Diplomarbeit

Flexible Kopplung heterogener autonom verwalteter Datenquellen an ein XML-basiertes Web-Site-Management-System

– Konzepte und Implementierung –

von

Stefan Siegel

Kaiserslautern, Dezember 2001

Fachbereich Informatik

AG Datenbanken und Informationssysteme
Prof. Dr. Theo Härder

Universität Kaiserslautern – Postfach 3049 – D-67653 Kaiserslautern

Universität Kaiserslautern
Fachbereich Informatik
AG Datenbanken und Informationssysteme
Prof. Dr. Theo Härder

Flexible Kopplung heterogener autonom verwalteter Datenquellen an ein XML-basiertes Web-Site-Management-System

Diplomarbeit

von

Stefan Siegel

Betreuer:

Dipl.-Inform. Marcus Flehmig

Dezember 2001

❑ Zusammenfassung

Zur Erstellung einer personalisierten Web-Umgebung sollen heterogene Datenquellen an eine Portalarchitektur angebunden werden. Die Realisierung soll mit Hilfe eines XML-Wrapper-Konzepts erfolgen. Es sollen hier nun die notwendigen Anforderungen formuliert und ein entsprechendes Framework spezifiziert werden.

❑ Abstract

A portal architecture should be augmented with access to heterogenous data sources to create a personalized Web-environment. This approach should be realized with an XML-Wrapper concept. The intention of this document is to discuss the needs to be fulfilled by the components and to specify a framework according to the needs found.

❑ CR-Klassifikation

D.1.5. Object-oriented Programming
D.2.1. Requirements/Specifications
E.1. Data Structures
H.2.5. Heterogenous Databases
H.2.8. Database Applications

❑ Schlagwörter

Heterogene Datenquellen, Datenintegration, Framework, Wrapper, XML

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur angefertigt habe.

Kaiserslautern, den

_____

(Unterschrift)

Inhaltsverzeichnis

Kapitel 1 Einleitung

1.1 Motivation

Seit Beginn des *World Wide Web* (WWW) Anfang der 1990er Jahre hat die Menge an Informationen, die darin zur Verfügung gestellt werden unaufhaltsam zugenommen. Ursprünglich als reine Austauschplattform für wissenschaftliche Inhalte konzipiert, besteht es primär aus den beiden Komponenten:

- *Hypertext Markup-Language* [HTML99] (HTML), der Sprache zur Formulierung von Inhalt und Aussehen der Informationen in Form von Dokumenten, und dem zugehörigen
- *Hypertext Transfer-Protocol* [RFC2616] (HTTP), das für den Datentransport zwischen dem Datenserver und den anfragenden Komponenten (meistens Web-Browser) zuständig ist.

Es wurde schnell klar, dass das Potential, das mit diesem neuen Medium vorlag für ein weitaus breiteres Spektrum geeignet war, als nur wissenschaftliche Veröffentlichungen im universitären Umfeld zur Verfügung zu stellen. Inhaltsanbieter aus unzähligen Bereichen buhlen heute um die Gunst der Reisenden auf der virtuellen Datenautobahn.

Falls nur wenige, sich kaum ändernde Informationen bereitgestellt werden sollen, ist dies problemlos mit wenigen statischen HTML-Dokumenten möglich. Sollen jedoch ständig neue Inhalte angeboten werden, etwa Nachrichten, Wetterberichte oder Börsenkurse, so ist dieses Verfahren nicht mehr sinnvoll anwendbar. Stattdessen müssen existierende Datenquellen effizient an das bereitstellende Medium, konkret den Web-Server, angebunden werden.

Man stelle sich etwa den Internetauftritt einer Bank vor. Die Startseite enthält Informationen zu Sonderaktionen, die die Bank Ihren Kunden anbietet, etwa den kostenlosen Umtausch von alten DM-Münzen in Euro, wenn der Betrag auf ein Konto eingezahlt wird. Dieser redaktionelle Teil wird sinnvollerweise von einem Mitarbeiter der Bank erstellt und betreut.

Möchte die Bank ihren Kunden bereits auf der Startseite immer die aktuellen Börsenkurse präsentieren, wäre es sehr ineffizient, Personen abzustellen, die unaufhörlich die Startseite überarbeiten, um die sich minütlich ändernden Werte einzustellen. Stattdessen bedient man sich sinnvollerweise der elektronisch vorliegenden Kurse und bindet diese automatisch bei einem Seitenaufruf in das Dokument ein. Soll der Kunde auch gleichzeitig sein Aktien-Portfolio angezeigt bekommen (natürlich nur nach einer Authentifizierung am System), wird eine weitere Datenquelle, der Großrechner der Bank, in die Erstellung der Internetpräsentation eingebunden.

Der Web-Server muss also dynamisch neue Seiten aus unterschiedlichen Datenquellen generieren. In unserem Beispiel stehen drei verschiedene Arten von Quellen zur Verfügung:

1. Quasi-statische Inhalte, die von Mitarbeitern erzeugt werden.
2. Datenquellen, die dem Einfluss des Autors unterliegen (hier der bankeigene Großrechner, mit dem das Datenaustauschformat relativ problemlos angepasst werden kann) und
3. entfernte Datenquellen, auf die der Autor keinen Einfluss hat. Hier ist er auf die Kooperation der Gegenstelle angewiesen. Bietet diese die Daten nicht in einem verwendbaren Format an, oder ändert sich deren Format unvorhersehbar, sind Probleme unvermeidlich.

Die Praxis der vergangenen Jahren hat gezeigt, dass es mit HTML, als reiner Präsentationssprache, die keinen Hinweis auf den bereitgestellten Inhalt liefert, nur bedingt möglich ist, Daten aus unterschiedlichen Quellen zu sammeln, und als ein neues Dokument zu präsentieren oder gar zu integrieren. Da die erhaltenen Daten vom Web-Management-System weiterverarbeitet werden sollen, um sie wiederum einem interessierten Publikum zu präsentieren, wäre es sinnvoll, wenn eine Trennung von Inhalt und Aussehen vorliegen würde. Das verwendete Austauschformat sollte also Informationen über die Daten bereitstellen, statt Verarbeitungshinweise bzw. Satzinformationen wie Fettdruck, Kursiv, Überschrift oder Absatz zu übergeben (vgl. Syntax 1). Diese Möglichkeit bietet die *Extensible Markup Language* [XML00] (XML).

Syntax 1 HTML vs. XML

HTML

```
<p>
  <b>Infomatec</b> 0.24 <i>(0.22)</i>
  <b>Infineon</b> 33.04 <i>(16.20)</i>
</p>
```

XML

```
<firma id="Ieo">
  <name>Infineon</name>
  <kurs>33.04</kurs>
  <vortag>16.20</vortag>
</firma>
<firma id="Imt">
  <name>Infomatec</name>
  <kurs>0.24</kurs>
  <vortag>0.22</vortag>
</firma>
```

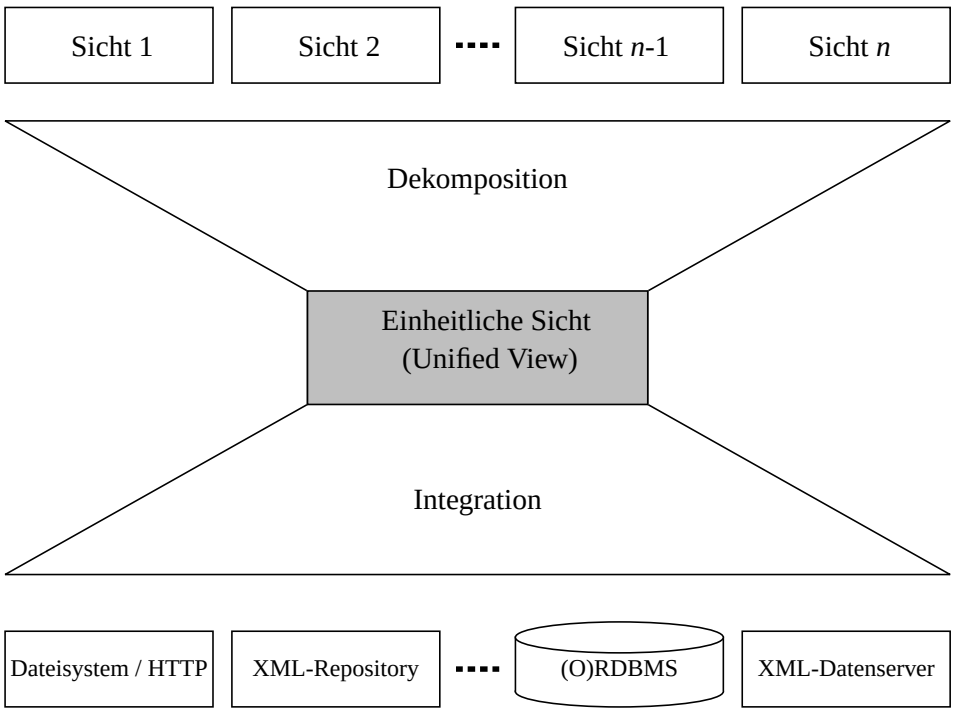
Es haben sich zwei Sichtweisen herausgebildet, wie mit XML Dokumenten umgegangen werden kann: eine dokumentorientierte und eine datenbankorientierte Sicht [ABS00]. Einerseits kann man die Dokumente als Ansammlung von Textfragmenten ansehen, die etwa direkt für die elektronische Veröffentlichung gedacht sind. Man spricht in diesem Zusammenhang auch von *„Presentation-Oriented Publishing“* (POP). Andererseits werden XML Dokumente auch immer häufiger als Datenaustausch-Format verwendet. Dies trifft insbesondere im Bereich der sog. *„Message-Oriented Middleware“* (MOM) zu.

Im Rahmen dieser Arbeit soll primär der zweite Ansatz verfolgt werden. Es soll diskutiert werden, wie mit Hilfe von sogenannten *Wrappern*, quasi kleinen Modulen die jeweils für eine Datenquelle verantwortlich sind, verschiedene XML Dokumente oder auch nur XML-Fragmente, aus diesen Quellen abgerufen werden können, um sie über eine *Einheitliche Sicht* an eine darüberliegende Applikation zu propagieren. Im konkreten Fall soll eine Web-Site-Management Anwendung mit den daraus resultierenden Schwerpunkten betrachtet werden.

1.2 Übersicht über das Gesamtprojekt

Mit dem Projekt SHARX soll gezeigt werden, wie mittels datenorientiertem, XML-basiertem Web Management in hochpersonalisiertem Umfeld gearbeitet werden kann. Zentraler Gedanke dieses Ansatzes ist die *Einheitliche Sicht* (*„Unified View“*) auf die Gesamtheit der zu Grunde liegenden Daten, die vom Systemverwalter oder sogar vom Endanwender auf unterschiedliche Bedürfnisse zugeschnitten, also personalisiert werden können (Vgl. Abbildung 1). Es erfolgt also eine Abstraktion von den physischen Speicherungsstrukturen.

Abbildung 1 Grobes Architekturschema von SHARX



Bei dem Projekt handelt es sich im weitesten Sinne um einen von der Mediator-Technik beeinflussten Ansatz, der von Gio Wiederhold [Wie92] eingeführt wurde.

Da die Datenquellen in unterschiedlichen Formaten vorliegen können, müssen diese in ein einheitliches Format gebracht werden. Das beinhaltet möglicherweise sogar die Wandlung in ein anderes Datenmodell. Weiterhin können die Quellen unterschiedliche Möglichkeiten bzgl.

