblog.de/
blog.tagesschau.de	http://blog.tagesschau.de/
Boing Boing	http://boingboing.net/
law blog	http://lawblog.de/
Lifehacker	http://lifehacker.com/
Spreeblick	http://spreeblick.com/
Techcrunch	http://www.techcrunch.com/

In der Foto-Community **flickr**[167] findet man z.B. mehrere Millionen Bilder, die unter einer Creative-Commons-Lizenz[168] veröffentlicht[169] werden. So werden zum Zeitpunkt des Verfassens dieses Textes allein über 10 Millionen Bilder angeboten, die kostenlos für kommerzielle Zwecke verwendet werden dürfen.[170] Nicht nur der Umfang und die kostenlose Verfügbarkeit, sondern auch die umfangreiche API hebt flickr von den anderen Bilddatenbanken ab. Man kann über REST, XML-RPC und SOAP auf den Bestand zugreifen. Die Daten bekommt man (abgesehen von den Bildern selbst) in den Formaten Atom, RSS oder in abfragespezifischen XML-Formaten geliefert.

Ebenfalls kostenlos ist die Bilddatenbank **pixelio.de**[171], die mit über 150.000 Bildern jedoch nur einen vergleichsweise kleinen Bestand aufweisen kann. Der Zugriff auf die Bilder erfolgt ausschließlich über den Webbrowser, derzeit existiert keine API zum Zugriff auf die Bilder. Die Nutzungsbedingungen[172] verlangen die Nennung von pixelio.de als Bildquelle.

9.1.4 Werbung

Insbesondere für Werbung auf Web-Seiten gibt zusätzlich zu den in Tabelle 9-9 (S. 116) dargestellten Beispielen eine Vielzahl weiterer Anbieter. Die Austauschformate, sowie die Schnittstellen für direkt geschaltete Werbung kann man als Anbieter selbst definieren. Nutzt man einen Vermittler, so wird er vorgeben, wie man als

[167] Die Foto-Community flickr erreicht man unter http://www.flickr.com/.

[168] Mehr Informationen zu den Creative-Commons-Lizenzen findet man unter http://creativecommons.org/about/ (Abruf: 2007-08-16).

[169] Die Fotos, die auf flickr unter einer Creative-Commons-Lizenz veröffentlich werden, findet man unter http://flickr.com/creativecommons/ (Abruf: 2007-08-16).

[170] Jedoch wird eine Namensnennung des Urhebers gefordert. Teilweise dürfen die Bilder nicht verändert werden oder sie müssen unter derselben Lizenz veröffentlicht werden wie das Ursprungswerk.

[171] Die Bilddatenbank pixelio.de erreicht man unter http://www.pixelio.de/.

[172] Die Nutzungsbedinungen von pixelio.de findet man unter http://www.pixelio.de/nutzungsbedingungen.php (Abruf: 2007-08-16).

Tab. 9-8: Beispiele für Bilddatenbanken

Quelle/Dienst	Schnittstelle
AFP Bildergalerie[a]	Webbrowser
AFP Fotodienst[b]	Satellit, Webbrowser
AFP Infografik[c]	Satellit, Webbrowser
AP Images[d]	Webbrowser
ddp Bilderdienst[e]	Webbrowser, FTP-Pull
dpa-Bilddatenbank[f]/Picture-Alliance[g]	Webbrowser, E-Mail, FTP-Pull
dpa-Bildfunk[h]	Satellit, Webbrowser
flickr[i]	Webbrowser, REST, XML-RPC, SOAP
pixelio.de[j]	Webbrowser
Reuters Pictures[k]	Webbrowser

[a] Die AFP Bildergalerie wird auf `http://www.afp.com/deutsch/products/?pid=online/bildergalerie` beschrieben. Unter `http://www.united-news.de/bildergalerie` findet man den Internet-Katalog, über den man auf die Bilder zugreifen kann.

[b] Der AFP Fotodienst wird auf `http://www.afp.com/deutsch/products/?pid=image/foto` beschrieben.

[c] Der Dienst AFP Infografik wird auf `http://www.afp.com/deutsch/products/?pid=image/graphics` beschrieben.

[d] Der Dienst AP Images ist unter `http://www.apimages.com/` zu finden. Neben Informationen zu diesem Dienst erhält man über diese Seite auch Zugriff auf den Dienst.

[e] Der ddp Bilderdienst wird auf `http://www.ddp.de/produkte_dienste/bilderdienst/` beschrieben.

[f] Die dpa-Bilddatenbank wird unter `http://www.dpa.com/de/produkte/bild/bilddatenbank.html` beschrieben.