Änderbarkeit der darin gespeicherten Daten oder Anfrageunterstützung bereitstellen. Diese Umsetzung der Datenquelleneigenschaften in eine für die Anwendung verständliche Form übernehmen sogenannte *Wrapper*, Komponenten die zwischen die Anwendung und die Datenquelle geschaltet werden, um die notwendigen Konvertierungen vorzunehmen.

Nachdem die unterschiedlichen Datenquellen in ein einheitliches Format konvertiert und über eine logische Eindokumentensicht zur weiteren Verarbeitung angeboten werden, wird aus diesem Dokument, das alle zur Verfügung stehenden Daten enthält, durch Anfragen und Auswahl eine beliebige Menge von Teilinformationen aufbereitet und dem Endanwender zur Verfügung gestellt. Somit wird es möglich, unterschiedliche Sichten des selben Datenbestandes anzubieten. Die beiden Prozesse (Integration und Dekomposition) sollen nun noch etwas genauer vorgestellt werden:

1.2.1 Integration

Die Datenintegration geschieht in 3 Schritten:

1. ***Schematransformation:*** Die in unterschiedlichen Datenquellen gespeicherten Informationen werden mittels sog. ***Wrapper,*** falls nötig konvertiert, um sie als XML Dokument(e) anzubieten.

2. ***Analyse:*** Der ***Partitionierer,*** versucht im zweiten Schritt Anfragen, die von der Anwendung kommen so aufzubereiten, dass er den einzelnen Wrappern genau die Teilanfragen schickt, die diese erledigen können und müssen.

3. ***Schemaintegration:*** Der XML ***Integrator,*** sammelt alle Teilergebnisse des Partitionierers auf und kombiniert sie zur *Einheitlichen Sicht*. Er kann beispielsweise ähnliche Quellen durch „umformen“ (etwa Umbenennen der Objektklammern) in der *Einheitlichen Sicht* als eine Objektmenge darstellen, falls dies erwünscht ist und Sinn macht. Die so bearbeiteten Daten weisen also in der *Einheitlichen Sicht* ein identisches Format auf.

1.2.2 Dekomposition

Da mit der Einheitlichen Sicht nicht nur alle Daten zur Verfügung stehen, sondern auch noch semantische Zusammenhänge durch die XML-Tags gegeben sind, sind im Folgenden umfassende persönliche Anpassungen möglich. So können etwa in unserem Beispiel von oben aus den Börsenkursen aller notierten Unternehmen exakt die ausgewählt werden, die den einzelnen Kunden interessieren. Es wird sogar möglich, konkrete Anfragen zu formulieren, etwa „Welche Unternehmen haben seit gestern einen Kursanstieg von mehr als 5% erlebt?“ oder „Gib mir die größten Verlierer seit dem Vortag, alphabetisch sortiert.“, „Welche Aktien sind heute für unter 20 Euro zu haben?“.

1.3 Einordnen dieser Arbeit in das Projekt

In dieser Arbeit soll es um die unterste Schicht der Integrationshierarchie gehen: die *Wrapper*. Es soll diskutiert werden, wie die unterschiedlichen Datenquellen einem Repository als XML-

Dokumente angeboten werden können. Dabei soll besonders auf die damit verbundenen Probleme wie Schematransformation, und Kodierung der Daten eingegangen werden.

1.4 Übersicht über ähnliche Projekte

Es gibt bereits eine Reihe von Projekten und kommerziellen Produkten, die sich mit ähnlichen Ansätzen wie SHARX beschäftigen. Es sollen hier einige vorgestellt, sowie Ähnlichkeiten und Unterschiede aufgezeigt werden.

Eine Möglichkeit, Projekte zu klassifizieren, ist der Grad der Daten-Materialisierung:

- beim sog. *„Warehousing“* Ansatz werden alle Daten in Form von Dokumenten materialisiert, bevor darauf gearbeitet wird,
- das andere Extrem ist eine Materialisierung nur genau der Daten, die für die aktuelle Anfrage benötigt werden.

Dazwischen gibt es Mischformen, die beide Ansätze verwenden.

1.4.1 STRUDEL

Bei STRUDEL [FFL+00], das 1999 in den AT&T Laboratorien entstand, handelt es sich um einen Ansatz zum Erstellen datenintensiver Web-Auftritte. Die Hauptidee, die verfolgt wird, ist die Aufspaltung der Dokumenterzeugung in drei Einheiten:

- Verwaltung der zu präsentierenden Daten,
- Spezifikation von Inhalt und Struktur,
- Erstellung der eigentlichen Seiten.

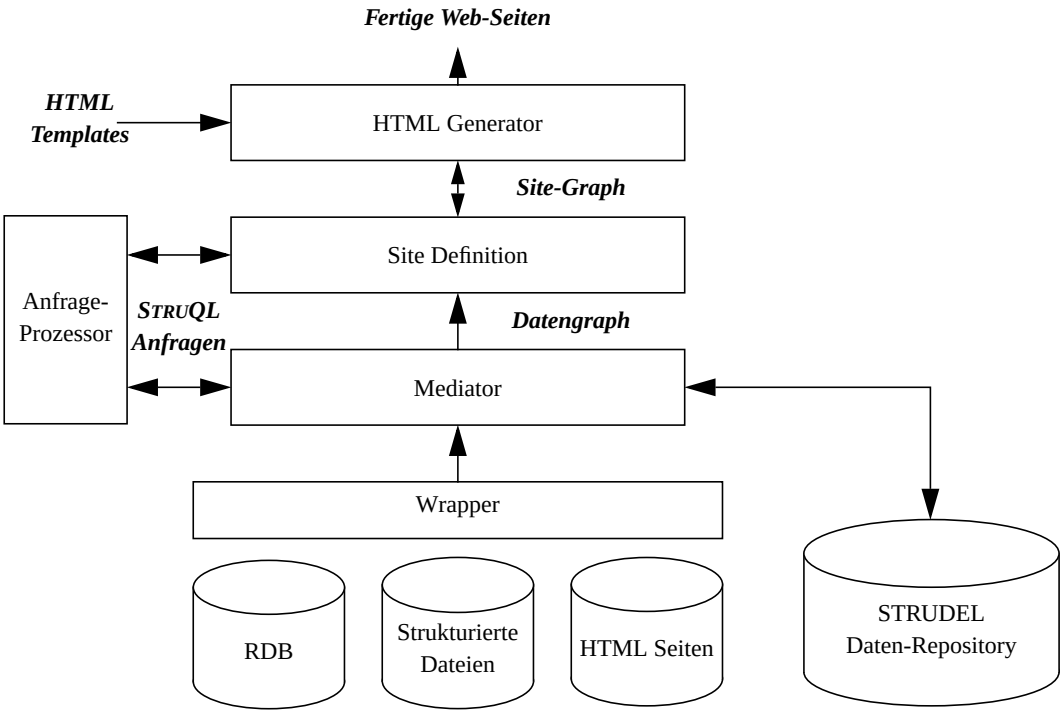
Strudel bietet sowohl eine beschreibende Anfragesprache (STRUQL), mit deren Hilfe Inhalt und Struktur der Seiten festgelegt werden können, als auch eine einfache Gerüstsprache (engl. *„template language“*), mit deren Hilfe das Aussehen der HTML-Dokumente erstellt bzw. beeinflusst werden kann. STRUDEL arbeitet auf dem semistrukturierten Datenmodell [Sic98], einer Variation des OEM (Object Exchange Model). Es zeichnet sich dadurch aus, dass es kaum Typzwänge gibt und dass ein sich schnell veränderndes oder gar kein Schema zu Grunde liegt. Die Repräsentation als gerichteter, beschrifteter Graph wurde gewählt, da dieser den Eigenschaften gängiger HTML-Dokumente entspricht. Die Autoren entschieden sich ursprünglich für einen reinen Warehousing-Ansatz, der die kompletten Daten, die mittels *Wrappern* in das System eingebunden werden, in einem Daten-Repository vorhielt, also materialisierte (vgl. Abbildung 2).

Die *Einheitliche Sicht* muss somit komplett vom System erstellt worden sein, bevor darauf gearbeitet werden kann. Man spricht bei einem solchen System von einem „Generator“.

Da sich dieser Ansatz bei großen Datenmengen als nicht sehr praxistauglich herausstellte, wurde in einer neueren Version auch die Möglichkeit der direkten dynamischen Anfrage auf Datenquellen implementiert.

Die Repository-Daten werden mittels STRUQL Anfrage in den Daten-Graphen gewandelt. Mittels einer weiteren STRUQL Anfrage wird aus der Site Definition der Site Graph erstellt. Ein Generator erzeugt dann anhand von unterschiedlichen Templates die unterschiedlichen Sichten.

Abbildung 2 Die Architektur von STRUDEL



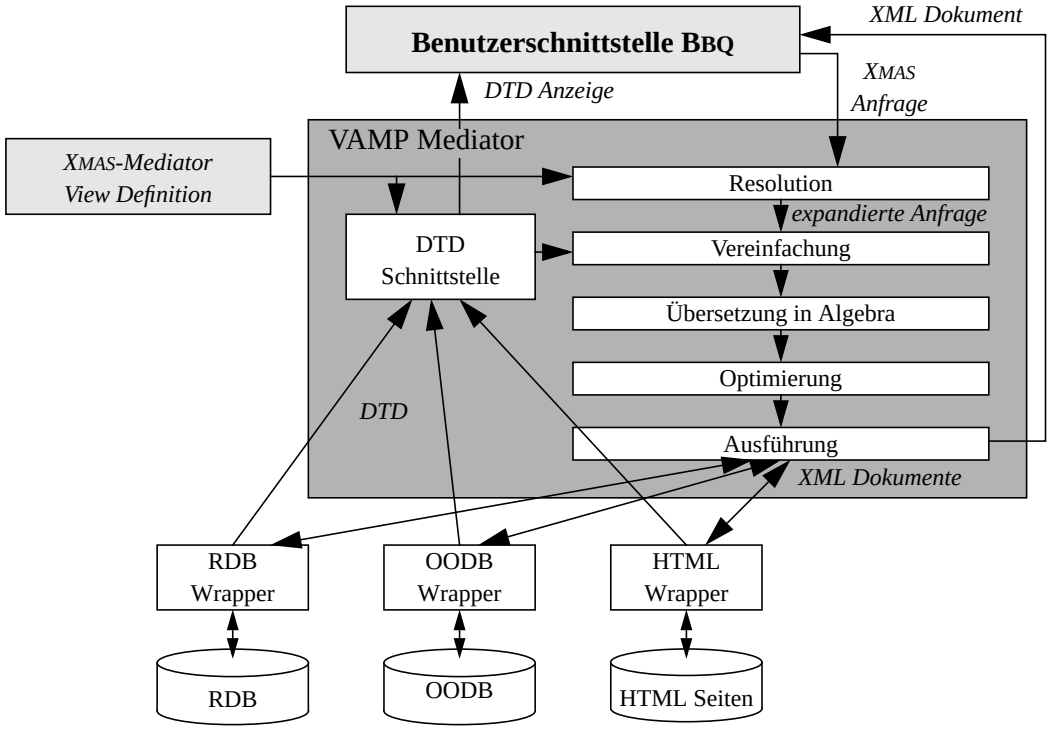
Größter Nachteil des ursprünglichen Ansatzes war die Notwendigkeit, alle Daten erst materialisieren zu müssen, um anschließend darauf arbeiten zu können. Dies ist in der Realität bei großen Projekten jedoch nicht sinnvoll realisierbar. Man denke an ein Warenhaus, dass seine Produkte (aktuellen Bestand, Produktinformation und Preise) in einer (relationalen) Datenbank vorhält. Diese Daten müssten also ständig extrahiert werden, um das Repository konsistent mit der DB zu halten. Detaillierte Erfahrungen der Autoren können in [FFK+98] nachgelesen werden.

1.4.2 MIX

Bei MIX [BGL+99] (*Mediation of Information using XML*), einem Kooperationsprojekt des San Diego Supercomputer Center (SDSC) und der Arbeitsgruppe Datenbanken der Universität von Kalifornien, Außenstelle San Diego (UCSD), haben die Autoren kein komplettes Web-Portal entworfen, sondern (wie der Produktname schon andeutet) nur mittels Mediator-Ansatz versucht Daten einheitlich an eine Benutzerschnittstelle (konkret BBQ – *„Blended Browsing and Querying“*) zu propagieren. Es wurde besonderen Wert darauf gelegt, dass die Daten über komplexe Anfragen an die darunterliegenden Datenquellen selektiert werden können. Bei diesem Projekt wurde – im Gegensatz etwa zu STRUDEL – keine komplette Materialisierung der Daten vorgenommen, sondern die eingehenden Anfragen so weit wie möglich an die Datenquellen „geschoben“, um nur die wirklich benötigten Datensätze im Speicher halten zu müssen.

Bei diesem Projekt wurde gezielt XML als einheitliches Format gewählt, da durch die Verwendung von DTDs, Strukturinformation mit einbezogen werden kann, um Anfrageoptimierungen vorzunehmen. Je nach Kapazität der Datenquelle beherrscht der Wrapper nur einen Teil der

Abbildung 3 Die Architektur von MIX



XML-Anfragen. In diesen Fällen muss der Mediator sehen, wie er die nicht unterstützten Anfragen umformen kann, um dennoch zum gewünschten Ergebnis zu kommen.

Wie in Abbildung 3 deutlich zu sehen, muss jede Datenquelle ihre DTD an die DTD-Schnittstelle propagieren, damit diese eine neue, zur Informationsanzeige geeignete DTD generieren kann.

Der Anwender generiert dann mittels BBQ eine XMAS-Anfrage (*„XML Matching and Structuring“*). Diese wird unter Zuhilfenahme des XMAS-Mediators in eine komplette Anfrage expandiert. Mit Hilfe der DTDs wird dann versucht eine Vereinfachung der Anfrage zu erreichen, die dann in eine Algebra überführt wird. Auf dieser Ebene wird versucht, mittels algebraischer Umformungen noch einmal Optimierungen zur erreichen, bevor die eigentliche Ausführung der Anfrage über die Wrapper an die Datenquellen gesandt wird. Die Kommunikation zwischen Mediator und Wrapper geschieht dabei durch Austausch von XML-Dokumenten. Das in der Ausführung gewonnene Dokument wird dann wieder an die Benutzerschnittstelle propagiert, wo es dem Anwender als Ergebnis seiner Anfrage angezeigt wird.

Kapitel 2 Datenmodelle

In diesem Kapitel soll ein kurzer Überblick über Datenmodelle gegeben werden, die im Web vorkommen. Es soll diskutiert werden in wieweit heterogene Datenquellen im jeweiligen Modell verwaltet werden können. Von besonderem Interesse ist dabei die Intention eine *Einheitliche Sicht* zu erstellen, aus der individuelle Sichten abgeleitet werden können, um diese personalisiert an den Endanwender zu propagieren. Die Rolle von XML als Vertreter des semistrukturierten Modells soll ebenfalls untersucht werden.

2.1 Relationales Datenmodell

Eines der am besten untersuchten Modelle der Datenbanken-Welt (DB) ist das sog. relationale Modell. Es beruht auf Arbeiten von E. F. Codd und anderen IBM Forschern um 1969 [Cod69]. Die Grundidee ist, dass alle Daten in Form von mathematischen Relationen repräsentiert werden.

Eine Relation besteht also aus

- ❑ einem Schema, das eine Menge von Attributen ist, wobei jedem Attribut ein Wertebereich zugeordnet wird.
- ❑ einer Ausprägung (auch Wert genannt).

Vergleicht man es mit einem anderen großen Modell der DB-Welt, dem objektorientierten Modell, kurz OO-Modell [ABD+90], werden alle Daten bildlich gesehen in Tabellen verwaltet, wobei jede Tabelle einer Objektmenge, jede Spalte einem Attribut und jede Zeile einem Objekt der Klasse entspräche.

Das Relationale Modell bietet einige Vorteile, die es für den Einsatz als *Einheitliche Sicht* sinnvoll erscheinen lassen:

- ✔ Es existiert bereits seit Jahrzehnten, ist daher auch bestens untersucht und erforscht worden.
- ✔ Es existieren genaue Vorstellungen (Algebra und Algorithmen) wie die Vereinigung verschiedener Tabellen auszusehen haben.
- ✔ Mit SQL [SQL92] existiert eine wohldefinierte Anfragesprache zum Umgang mit Daten, die in dieser Form vorgehalten werden.
- ✔ Es existieren ausgereifte, praxistaugliche Verfahren zur Anfrageoptimierung.

✔ DB-Systeme, die das relationale Paradigma verwenden, bieten die Möglichkeit, Transaktionseigenschaften (ACID [GR93]) zu fordern.

Dennoch gibt es auch Argumente, die gegen das Relationale Modell für die *Einheitliche Sicht* sprechen:

✘ Die Mehrheit der anzubindenden heterogene Datenquellen liegen nicht in relationaler Form vor.

✘ Im Web haben sich seit geraumer Zeit semistrukturierte Daten in Form von *Markup Languages* als Austauschformat etabliert (etwa HTML und XML).

✘ Untersucheungen haben gezeigt, dass die Semantik von semistrukturierten Anfragen auf XML-Dokumenten bei relationaler Speicherung nur sehr ineffizient (teilweise sogar überhaupt nicht) nachgebildet werden kann [DHN+99][FK99].

2.2 Semistrukturierte Daten

Häufig wird der Ausdruck ***Semistrukturierte Daten*** mit Begriffen wie *„selbstbeschreibend“* oder *„schemaless“* (engl. für: keine explizite Schemabeschreibung) erläutert [ABS00]. Das ist außergewöhnlich, da die klassische Herangehensweise in der Informatik damit beginnt, die Struktur eines Datums (Typ bzw. Schema) festzulegen. Anschließend werden dann Instanzen dieses Typs erzeugt. Bei Semistrukturierten Daten werden die Daten auf eine einfache Art beschrieben: Es handelt sich um Tupel aus Objekt und Wert:

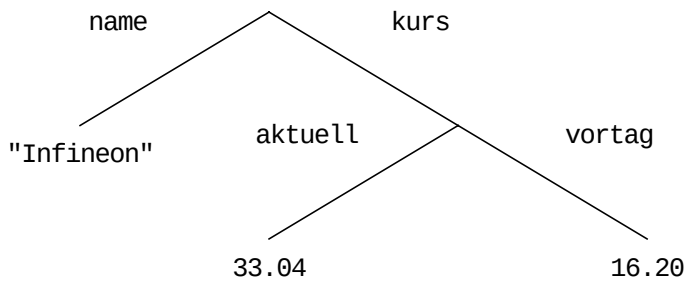
```
{name: "Infineon", kurs: 33.04, vortag: 16.20}
```

Die Werte können ebenfalls wieder solche Strukturen sein:

```
{name: "Infineon", kurs: {aktuell: 33.04, vortag: 16.20}}
```

In den meisten Fällen wird als grafische Repräsentation ein gerichteter Graph verwendet. Die Kanten repräsentieren dabei die Objekte, die Blätter enthalten Werte (vgl. Abbildung 4). Es ist daher möglich, von der Wurzel kommend, durch die verschiedenen Knoten zu navigieren.

Abbildung 4 Gerichteter Graph als Repräsentation Semistrukturierter Daten



Die Vorteile dieses Modells für das Erstellen der *Einheitlichen Sicht* wären:

- ✔ Relationale Datenbestände lassen sich problemlos auf dieses Modell abbilden, indem man etwa den Namen der DB als Wurzel, jede Tabelle als Ast erster Ordnung, jede Zeile als Ast 2. Ordnung und jede Spalte einer Zeile als Blatt modelliert.
- ✔ Unterschiedliche Datenquellen lassen sich durch Vereinigung in eine neue Quelle überführen, indem man sie durch eine neue Wurzel miteinander klammert. So werden aus Äpfeln und Birnen einfach Obst. Das ist etwa vergleichbar mit der Generalisierung in der OO-Welt.
- ✔ Das Web bietet eine große Anzahl semistrukturierter Datenquellen, die über ein etabliertes Protokoll (HTTP) abrufbar sind.
- ✔ Es lassen sich auch komplexere Zusammenhänge modellieren.

Aber auch dieses Modell ist nicht perfekt:

- ✘ Durch die hohe Flexibilität, die semistrukturierte Datenquellen bieten, wird es schwieriger Informationen aus unterschiedlichen Quellen miteinander zu verknüpfen, da unterschiedlich benamte Objekte dasselbe aussagen können. Es gibt also Fälle in denen bei unterschiedlicher Syntax selbe Semantik vorliegt.
- ✘ Identische Namen der Objekte aus unterschiedlichen Datenquellen implizieren nicht automatisch deren Verwandtheit. Eine identische Syntax kann also eine unterschiedliche Semantik haben.

2.3 XML als Vertreter semistrukturierter Datenmodelle

Es soll nun XML [XML00] als ein Vertreter aus der Gattung der semistrukturierten Datenmodelle etwas genauer betrachtet werden.

Ursprünglich wurde XML entwickelt, um der Herausforderung zu begegnen, die sich mit elektronischem Publizieren im großen Stil ergaben. Mittlerweile spielt es eine immer größere Rolle beim Datenaustauch, sowohl im Web, als auch zwischen Anwendungen.

XML bietet (laut [Con00]) :

- ❑ internationalisiertes, medienunabhängiges elektronisches Veröffentlichen von Daten,
- ❑ interessierten Branchen die Möglichkeit plattformunabhängige Datenaustauschprotokolle zu definieren, etwa für den elektronischen Handel,
- ❑ Anwendungen die Möglichkeit, Daten in einem Format zu erfragen, dass von diesen nach Erhalt verarbeitet werden kann,
- ❑ einen einfachen Ansatz, um Software zu erstellen, die spezialisierte, über das gesamte Web verteilte Informationen, verarbeiten kann,
- ❑ eine einfache Möglichkeit zur Datenverarbeitung mittels kostengünstiger Standardsoftware,
- ❑ dem Autor mit Hilfe von sog. *Style-Sheets* eine Möglichkeit festzulegen, wie seine Informationen repräsentiert werden sollen,

- zahlreiche Möglichkeiten, Metatateten (also Daten über die zur Verfügung gestellten Informationen) mitzuliefern, damit Informationen besser gefunden werden können. Somit haben Informationsproduzenten und -konsumenten die Möglichkeit sich schneller zu finden.

Laut offizieller XML-Spezifikation müssen Anwendungen beim Umgang mit inkorrekten XML-Dokumenten, wesentlich restriktiver handeln, als das etwa gängige Browser bei der Verarbeitung von HTML-Seiten tun: Wenn ein XML-Dokument nicht vollständig standardkonform ist, muss die Anwendung genau dort anhalten und eine Fehlermeldung ausgeben [XML00].

Die Daten werden in XML von Objekten begrenzt, quasi geklammert – daher der Name Klammerobjekt (engl. „*Tag*“). Jedes dieser Objekte besteht aus einer Start- und einer Ende-Klammer. Die Startklammern können auch zusätzliche Attribute enthalten. Vergleiche etwa Syntax 1, »HTML vs. XML«, auf Seite 2.