[g] Der Zugriff auf die Bilder aus der dpa-Bilddatenbank und einiger weiterer Anbieter erfolgt über das Bildportal der Picture Alliance, das unter `http://www.picture-alliance.com/` zu finden ist.

[h] Der Dienst dpa-Bildfunk wird auf `http://www.dpa.com/de/produkte/bild/bildfunk.html` beschrieben.

[i] Die Foto-Community flickr findet man unter `http://flickr.com/`.

[j] Die kostenlose Bilddatenbank pixelio.de findet man unter `http://www.pixelio.de/`.

[k] Der Dienst Reuters Pictures ist unter `http://www.pictures.reuters.com/` zu finden.

Inhaltsanbieter die vermittelte Werbung integrieren kann. Im Falle von Google AdSense bindet man die Werbung über JavaScript in die eigene Web-Seite ein. Bei Google Print Ads werden potenzielle Werbeanzeigen mit einer Preisvorstellung an die Zeitung weitergeleitet, die dieses Angebot annimmt oder ablehnt.

Tab. 9-9: Beispiele für Möglichkeiten, in einem IID Werbung zu schalten

Typ	Art der Inhalte
Direkt geschaltete Kleinanzeigen	Text, evtl. Bild
Direkt geschaltete Produktwerbung	Text, Bild
Vermittler, z.B.:	
• Google AdSense	Text
• Google Print Ads	Text, Bild
• Microsoft adCenter[173]	Text
• Yahoo Publisher Network[174]	Text

9.1.5 Persönliche Assistenzen

Wie in Abschnitt 9.1.1 gilt auch hier, dass die Angabe „XML“ für ein vom Anbieter nicht ferner spezifiziertes XML-Format spricht. Ebenso wurden die Austauschformate nur für die Textinhalte bzw. das Containerformat erfasst. Die angegebenen Web-Seiten wurden alle am 16.08.2007 abgerufen.

Tab. 9-10: Beispiele für persönliche Assistenzen

Quelle/Dienst	Art der Inhalte	Austauschformat	Schnittstelle
Amazon[a]: Artikelsuche	Text	XML	SOAP, REST
eBay[b]: Artikelsuche	Text	XML, JSON	SOAP, REST
Flickr[c]: Neueste Fotos für einen Ort	Text, Bild	Atom, RSS, XML	REST, XML-RPC, SOAP
Google Calendar[d]: Meine Termine	Text	Atom	REST
ImmobilienScout24[e]: Angebote einer bestimmten Gegend	Text, Bild	XML	HTTP
Reuters Investor[f]: Aktienkurse, Wechselkurse	Text, Bild	HTML, XML	HTTP
Reuters Weather[g]: Wetterprognosen für einen Ort	Text, Bild	unbekannt	unbekannt
Upcoming[h]: Neue Ereignisse für meine Stadt	Text	XML	REST
Weather.com[i]: Wetterprognosen für einen Ort	Text	XML	REST
Yahoo-Finance: Aktienkurse[j]	Text/Zahlen	CSV	REST
Yahoo-Verkehrsinformationen[k] (derzeit nur USA)	Text	XML	REST

[a] Die Beschreibung der Amazon E-Commerce Service API findet man unter `http://www.amazon.com/E-Commerce-Service-AWS-home-page/b?node=12738641`.

[b] Die eBay-API wird auf `http://developer.ebay.com/common/api/` beschrieben.

[c] Die flickr-API wird auf `http://www.flickr.com/services/` beschrieben.

[d] Die Google-Calendar-API wird auf `http://code.google.com/apis/calendar/` beschrieben.

[e] Die API von ImmobilienScout24 wird auf `http://api.immobilienscout24.de/` beschrrieben.

[f] Der Dienst Reuters Investor wird auf `http://about.reuters.com/productinfo/investor/` beschrieben.

[g] Der Dienst Reuters Weather wird mit seinen verschiedenen Ausführungen auf `http://about.reuters.com/media/categories/online/other/weather/default.aspx?category=1064` beschrieben.

[h] Die Upcoming-API wird auf `http://upcoming.yahoo.com/services/api/` beschrieben.

[i] Für den Weather.com XML Data Feed kann man sich unter `http://www.weather.com/services/xmloap.html` anmelden.

[j] Den Dienst Yahoo! Finance findet man unter `http://finance.yahoo.com/`. Eine inoffizielle Beschreibung der API zum Abruf von Aktienkursen findet man auf `http://www.gummy-stuff.org/Yahoo-data.htm`.