Es gibt auch Objekte die keine Daten klammern sollen. Um diese effizient und dennoch standardkonform zu schreiben, gibt es eine Kurzschreibweise, die das direkte Anhängen einer schließenden Klammer ersetzt.

XML-Dokumente können für unsere Zwecke in fünf verschiedene funktionale Klassen eingeteilt werden [Fle01]:

- operational,
- strukturierend,
- navigationsorientiert,
- funktional (oder logisch),
- abgeleitet.

Die erste Klasse enthält operationale Daten, also die eigentlichen Informationen, die dem Anwender angeboten werden sollen.

Mit Hilfe von strukturierenden Dokumenten kann Einfluss auf das Erstellen der *Einheitlichen Sicht* genommen werden. Sie enthalten geschachtelte Platzhalter (leere Elemente) um ein Gerüst der eigentlichen Sicht zu erstellen.

In navigationsorientierten Dokumenten sind Verweise zu weiteren Dokumenten oder anderen Teilen des gleichen Dokuments enthalten. Mit ihrer Hilfe werden Abhängigkeiten zwischen Dokumenten oder Dokumentfragmenten ausgedrückt. Sie müssen nicht Teil einer Datenquelle sein. Sie sind prädestiniert, Teile der *Einheitlichen Sicht* beliebig zu kombinieren.

Funktionale Dokumente enthalten einfache Anwendungslogik, sie definieren etwa dynamische Inhalte mittels *Simple Object Access Protocol*, kurz SOAP [SOAP00] (ein Protokoll zum Informationsaustausch in einer dezentralisierten verteilten Umgebung).

Um besser auf Wünsche und Interessen der Anwender eingehen zu können, werden Dokumente mit abgeleiteten Daten benötigt. Sie enthalten Informationen, wie das Benutzerverhalten einzelner oder einer Gruppe von Anwendern, bei gegebener Sicht.

2.3.1 Meta-Tags – Ein Versuch Inhalt zu klassifizieren

HTML als erster Vertreter der Textauszeichnungssprachen (engl. *„Markup Language“*) bot keinerlei Möglichkeit, aus der Form Rückschlüsse über den Inhalt des Textes zu geben. Dieser Dialekt wurde ja auch als Präsentationssprache konzipiert und nicht als Datenaustauschsprache.

Einen Versuch, diesen Mangel wenigstens teilweise zu beheben, unternahm man mit der Einführung sogenannter Meta-Tags. Sie bieten die Möglichkeit, Schlagwörter oder Inhaltsangaben im Dokument zu speichern, sodass etwa das Finden gewünschter Texte aus einer Menge von Dokumenten erleichtert werden sollte. Es stellte sich jedoch schnell heraus, dass dieser Ansatz nicht in die richtige Richtung zielte. Viele Webseiten-Betreiber versuchten nun Anwender auf ihre Seiten zu locken, indem in den beschreibenden Meta-Tags Begriffe verwendet wurden, die mit dem Inhalt der Seite gar nichts zu tun hatten. Die Suchmaschinen, die diese Fragmente als Hilfe zur Klassifizierung des bereitgestellten Inhalts untersuchten, wurden somit bewusst in die Irre geleitet.

2.3.2 Freie Wahl der Tag-Namen

Der große Vorteil, den XML gegenüber HTML besitzt – ähnlich wie sein Vorläufer SGML (*„Standard Generalized Markup Language“*) [ISO8879] – ist die Möglichkeit, die verwendeten Klammern zu definieren, und die Syntax damit genauer an die benötigten Bedürfnisse anzupassen.

Somit besteht zumindest die Möglichkeit auch Semantik direkt im Dokument zu halten.

2.3.3 Kodierung der Dokumente

Ein großes Problem des heutigen Datenbestandes ist die uneinheitliche Kodierung. In den Kindertagen der elektronischen Datenverarbeitung war Speicherplatz noch sehr teuer. Man begann also Daten in 7-Bit Kodierungen zu speichern. Viele Hersteller verwendeten eigene Standards. Das bekannteste Fossil aus dieser Zeit ist sicher EBCDIC [EBCDIC], ein proprietärer Standard von IBM.

Ein erster Vereinheitlichungsversuch der amerikanischen Normungsbehörde führte dann zum *„American National Standard Code for Information Interchange“*, kurz ASCII [ASCII86].

Anschließend wurden vom internationalen Normungsgremium ISO 8-Bit Kodierungen definiert, die auf ASCII als untere 128 Zeichen aufsetzen. Der heute meist gebräuchlichste ist Latin 1 [ISO8859-1], der wegen des fehlenden Euro Symbols in Zukunft immer häufiger von Latin 9 [ISO8859-15] Konkurrenz bekommen wird.

Doch auch dies ist nur eine Zwischenlösung, da etliche Zeichensätze dadurch nicht abgedeckt sind. So werden etwa Griechisch, Kyrillisch sowie die asiatischen Sprachen (um nur die gängigsten aufzuzählen) vollkommen vernachlässigt. Abhilfe schafft der UCS (*„Universal Multiple-Octet Coded Character Set“*) [ISO10646-1]. Die bekannteste Implementierung des Standards ist *Unicode*, speziell die Variante UTF-8. Es handelt sich dabei um eine platzsparende Serialisierung der UCS-Zeichen in eine Sequenz von ein bis vier Bytes. Der ASCII-Zeichenumfang bleibt dabei unverändert erhalten. Es ist bei englischsprachigen Texten daher interpretationssache, ob diese UTF-8 oder „nur“ ASCII kodieret sind.

2.3.4 XLink – Referenzen in Dokumenten

Da die Datenquellen, die wir betrachten wollen sich auch gegenseitig referenzieren können sollen, wird eine Spezifikation benötigt, wie dies formal auszusehen hat. Wir wollen mit XML-Bordmitteln nicht nur Verweise erstellen, die den Hyperlinks von HTML entsprechen, sondern sogar elegantere, ausgereiftere Varianten.

Ein wünschenswertes Beispiel hierfür sind bidirektionale Verweise. Also Verweise an eine andere Stelle der Einheitlichen Sicht, die wiederum Verweise auf die Ausgangsstelle enthält. Das ist zwar theoretisch bereits mit Hyperlinks machbar, führt jedoch zu einem sehr hohen Verwaltungsaufwand. Wenn eine der Verweisenden ihre Position ändert muss die Gegenseite manuell angepasst werden. Diesen Aufwand wollen wir uns sparen.

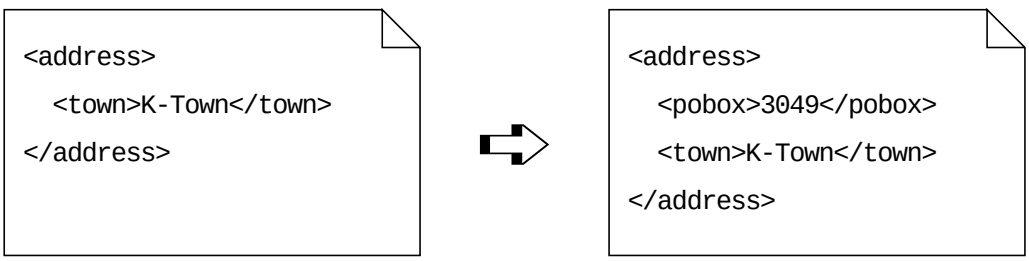
Leisten soll das die *XML Linking Language*, kurz XLink [XLink01] – eine Sprache die vom W3C („*World Wide Web Consortium*“) entwickelt wurde, um XML Dokumente mittels einfachen XML Elementen um ausgeklügelte Verweise zu erweitern.

2.3.5 XUpdate – Aktualisieren der Dokumente

Die *XML Update Language*, kurz XUpdate, deren Syntax und Semantik eine Arbeitsgruppe der XML:DB Initiative in einem Arbeitspapier [XUpdate00] vorstellt, bietet dem Anwender Möglichkeiten XML-Dokumente zu aktualisieren. Intention der Autoren war es, eine Implementierungsneutrale Sprache zu entwickeln. Zur Auswahl der zu aktualisierenden Elemente greift die Spezifikaton auf XPath-Ausdrücke [XPath99] zurück. Es werden folgende Änderungsmöglichkeiten angeboten: `insert-before`, `insert-after`, `append`, `remove`, `rename`, `variable`, `value-of` und bedingte Ausdrücke mittels `if`. In Abbildung 5 sieht man die Wirkungsweise an einem Beispiel.

Abbildung 5 Aktualisieren einer Adresse

```
<xupdate:insert-before select="address/town">
  <xupdate:element name="address">
  <pobox>3049</pobox>
  </xupdate:element>
</xupdate:insert-before>
```



Kapitel 3 XML Zugriffsschnittstellen

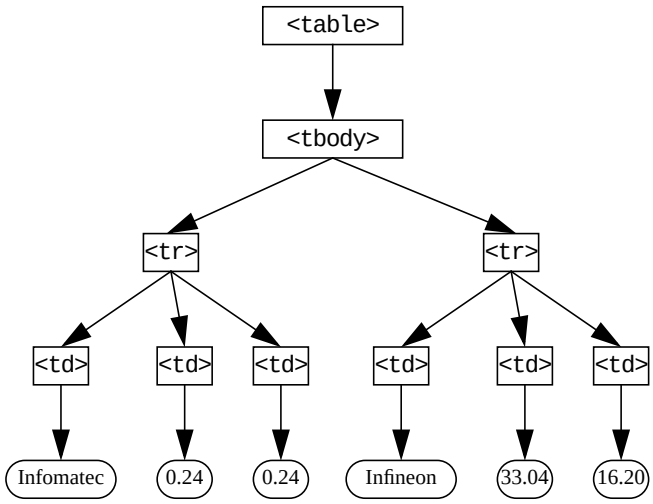
Dieses Kapitel soll einen Überblick über existierende Schnittstellen zum Zugriff auf XML-Datenquellen bieten. Konkret soll auf die zwei bekanntesten Ansätze (objekt- und ereignisbasiert) eingegangen werden.

3.1 DOM – Document Object Model

Das *Document Object Model* [DOM01], kurz DOM, ist eine Plattform- und Programmiersprachen-neutrale Schnittstelle, die es Programmen oder Skripten ermöglicht, dynamisch auf Inhalt, Struktur und Stil eines XML-Dokuments zuzugreifen, bzw. diese zu aktualisieren.

Abbildung 6 Grafische Repräsentation des DOM-Baumes

```
<table>
<tbody>
<tr>
<td>Infomatec</td>
<td>0.24</td>
<td>0.22</td>
</tr>
<tr>
<td>Infineon</td>
<td>33.04</td>
<td>16.20</td>
</tr>
</tbody>
</table>
```

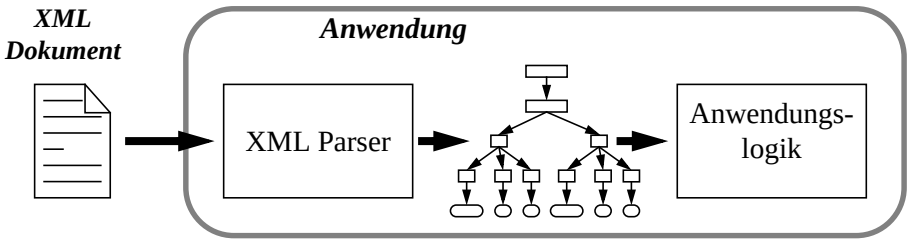


Die Idee zu DOM entstand als mit JavaScript und Java Klient-Programmiersprachen zur Verfügung standen, die die Möglichkeit boten, HTML-Seiten dynamisch zu erstellen bzw. zu ändern. Da ein plattform- und browserunabhängiger Mechanismus benötigt wurde, fand sich unter dem Dach des W3C eine Gruppe, die sich der Normung dieses Standards annahm. Schnell wurde klar,

dass dieses Konzept auch sinnvoll für weitere Auszeichnungssprachen, wie XML verwendet werden kann.

DOM verfolgt einen objektbasierten Ansatz, denn es verwaltet Dokumente als Baum. Dieser muss erst vollständig erzeugt werden, bevor die Anwendung darauf arbeiten kann (siehe Abbildung 7). Ist er jedoch erstellt, kann mittels klassischer Durchlaufalgorithmen durch das Dokument navigiert werden.

Abbildung 7 DOM Verarbeitungsschema



Der Nachteil dieser Herangehensweise liegt auf der Hand: Er ist sehr speicherintensiv. Nehmen wir etwa an, wir haben ein 20 MB großes Literaturarchiv-Dokument, von dem wir wissen es existiert genau ein Buchtitel mit dem Wort „Nebel“. Die Aufgabe lautet somit:

> Finde das erste Buch, dessen Titel das Wort „Nebel“ enthält.

Es wäre eigentlich nicht nötig für diese Anfrage den kompletten Baum zu materialisieren, doch bei der Verwendung von DOM ist es unvermeidlich. Mit einem ereignisbasierten Ansatz wäre diese Frage mit wesentlich weniger Speicherplatz lösbar.

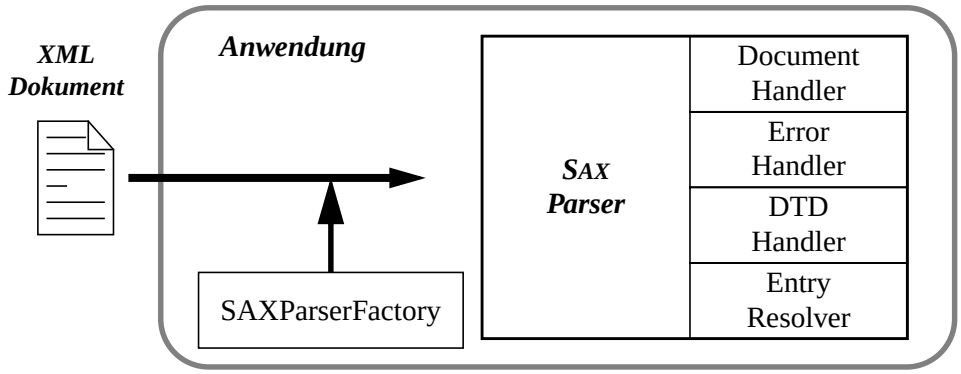
3.2 SAX – Simple API for XML

Die *Simple API for XML* [SAX01][SAX02][SAX03], kurz SAX ist ein ereignisbasierter Ansatz zur Verarbeitung von XML Dokumenten.

Peter Murray-Rust, der Autor eines Java-basierten XML-Browsers namens JUMBO entschied sich im Dezember 1997, eine ereignisbasierte Java-API zu entwickeln, der er den Namen YAXPAPI (*Yet Another XML Parser API*) gab. Nach einigen Diskussionen in der `XML-DEV` Maillingliste (Informationen dazu unter `http://www.xml.org/xml/xmldev.shtml`) kam er mit Tim Bary und David Megginson überein, einen Java Modul-Standard festzulegen, und zum offiziellen Bestandteil der Domäne `xml.org` werden zu lassen. Man einigte sich auf den Namen SAX. Es gibt bis heute keine formale sprach- und plattformunabhängige Spezifikation der API sondern nur informelle Texte der ursprünglichen Autoren, in denen Vorstellungen skizziert werden, was man unter SAX verstehen könne. Diese fehlende Standardisierung sowie der nicht vorgenommene markenrechtliche Schutz führten recht schnell zu einer Verwässerung des Ansatzes: große Unternehmen – an deren Spitze Microsoft – erstellten unter dem Namen SAX eigene Produkte, die zwar eine entfernte Verwandtschaft mit dem Java Original aufweisen, jedoch durchsetzt mit

eigenen Änderungen und Erweiterungen, sodass die Verwendung des Begriffes „SAX“ mittlerweile nur noch in Verbindung mit dem verwendeten Produkt Sinn macht.

Abbildung 8 Struktur der Java Version von SAX:



Die ursprüngliche Java-Variante des SAX Parser hat vier verschiedene Schnittstellen:

1. ***DocumentHandler***: Zu Beginn des Dokuments, an dessen Ende sowie bei Auffinden eines XML Klammer-Elements wird jeweils die entsprechende Methode aufgerufen: `startDocument`, `endDocument`, `startElement`, `endElement`. Wird Text zwischen den Klammern gefunden, wird die Methode `characters` aufgerufen.

2. ***ErrorHandler***: Die Methoden `error`, `fatalerror` bzw. `warning` werden je nach auftretendem Fehlerereignis aufgerufen.

3. ***DTDHandler***: Die Methoden dieser Schnittstelle werden aufgerufen, wenn DTD Definitionen verarbeitet werden müssen.

4. ***EntryResolver***: Die Methode `resolveEntry` wird aufgerufen, wenn der Parser beim Verarbeiten auf eine URI stößt.

Die Grundidee hinter SAX ist der Versuch eine *On-The-Fly* Dokumentverarbeitung vorzunehmen – es dem Programm also zu ermöglichen, mit der Verarbeitung des Dokuments bereits zu beginnen, wenn erst Teile vorliegen. So könnte ein Browser, der die Ausgabe des erhaltenen Textes vornehmen möchte, bereits mit der Visualisierung beginnen, während er noch auf die Beendigung der Übertragung warten muss.

Abbildung 9 zeigt, wie ein kleines XML-Fragment in SAX-Ereignisse umgesetzt wird. Für die Implementierung sei angemerkt, dass die textübergebenden Ereignisse nicht generell den kompletten Text bis zum nächsten Start/Stopp-Ereignis beinhalten müssen. Es ist möglich, dass dieser in Teilen propagiert wird. Möchte man also einen solchen Textbereich als gesamten Block auf einmal verarbeiten, muss man die Fragmente aller `characters`-Ereignisse sammeln und gemeinsam weiterreichen, wenn man auf ein Start/Stopp-Ereignis trifft.

Abbildung 9 Ereignisse beim Verarbeiten eines XML Fragments

```
<?xml version="1.0"
      encoding="UTF-8" ?>

<table>
<tbody>
<tr>
<td>Infomatec</td>
<td>0.24</td>
<td>0.22</td>
</tr>
</tbody>
</table>
```

```
start document
start element: table
start element: tbody
start element: tr
start element: td
characters: Infomatec
end element: td
start element: td
characters: 0.24
end element: td
start element: td
characters: 0.22
end element: td
end element: tr
end element: tbody
end element: table
end document
```

Wie es dazu kommen kann wird klar, wenn man sich überlegt was passiert, wenn beispielsweise im Text in Abbildung 10: „`Dies ist ein <emphasis>schöner</emphasis> Tag.`“ das hervorgehobene Wort „`<emphasis>schöner</emphasis>`“ ersetzt wir durch den Text „`normaler`“.

Abbildung 10 Entstehung mehrerer benachbarter characters-Ereignisse (1)

```
<text>Heute ist ein
<emphasis>schöner</emphasis>
Tag.</text>
```

```
start document
start element: text
characters: Heute ist ein
start element: emphasis
characters: schöner
end element: emphasis
characters: Tag.
end element: text
end document
```

Wir erhalten drei benachbarte `characters` Ereignisse, wie es in Abbildung 11 zu sehen ist. Es liegt nun am `EventHandler`, ob er in der Lage ist, diese zu einem zusammen zu ziehen, oder ob

Abbildung 11 Entstehung mehrerer benachbarter characters-Ereignisse (2)

```
<text>Heute ist ein
normaler
Tag.</text>
```

```
start document
start element: text
characters: Heute ist ein
characters: normaler
characters: Tag.
end element: text
end document
```

er sie als Einzelereignisse verarbeitet. Da darüber keine allgemeingültige Aussage möglich ist, muss die Anwendung immer damit rechnen, mit mehreren benachbarten `characters`-Ereignissen konfrontiert zu werden.

Schauen wir uns hier noch einmal das Beispiel aus Abschnitt 3.1 an: das 20 MB große Literaturarchiv-Dokument. Wollen wir nun die Frage

> Finde das erste Buch, dessen Titel das Wort „Nebel“ enthält.

beantworten, so bekommen wir jeweils einzelne Ereignisse und müssen nur warten, bis direkt nach einem „`start element: titel`“ eine Reihe `characters` Ereignis kommt, die den Begriff „Nebel“ enthalten. Der Dokumentserver kann sich mittels sequenziellem Zugriff immer kleine Teile der Datei, die das Dokument beinhaltet, laden, statt es komplett in den Speicher zu holen.

3.3 JAXP – Java APIs for XML Processing

JAXP, die *Java APIs for XML Processing* [JAXP01] ist ein sog. *Framework*, also eine Umgebung, die die Verarbeitung von XML Dokumenten mittels DOM, SAX, und XSLT vereinfacht, indem eine Java-Programmierschnittstelle vorgegeben wird. Durch die Einführung eines sog. *Plugability Layers* müssen die Module zur Verarbeitung der DOM und SAX Anfragen der darüberliegenden Anwendung nicht bekannt sein und können – falls gewünscht – auch erst zur Laufzeit ausgewählt werden.

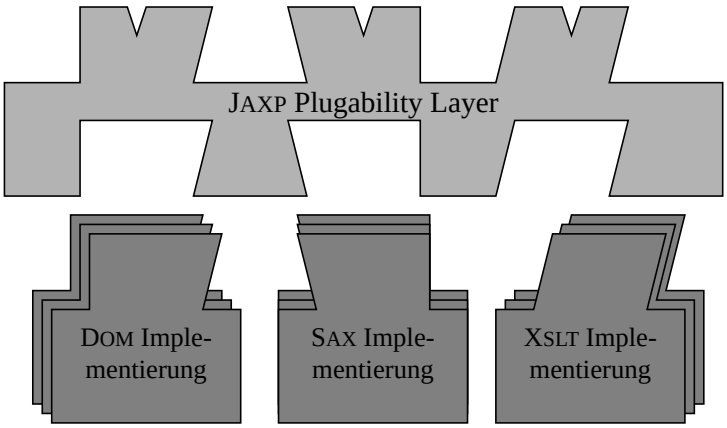
Diese Abstraktionsebene macht den Programmierer unabhängiger, da er im Falle einer inkorrekten Implementierung eines Moduls ohne große Programmänderungen eine Alternativversion verwenden kann (siehe Abbildung 12).

Die abstrakten Klassen `SAXParser`, `DocumentBuilder` und `Transformer` sowie deren Erzeugerklassen `SAXParserFactory`, `DocumentBuilderFactory` und `TransformerFactory` werden durch die Spezifikation leider nicht als threadsicher definiert. Es ist also sehr wahrscheinlich, dass bei der Verwendung der selben Instanz durch mehrere JAXP-Threads unerwünschte Seiteneffekte beim Parsen der XML-Dokumente auftreten. Somit muss der Programmierer eine geeignete Kapselung (etwa die Verwendung von Monitoren oder Semaphoren) vornehmen, damit immer nur ein Thread die Dokumentverarbeitung in Anspruch nehmen kann oder er muss über die Erzeuger-Methoden `newSAXParser`, `newDocumentBuilder` und `newTransformer` weitere Instanzen der Klassen erstellen, die laut Spezifikation threadsicher gegen andere Instanzen sind.