[k] Die Yahoo Traffic Services REST API wird auf `http://developer.yahoo.com/traffic/rest/V1/index.html` beschrieben.

9.2 RELAX-NG-Schemata

Die RELAX NG Compact Syntax[175] wurde hier aufgrund der leichten Lesbarkeit der Schemadefinition gewählt.

9.2.1 RELAX-NG-Schema für die Serialisierung von Informationseinheiten

Wie in Abschnitt 5.5.4 erläutert wurde, muss jede Informationseinheit eine Methode zur XML-Serialisierung anbieten. Die serialisierten Daten müssen dabei dem unten definierten RELAX-NG-Schema entsprechen. Eine äquivalente XML Schema Definition (XSD) liegt ebenfalls dem Quellcode dieser Arbeit bei.

```
default namespace = "http://gedankenkonstrukt.de/infopool/schema/info"

start = infos | info

infos =
    element infos {
        attribute count { xsd:nonNegativeInteger }?,
        info*
    }

info =
    element info {
        element id { xsd:token },
        element date { xsd:dateTime },
        element type { xsd:token },
        element format { xsd:token },
        element content {
            attribute size { xsd:nonNegativeInteger }?,
            ( text | anyElement )*
        }
    }

anyElement =
    element* {
        ( attribute* { text } | text | anyElement )*
    }
```

Zum Vergleich sei hier nochmals das in Abschnitt 5.5.4 dargestellte Beispiel abgedruckt. Es kann mit Jing[176] gegen das Schema validiert werden.

[175] Für die Spezifikation der RELAX NG Compact Syntax siehe The Organization for the Advancement of Structured Information Standards (OASIS) (2002), für eine Einführung siehe The Organization for the Advancement of Structured Information Standards (OASIS) (2003).

[176] Jing kann man von der Seite http://www.thaiopensource.com/relaxng/ (Abruf: 2007-09-22) herunterladen.

```
<infos count="4" xmlns="http://gedankenkonstrukt.de/infopool/schema/info">
    <info>
        <id>unique234823749</id>
        <date>2007-09-12T22:49:35</date>
        <type>some binary text</type>
        <format>text/plain</format>
        <content size="3">Zm9v</content>
    </info>
    <!--
    <info>...</info>
    <info>...</info>
    <info>...</info>
    -->
</infos>
```

9.2.2 RELAX-NG-Schema für die Atom News Extensions (Atom/NX)

Das folgenden RELAX-NG-Schema zeigt die „News Extensions“ (NX) genannte Erweiterung[177] eines Atom-Eintrags:

```
# Atom/NX
namespace atom  = "http://www.w3.org/2005/Atom"
namespace nx    = "http://gedankenkonstrukt.de/infopool/schema/nx"
namespace local = ""

include "AtomRev.rnc" {
    simpleExtensionElement =
        element * - ( atom:* | nx:* ) { text }
    structuredExtensionElement =
        element * - ( atom:* | nx:* ) { structuredExtensionElementContent }
    undefinedAttribute =
        attribute * - ( xml:base | xml:lang | local:* ) { text }
}

atomFeedContent  &= nx
atomEntryContent &= nx & nxMedia*

nxMedia =
    element nx:media {
          nx
        & atomId
        & atomTitle?
        & atomSummary?
```

[177] Das Erweiterungsschema basiert auf einer Umformulierung des Atom-Schemas von Murata Makoto, die unter `http://www.asahi-net.or.jp/~eb2m-mrt/atom/schemaForAtomExtensions.zip` (Abruf: 2007-09-22) heruntergeladen werden kann.

```
        & atomUpdated?
        & atomPublished?
        & atomAuthor*
        & atomContributor*
        & atomLink*
        & atomRights?
        & atomContent?
        & extensionElement*
    }

nx =
      nxType
    & nxSubtitle?
    & nxSubsubtitle?
    & nxProvider*
    & nxLocation*

nxType        = element nx:type        { text }
nxSubtitle    = element nx:subtitle    { text }
nxSubsubtitle = element nx:subsubtitle { text }
nxProvider    = element nx:provider    { text }
nxLocation    = element nx:location    { text }
```