Die Verwendung dieses Konzepts anstelle einer direkten Einbindung von SAX- und DOM-Implementierungen bringt also die folgenden Vorteile:

- Threadsicherheit der einzelnen Instanzen
- Transparente Austauschbarkeit der DOM/SAX/XSLT Implementierungen

Abbildung 12 JAXP Plugability Layer



Kapitel 4 Anbinden von Datenquellen

In diesem Kapitel soll erarbeitet werden, wie die Anbindung heterogener Datenquellen an eine Anwendung realisiert werden kann. Wo sich bei diesem Versuch Probleme ergeben können und wie man diesen begegnen kann.

4.1 Problemfälle

Stellt man sich eine Anwendung vor, an die Datenquellen angebunden werden sollen, so fallen einem recht schnell Aspekt ein, die Probleme bereiten können, oder zumindest eine genauere Betrachtung benötigen.

4.1.1 Verteilte Quellen

Ein Aspekt, der beim Versuch Datenquellen an eine Anwendung anzubinden recht schnell auftritt, ist die Tatsache, dass die Quellen sich auf einem anderen Rechner befinden. Es muss also eine Möglichkeit vorgesehen werden, mit datenanbietenden Anwendungen auf entfernten Rechnern in Kontakt treten zu können. Diese Mechanismen besitzen häufig einen Kommunikationsüberbau, der beim Zugriff auf lokalen Systemen vermieden werden kann. Es ist daher ratsam, unterschiedliche Zugriffsmechanismen vorzusehen. Man benötigt also eine „universelle“ Variante, die alle Quellen über Netzwerkprotokolle anspricht sowie eine Variante, die es erlaubt auf lokale Quellen effizienter zuzugreifen. Sinnvoll wäre es auch, die Lage der Quelle für die Anwendung transparent zu halten. Dies bietet die Möglichkeit, zur Lebenszeit der Anwendung die Quelle zu migrieren. Man erhält somit eine Anwendung, die besser skalierbar ist. Auf die verschiedenen möglichen Szenarien wird später noch genauer eingegangen.

4.1.2 Heterogene Datenmodelle

Sollen Daten aus unterschiedlichen Quellen zusammengeführt werden, muss sichergestellt sein, dass dies auf einem einheitlichen Datenmodell (vgl. Kapitel 2) geschieht. Das Modul, das für die Anbindung der Datenquelle zuständig ist, muss daher die möglicherweise notwendige Konvertierung vornehmen.

4.1.3 Unterschiedliche Kodierungen

Durch den Wunsch unterschiedliche Quellen zu unterstützen, wird man recht schnell mit dem Problem konfrontiert, dass die Daten in unterschiedlichen Kodierungen repräsentiert werden. Man muss sich also für eine einheitliche Kodierung entscheiden und gewährleisten, dass die Quellen in diesem Format angeboten werden.

4.2 Der Mediator Ansatz

Bereits im Herbst 1991 befasste sich Gio Wiederhold in einem Artikel [Wie92] mit Mediatoren als Konzept für „zukünftige Informationssysteme“. Er prägte den Begriff „Mediator“ – ein Modul, das in der Lage sein soll, quasi als Vermittler eine Datenquelle eines verteilten Informationssystems in ihrem nativen Format anzusprechen und die erhaltenen Informationen dem anfragenden System in einer ihm verständlichen Form anbieten zu können. Wichtig ist dabei, dass ein Mediator genügend verwaltungstechnisches Wissen haben muss, um eine Entscheidungsfindung (etwa Anfrageoptimierungen) vornehmen zu können. Wiederhold fand zwei zentrale Probleme:

1. ein Mangel an Datenabstraktion (vgl. Abbildung 13). Die vorhandenen Basisdaten für sich genommen sind häufig nicht sehr aussagekräftig. Stattdessen ist man eher an abgeleiteten Daten interessiert. Ein Controller etwa ist nicht an den einzelnen Tagesumsätzen eines Produkts interessiert, sondern eher an den saisonbereinigten Monatsumsätzen. Diese liegen jedoch nicht als Basisdaten vor. Der Mediator muss also in der Lage sein sie durch geschicktes Befragen und Aufbereiten der vorhandenen Daten selbst zu erzeugen.

2. Anpassungsschwierigkeiten, wenn Informationen aus verschiedenen Datenquellen kombiniert werden sollen. Etwa Mehrdeutigkeiten die sich aus der Informationsrepräsentation ergeben:

   > Zwei Spalten in DB-Tabellen, die beide „Einkommen“ heißen, können unterschiedliche Bedeutungen haben: Persönliches Einkommen eines Angestellten in einer Firmen-DB $\leftrightarrow$ zu versteuerndes Familieneinkommen beim Finanzamt

   Oder Inkonsistenzen im Speicherungsformat:

   > Speicherung des selben Geldbetrages:
   > in Eurocent als Ganzzahlen $\leftrightarrow$ in Euro mit Nachkommastellen als Fließkommazahlen

Wiederholds Grundidee, Verschiedenen Anwendern über Mediatoren einen transparenten personalisierten Zugriff auf verschiedene Datenquellen zu bieten, erreichte schnell eine größere Popularität durch das aufkommende *World Wide Web* und den damit weltweit zur Verfügung stehenden Datenquellen, die über ein Standardprotokoll (HTTP) zugänglich wurden.

Der Mediator Ansatz basiert jedoch auf leicht anderen Annahmen, als die, dieser Arbeit zu Grunde liegenden: Da Wiederhold bei seinen Überlegungen ein einheitliches Datenmodell (relationale DBS) für seine Datenquellen voraussetzt, besteht keine Notwendigkeit für eine Datenkonvertierung (vgl. Abbildung 11). Er beschäftigt sich daher direkt mit dem Anbinden realer Datenbanken an Arbeitsplätze (Workstation) und den damit verbundenen Problemen.

Abbildung 13 Abstraktionsfunktionen

Abstraktionsart	Beispiel Basisdaten	Beispiel Abstraktion
Granularität	Verkaufszahlen	Produktgruppenübersicht
Generalisierung	Produktkenndaten	Warengattung
Zeitreihung	Tagesumsatz	saisonbereinigter Monatsumsatz
Relativierung	Produktionskosten	inflationsbereinigte Markttendenzen
Ausnahmenerkennung	Buchführung	Unterschlagungsnachweis
Pfadberechnung	Flugdaten einer Fluggesellschaft	Reisezeit und -kosten einer Dienstreise

Abbildung 14 Die Drei Ebenen Architektur des Mediator Ansatzes

Ergebnis → Treffen von Entscheidungen

Ebene 3 Unabhängige **Anwendungen** an Arbeitsplätzen – Verwaltet von Entscheidern

... Netzwerkdienste zu Informationsservern ...

Ebene 2 Mehrere **Mediatoren** – Verwaltet von Bereichsspezialisten

... Netzwerkdienste zu Datenservern ...

Ebene 1 Mehrere **Datenbanken** – Verwaltet von DB-Administration

Eingabe ← Änderungen in der realen Welt

In dieser Arbeit soll jedoch davon ausgegangen werden, dass heterogene Datenquellen zur Verfügung stehen, etwa relationale DBS, Dateien aus Dateisystemen und XML-Dokumente aus unterschiedlichen Quellen.

4.3 Die Verwendung von Wrappern

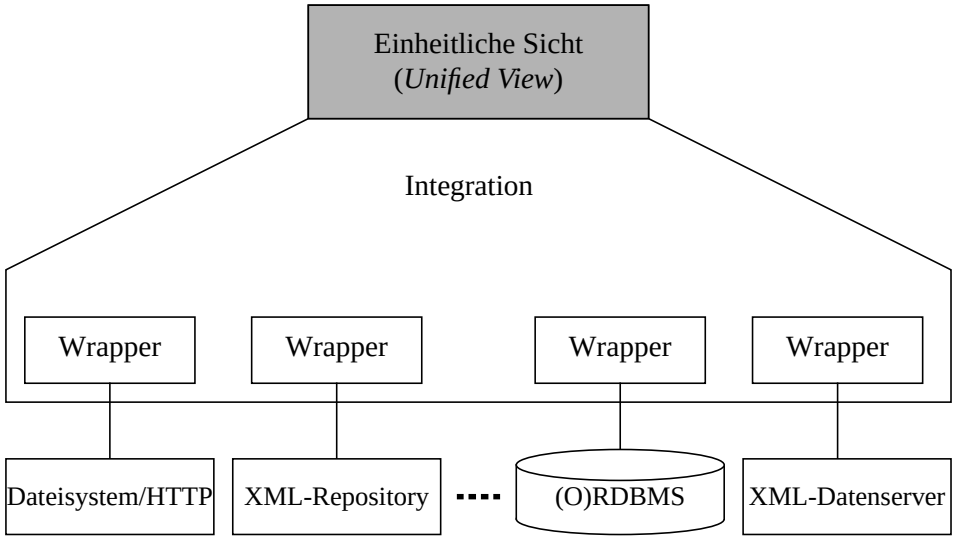
Als Reaktion auf den Wunsch, unterschiedliche Datenquellenformate zu unterstützen, müssen nun natürlich Maßnahmen ergriffen werden, damit dies transparent für den Anwender geschieht. Er soll nicht mit den unterschiedlichen Eingangsformaten konfrontiert werden, sondern auf einer einheitlichen Datensicht arbeiten können. Daher muss der Mediator-Ansatz um eine Schicht erweitert werden, die diese Schematransformation vornimmt. In der Literatur findet sich hierfür immer häufiger der Begriff *„Wrapper“*. Es handelt sich dabei also um Module, die speziell auf eine Datenquelle zugeschnitten sind.

Sie müssen zumindest die Minimalanforderung „Propagieren der durch sie bereitgestellten Datensätze im einheitlichen Format“ erfüllen. Hilfreich wäre es natürlich, wenn auch weitere Eigenschaften des zu Grunde liegenden Datenhaltungssystems (etwa eine DB) der anfragenden

Anwendung zugänglich gemacht werden könnten (etwa Selektions- oder Aggregationsmöglichkeiten). Nur so wird es möglich, dass der Mediator die von ihm geforderte Aufgabe (Hilfe bei der Anfrageoptimierung) auch wirklich wahrnehmen kann.

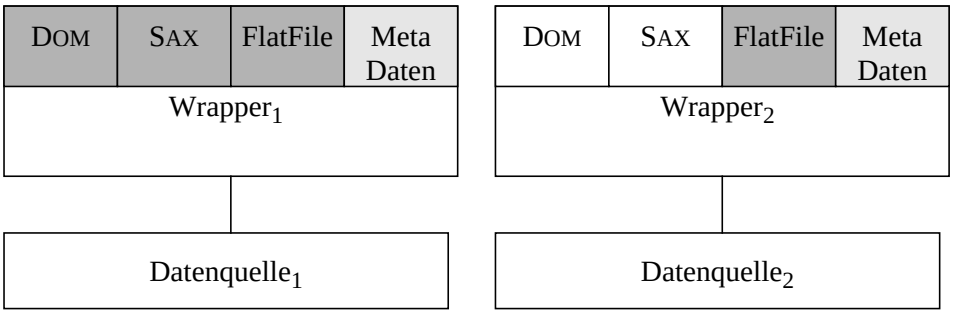
Der Integrations-Teil unseres Architekturschemas wird daher um eine Schicht, die für die Datenkonvertierung zuständig ist, erweitert (siehe Abbildung 15).

Abbildung 15 Verfeinerung des Architekturschemas



Bevor ein Wrapper seinen Dienst aufnehmen kann müssen der Anwendung einige Dinge bekannt gemacht werden. Daher ist es sinnvoll eine Metadaten-Schnittstelle vorzusehen. Mit ihrer Hilfe können Informationen, wie die Protokolle mit denen die Daten angeboten werden oder Eigenschafen der Datenquelle, etwa Anfrage- und Änderungsmöglichkeiten, bereitgestellt werden.

Abbildung 16 Verfeinerung der Wrapper



In Abbildung 16 sieht man 2 Wrapper, die jeweils für eine Datenquelle verantwortlich sind. Wrapper 1 beherrscht zusätzlich zur Übergabe des ganzen Dokuments (Schnittstelle FlatFile) auch noch DOM und SAX, wohingegen Wrapper 2 nur die Dateien als Ganzes übergeben kann.

Ist die Quelle etwa in der Lage neue Dokumente oder Dokumentfragmente zu erstellen, vorhande zu löschen oder zu ändern, so soll dies der Anwendung mitgeteilt werden. Diese kann davon dann regen Gebrauch machen.

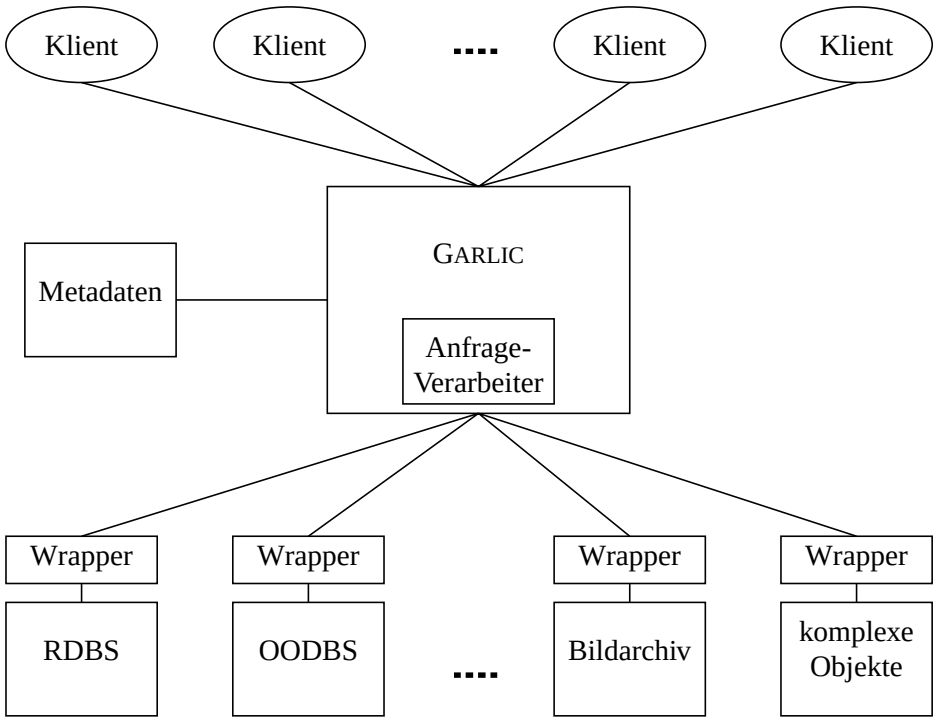
4.4 GARLIC – ein Middleware System

Bei GARLIC [TRS97] handelt es sich um ein Middleware System des IBM Almaden Forschungszentrums, das auf dem Mediator Ansatz aufbaut.

Die einzelnen Quellen werden durch Wrapper aufbereitet, sodass der Anwender den physischen Aufbewahrungsort der Daten nicht kennt (Ortstransparenz). IBM redet in diesem Kontext von Kapselung der Datenquellen, da es sich bei GARLIC um eine objektorientierte (OO) Implementierung in C++ handelt. Zentrale Aufgaben fallen in diesem Konzept den Wrappern zu, die Daten der Middleware als Objekte zur Verfügung stellen sollen.

Der vermittelnde Teil – hier Anfrage-Verarbeiter genannt – befindet sich im GARLIC Kern. Er bereitet die in den Datenquellen gehaltenen Informationen mit Hilfe der Wrapper für die einzelnen Klienten auf. Zur Unterstützung der Verarbeitungsaufgaben steht GARLIC ein Metadatenspeicher zur Verfügung. Deutlich wird das in Abbildung 17.

Abbildung 17 Die Architektur von GARLIC



Die Entwickler haben sich vier (nicht formale) Ziele gesetzt, die sie mit ihrer Architektur verfolgen:

1. Die Ersterstellungskosten für einen Wrapper sollen gering sein.

   Anwendern soll über eine einheitliche Programmierschnttstelle (API) die Möglichkeit gegeben werden, ohne genaues Wissen über GARLICs interne Struktur binnen Stunden einen Wrapper schreiben zu können, der auf die speziellen Ansprüche des Anwenders abgestimmt ist.

2. Wrapper sollen in der Lage sein sich weiterzuentwickeln.

   Die Wrapper haben nur einen Minimalumfang an Voraussetzungen zu erfüllen (s. u.). Sie können dann immer weiter auf die Bedürfnisse GARLICs zugeschnitten werden (etwa bzgl. Anfrageoptimierung).

3. Die Architektur soll flexibel sein und Erweiterungen im laufenden Betrieb ermöglichen.

   Da jeder Wrapper für genau eine Datenquelle verantwortlich ist, stört es ihn nicht, wenn im laufenden Betrieb weitere Datenquellen angebunden werden. Ebenso werden Anwendungen, die nur auf einem Teil der Datenquellen arbeiten nicht dadurch gestört, dass über die einheitliche Sicht plötzlich mehr Daten zur Verfügung stehen, da sie auf diese einfach nicht zugreifen.

4. Die Architektur soll Ansätze zur Anfrageoptimierung bereitstellen.

   Ein Wrapper muss nur in der Lage sein, auf Anfragen zu reagieren. Sind dem Autor des Wrappers jedoch spezielle Informationen über die verwalteten Daten bekannt (bei Bildern etwa das Motiv oder die Quelle des Originals), so können diese dem System mitgeteilt werden und den Wrapper dadurch mit in den Optimierungsprozess einer Anfrage einbinden.

Ein GARLIC Wrapper muss vier Dienste bieten. Erstens muss er den Inhalt der von ihm verwalteten Quelle als GARLIC-Objekt bereitstellen, damit es von GARLIC verarbeitet werden kann. Zweitens muss der Wrapper es ermöglichen, Methoden auf diesen Objekten auszuführen und deren Attribute zu erfragen. Das ist der zentrale Mechanismus für GARLIC, mit dessen Hilfe Daten aus der Quelle abgerufen werden können, selbst wenn die Quelle (etwa das Dateisystem) eigentlich gar keine Anfragemöglichkeiten bietet. Wenn GARLIC Daten eines Wrappers zur Bearbeitung einer Anfrage benötigt, nimmt dieser drittens aktiv an Anfrageoptimierungen teil. Das ist notwendig, da die GARLIC keine Informationen sammelt, die ihm helfen können eine Optimierung durchzuführen, sondern es verlässt sich darauf, dass die Wrapper ihm im Einzelfall mitteilen wenn sie ihm Arbeit abnehmen können. Der Wrapper bestimmt also eigenständig, in wie weit er die Anfrage unterstützen kann und will. Im ungünstigsten Fall muss der Anfrage-Verarbeiter alle Datensätze materialisieren um die geforderten Selektionen und Projektionen selbst durchzuführen. Der vierte Dienst, den ein Wrapper bieten muss, ist die Anfrageverarbeitung. Er muss den Teil der Arbeit, ausführen, von dem er in der Planungsphase behauptete, er könne ihn ausführen. Der Wrapper ist hier also ein intelligenter Partner des eigendlichen Systems und nicht nur ein Umsetzer von einem in ein anders Datenformat.

4.5 SQL/MED – Verwalten externer Daten

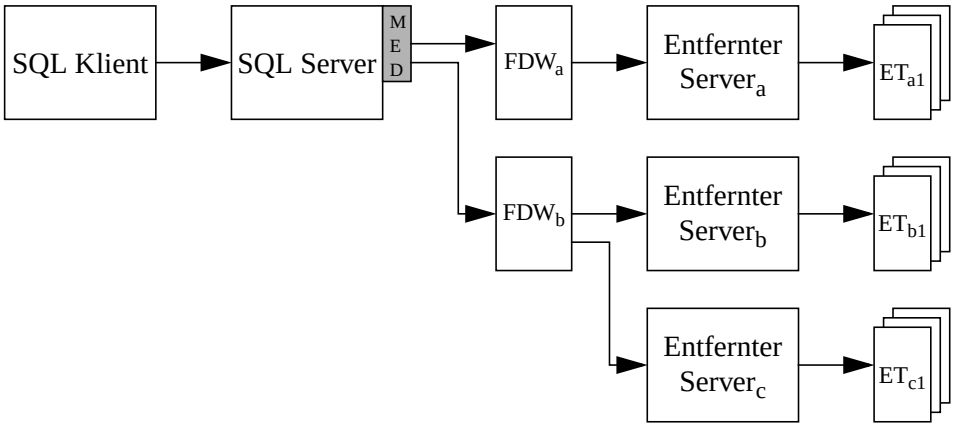
Mit dem ISO-Standard SQL/MED [ISO9075-9] wurde der Versuch unternommen die Schnittstelle zur Anbindung externer Daten an ein OR-DBS zu normen. Der Standard befasst sich mit der Verwaltung von sog. LOBs („*Large OBjects*“, der Verwaltung großer Objekte, die nicht unbedingt innerhalb der DB gehalten werden müssen oder können), Embedded SQL (dem Einbetten von SQL Anweisungen in Wirtssprachen, wie etwa C, COBOL, FORTRAN oder PASCAL), sowie der Spezifikation eines Call-Level Interfaces. Der in der Norm definierte Datentyp DATALINK, stellt zwar folgende wünschenswerte Eigenschaften für die Verwaltung externer Dateien in DBS bereit:

1. ***Referentielle Integrität*** – Eine durch eine DB referenzierte externe Datei kann weder umbenannt noch gelöscht werden.
2. ***Transaktionskonsistenz*** – Änderungen, die sowohl die DB als auch externe Dateien betreffen werden in einer Transaktion ausgeführt.
3. ***Sicherheit –*** Dateizugriffe werden über DB privilegien geregelt.
4. ***Koordiniertes Anlegen von Sicherungskopien bzw. Zurückspielen selbiger*** – Das DBMS ist verantwortlich für das synchrone Anlegen bzw. Zurückspielen von Sicherungskopien der intern verwalteten Daten mit extern gehaltenen Dateien.

Dennoch sind diese im Kontext von Web-basierten Informatinssystemen leider nicht realisierbar: Server im Internet (die einem nicht selbst unterstehen) stellen normalerweise keine der vier Eigenschaften bereit.

Ein weiteres Konzept, der in dieser Norm definiert wird, ist die Verwendung von sog. „*Foreign Data Wrappern*“ (FDW).

Abbildung 18 SQL/MED Schnittstellen



Bei dem SQL Klienten handelt es sich um die Kommunikationsschnittstelle des Anwenders zum System. Dieser Klient greift über standard SQL-Anfragen auf seinen SQL-Server zu.

Dieser verwendet nun den Standard ISO 9075 SQL/MED Teil 9 zur Kommunikation mit den Wrappern (FDW). Jeder Wrapper dient zur Kommunikation mit mindestens einem entfernten Server. Die entfernten Server ihrerseits verwalten mindestens eine Datenquelle. Da SQL auf dem relationalen Datenmodell aufsetzt, handelt es sich um nicht direkt von Server verwaltetet sondern entfernte Tabellen (ET). Liegen die Daten in anderen Formaten vor, muss der Wrapper die notwendige Schematransformation vornehmen.