Das Schema erlaubt es, einen Atom-Feed oder -Eintrag mit Elementen für Typ, Untertitel, Unteruntertitel, Anbieter und Ort auszuzeichnen. Zusätzlich kann ein Atom-Eintrag mehrere Medienobjekte enthalten, die wiederum Titel, Untertitel usw. enthalten können. Ein Minimalbeispiel könnte wie folgt aussehen:

```
<entry xmlns="http://www.w3.org/2005/Atom"
 xmlns:nx="http://gedankenkonstrukt.de/infopool/schema/nx">
    <id>urn:uuid:234213374711</id>
    <nx:type>blog-entry</nx:type>
    <title>Look! Atom with news extensions!</title>
    <updated>2007-09-13T12:12:52Z</updated>
    <content type="xhtml">
        <div xmlns="http://www.w3.org/1999/xhtml">
            Hello World!
            <object data="#photo0" />
        </div>
    </content>
    <nx:media>
        <id>photo0</id>
        <nx:type>image</nx:type>
        <content type="image/jpeg" src="foobar.jpg" />
    </nx:media>
</entry>
```

9.3 Auswertung der NewsML- und NITF-Beispieldaten

Die unten dargestellen Ergebnisse wurden aus gut 300 NewsML-Instanzen aus dem AFP Live-Katalog, gut 470 NewsML-Instanzen aus einem Beispielpaket von Reuters sowie über 3300 NITF-Instanzen aus einem Beispielpaket der dpa erhoben. Für jedes Element im Baum wird die Häufigkeit des Auftretens dieses Elements grafisch darstellt.

Abb. 9-29: Ergebnis der Analyse der Häufigkeiten der tatsächlich benutzten Strukturelemente für NewsML-Instanzen der AFP und von Reuters

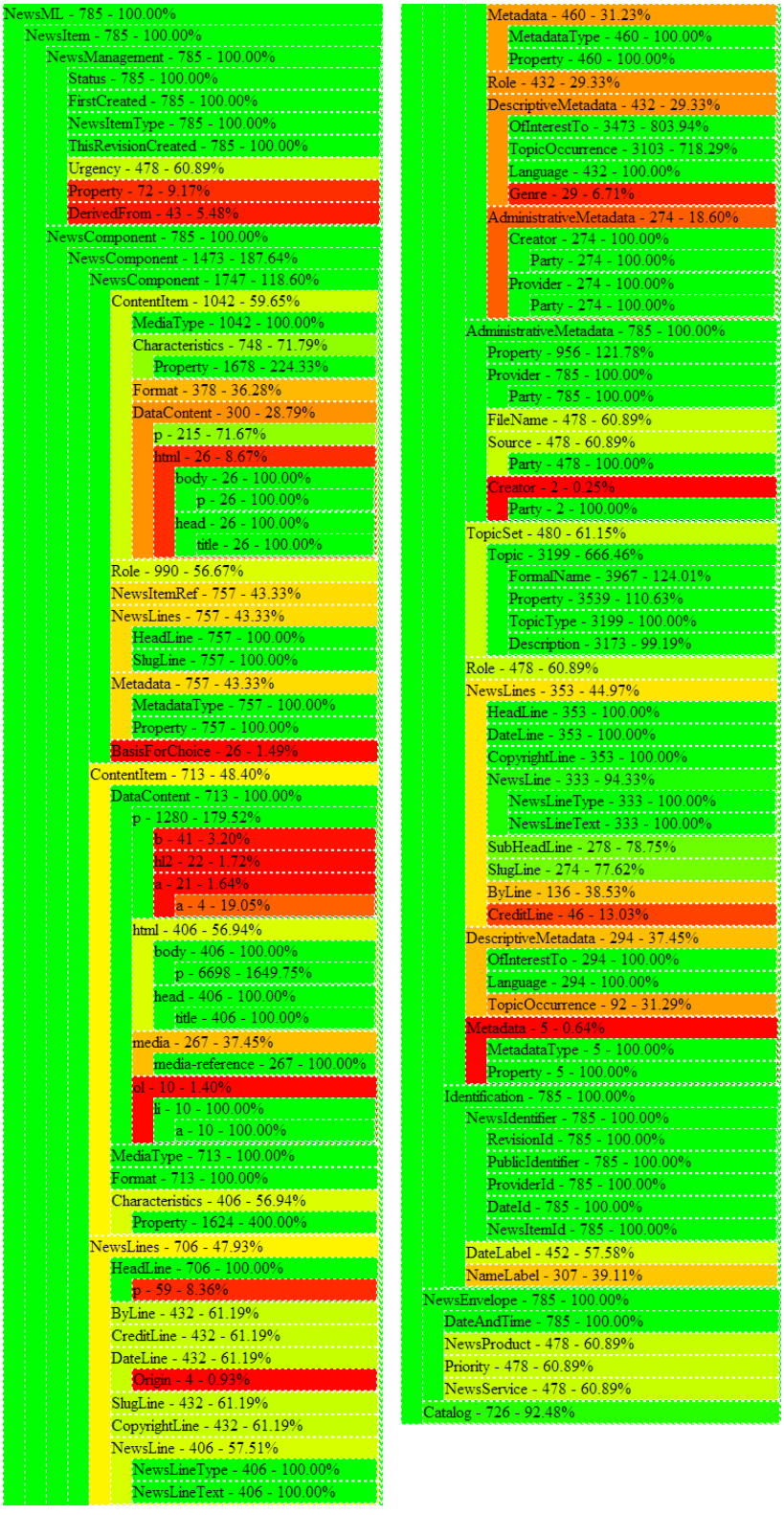
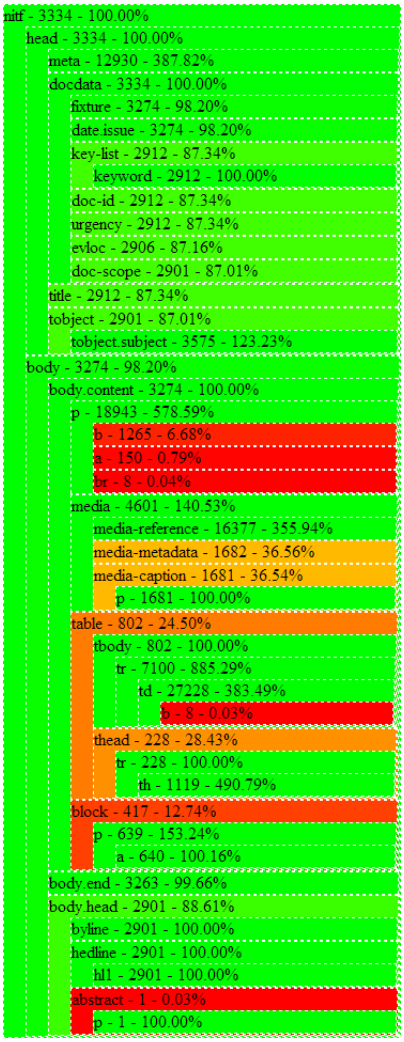


Abb. 9-30: Ergebnis der Analyse der Häufigkeiten der tatsächlich benutzten Strukturelemente für NITF-Instanzen der dpa



9.4 Übersicht verwendeter Bibliotheken und Werkzeuge

Die verwendeten Bibliotheken und Werkzeuge wurden in Kapitel 6 jeweils im Kontext der gerade dargestellten Lösung erwähnt. Einen Gesamtüberblick über die verwendeten Bibliotheken und Werkzeuge gibt Tabelle 9-11 (S. 125). Alle URLs wurden am 24.09.2007 abgerufen. Die Bedeutung der Lizenzkürzel finden sich im Abkürzungsverzeichnis (S. IX).

Tab. 9-11: Übersicht verwendeter Bibliotheken und Werkzeuge

Name	Version	Lizenz	URL
Altova XMLSpy	2007	kommerziell	http://www.altova.com/products/xmlspy/xml_editor.html
ANTLR	3.0.1	BSD	http://www.antlr.org/
ANTLRWorks	1.1.3	BSD	http://www.antlr.org/works
Eclipse	3.3	EPL	http://www.eclipse.org/
edtFTPj	1.5.5	LGPL	http://www.enterprisedt.com/downloads/ftp.html
eXist	1.1	LGPL	http://exist.sourceforge.net/
HttpClient	3.1	Apache License	http://jakarta.apache.org/httpcomponents/httpclient-3.x/
Jing	20030619	unbekannt	http://www.thaiopensource.com/relaxng/
JTidy	SVN 2007-09-18	©1998-2000 W3C	http://jtidy.sourceforge.net/
JUnit	4.3.1	CPL	http://www.junit.org/
Log4j	1.2.14	Apache	http://logging.apache.org/log4j/
ROME	0.9	Apache License	https://rome.dev.java.net/
Saxon	8.9	MPL	http://saxon.sourceforge.net/
SWT	3.3	EPL	http://www.eclipse.org/swt/
XOM	1.1	LGPL	http://www.cafeconleche.org/XOM/

10 Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Köln, den 29.09.2007

11 Lebenslauf

Der Lebenslauf soll neben den persönlichen Daten insbesondere Angaben zum Bildungsgang in Schule(n) und Hochschule(n), ggf. zum beruflichen Bildungsweg einschließlich Praktika sowie zu den abgelegten Prüfungen enthalten. Die Form ist freigestellt (z.B. tabellarisch.)

!!!