Der SQL-Server hat zwei Möglichkeiten eingehende Anfragen zu verarbeiten:

- ***Decomposition*** – Der SQL-Server analysiert die vom Klienten gestellte SQL-Anfrage und reicht die Zugriffe auf entfernte Tabellen an den jeweils dafür verantwortlichen Wrapper über einen sog. `InitRequest` weiter.
- ***Pass-Through*** – Hier wird nur auf Daten zugegriffen, die von genau einem Wrapper bereitgestellt werden. Der SQL-Server reicht in diesem Fall die Anfrage mit einem sog. `TransmitRequest` direkt in der Sprache des entfernten Servers an diesen durch.

Man kann die Gruppe der Routinen zum Zugriff auf entfernte Daten mittels Wrapper in Kategorien unterteilen:

- ***Handle Routinen*** – Routinen zum Erhalt, zur Bearbeitung oder zur Freigabe einer Verarbeitungs-Marke (*Execution Handle*), etwa `GetExecutionHandle`, `FreeExecutionHandle`.
- ***Initialisierungsroutinen*** – Routinen zum Allozieren verschiedener Elemente (etwa `AllocDescriptor`, `AllocWrapperEnv`) zur Kontaktaufname mit dem Server (`ConnectServer`), zur Vorbereitung einer Anfrage (`InitRequest`) und zur Abfrage verschiedener Parameter (etwa `GetOps`, `GetServerName`, `GetServerType`, `GetWrapperName`).
- ***Zugriffsroutinen*** – Routinen zum Aufbau/Schließen einer Verbindung (`Open`, `Close`, `ReOpen`), Durchreichen einer Anfrage (`TransmitRequest`), Übertragen von Tupeln (`Iterate`), erstellen von Statistiken (`GetStatistic`).

Der komplette Datenzugriff geschieht hier auf SQL-Basis. Da zur Kommunikation mit dem Wrapper bzw. dem entferntem Server spezielle Zugriffsroutinen verwendet werden, geschieht diese Verarbeitung nicht mehr komplett transparent für den Anwender. Es ist jedoch nicht ersichtlich, ob auf der Gegenseite ein SQL-fähiger Server arbeitet, oder ob der Wrapper hier eine andere Datenhaltung kaschiert. SQL/MED arbeitet also auf einem rein relationalen Datenmodell.

4.6 XML:DB – ein XML basiertes Datenbank-Framework

XML:DB [XML:DB01] ist eine Organisation, die ursprünglich von der dbXML Group L.L.C. SMB GmbH und der OpenHealthCare Group initiiert wurde und sich vier zentralen Zielen verschieben hat:

- Entwicklung von Technik-Spezifikationen für die Verwaltung von Daten in XML-Datenbanken.

- Erstellen von Referenzimplementierungen der Spezifikationen unter einer Open Source Lizenz.
- Einrichten einer Gemeinde von XML-Datenbank-Verkäufern und -Anwendern, in der diese Fragen besprechen und Informationen austauschen können, sodass beide Seiten mehr über die XML-Datenbanktechnologie und sie nutzende Anwendungen lernen können.
- Die Ideologie der XML-Datenbankprodukte und -techniken predigen, um die Sichtbarkeit von XML-Datenbanken auf den Märkten zu erhöhen.

Die Organisation hat mittlerweile eine recht ausgereifte Spezifikation zum Umgang mit XML-Datenbanken entwickelt, die folgenden Forderungen genügt:

- ***Sprachunabhängigkeit*** – Die Schnittstelle ist neutral verfasst, sie *darf nicht* exklusiv eine Programmiersprache bedient (Anm: Die Referenzimplementierung wurde in JAVA vorgenommen).
- ***Text-Schnittstelle*** – Die Schnittstelle *muss* das Ergebnis einer Anfrage an die Datenbank als XML-Textdokument liefern können.
- ***XML-Programmierschnittstelle*** – Die Schnittstelle *sollte* je eine SAX- und eine DOM-basierte Repräsentation der Anfrage anbieten.

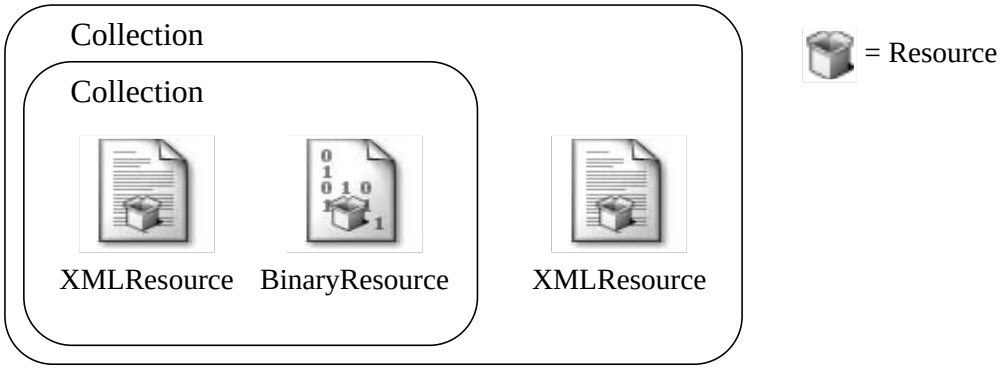
An die Programmierschnittstelle wurden weitere Forderungen gestellt:

- ***Einfügen*** – Die Schnittstelle *muss* das Einfügen neuer Dokumente ermöglichen, wobei diesem dabei zusätzlich eine eindeutige ID gegeben werden *kann*.
- ***Suchen*** – Die Schnittstelle *muss* eine Anfragemöglichkeit für in ihr gespeicherte Dokumente vorsehen, dafür *muss* jedes Dokument über eine eindeutige ID verfügen.
- ***Löschen*** – Die Schnittstelle *muss* es ermöglichen Dokumente zu löschen.
- ***Aktualisieren*** – Die Schnittstelle *muss* Aktualisierungsmöglichkeiten für die Dokumente bieten.
- ***Anfrage-Verarbeitung*** – Die Schnittstelle *muss* es ermöglichen Anfragen zu verarbeiten, *darf* dafür aber *nicht* eine spezielle Anfragesprache erwarten.
- ***Ansammlungen*** – Die Schnittstelle *muss* in der Lage sein, mit verschachtelten Ansammlungen (Abbildung 19) arbeiten zu können.
- ***Transaktionen*** – Die Schnittstelle *kann* transaktionale Verarbeitung anbieten.

Grundgedanke des XML:DB-Ansatzes ist die Speicherung der Daten in einer Art Behälter, der Auskunft darüber gibt, was sich in ihm befindet. Diese Rohdatencontainer werden Ressourcen genannt. Auf diesen Rohdatenbehältern existieren nur recht grundlegende Methoden zum Setzen und Auslesen des Inhalts, sowie zum Feststellen, um welchen Ressourcentyp es sich handelt.

Da diese Rohdaten nicht sinnvoll verarbeitet werden können, muss für jede Ressourcenart eine entsprechende Implementierung spezieller, auf sie zugeschnittener Methoden erfolgen.

Abbildung 19 XML:DB Ansicht



In der Standard-API existieren zwei Ressourcentypen:

- ***BinaryResource*** – eine Ressource zum Verwalten von Binärdaten (etwa Bilder), die neben Text ebenfalls in der Datenbank gehalten werden sollen.
- ***XMLResource*** – eine Ressource zum Verwalten der XML-Daten der Datenbank. Neben den oben bereits beschriebenen Methoden gibt es für diesen Datentyp weitere, speziell auf die Verarbeitung von XML-Daten zugeschnittene Methoden. So gibt es einerseits die Möglichkeit, sich die Daten als DOM Objekt geben zu lassen bzw. wieder zu speichern; andererseits existieren auch Methoden, um nach Angabe eines *Content-Handlers* eine ereignisbasierte Verarbeitung durch einen SAX-Parser vornehmen zu lassen.

Die in einer Datenbank gehaltenen Ressourcen werden in Ansammlungen (engl. *Collection*) gruppiert. Es ist möglich, dass die Datenbank die Ansammlungen als eine Hierarchie von Vater-Sohn Ansammlungen bereitstellt (Abbildung 19).

Um auf allen Ressourcen einer Ansammlung Aktionen ausführen zu können, existiert die Schnittstelle `ResourceIterator`. Mit der Methode `nextResource` können in einer Schleife alle Ressourcen abgearbeitet werden.

Die Basisfunktionalität kann erweitert werden. Hierfür stehen Dienste (engl. *Service*) zur Verfügung. Der einfachste Vertreter ist der `ColectionManagementService`. Er bietet nur Methoden zum Verwalten von Ansammlungen.

Der `TransactionService` bietet die Möglichkeit eine Reihe von Operationen als Transaktion zu kapseln, mit den beiden möglichen Ausgängen „commit“ und „rollback“. Aber auch die Anfragemöglichkeiten werden – wenn überhaupt – durch eigene Dienste bereitgestellt. So ermöglicht es der `XPathQueryService`, XPath Anfragen über Ansammlungen und die darin enthaltenen Ressourcen zu stellen. Mit dem `XUptdateQueryService` wird es möglich, XUpdate Anfragen auf einzelne XML-Ressourcen anzuwenden.

Kapitel 5 Erarbeiten eines Wrapper-Frameworks

Die grundlegenden Modelle und Konzepte, die in den vorangehenden Kapiteln eingeführt und erläutert wurden, sollen nun daraufhin untersucht werden, welche Anforderungen erfüllt sein müssen, damit eine Anwendung aus einer Menge heterogener Datenquellen die Erstellung einer *Einheitlichen Sicht* vornehmen kann. Ziel ist es, ein *Wrapper-Framework* zu erarbeiten, also eine dynamisch erweiterbare Datenaustausch-Schnittstelle, mit deren Hilfe sich Module, die eine Datenquelle bereitstellen wollen anmelden und die enthaltenen Informationen zur Verfügung stellen können.

5.1 Nichtfunktionale Anforderungen

Die vier Ziele, die sich das GARLIC Team für das Anbinden heterogener Quellen gesetzt hat, können für den Framework direkt übernommen werden:

1. Der Aufwand zur Realisierung eines einzelnen Wrappers soll so gering wie möglich sein, sodass der Anwendung möglichst einfach eine neue Datenquelle hinzugefügt werden kann. Das bringt uns zu:

Anforderung 1 *Erstellungskosten*

Die Erstellungskosten eines Wrappers sollen durch eine einheitliche Programmierschnittstelle so gering wie möglich gehalten werden.

2. Damit die Anbindung schnell erfolgen kann, muss der Wrapper als Mindestanforderung nur die Daten seiner Quelle weiterreichen können. Weitere Eigenschaften und Möglichkeiten der Datenquelle, müssen der Anwendung nicht unbedingt bekannt gemacht werden.

Anforderung 2 *Erweiterbarkeit*

Der Wrapper muss nur einen Minimalumfang an Dienstleistungen bereitstellen. Sie sollen jedoch auch weiter auf die Bedürfnisse der Anwendung zugeschnitten werden können.

3. Damit der laufende Betrieb nicht beeinträchtigt wird, muss eine Möglichkeit vorgesehen werden, dass zur Laufzeit der Anwendung weitere Quellen angebunden werden können, bzw. angemessen auf den (möglicherweise auch nur temporären) Wegfall einer Quelle reagiert werden kann:

Anforderung 3 *Dynamische Anpassbarkeit*

Es soll möglich sein im laufenden Betrieb neue Datenquellen hinzuzufügen, bzw. sinnvoll auf den Wegfall einer Quelle zu reagieren.

4. Falls die Datenquelle Möglichkeiten bietet, die der Anwendung helfen können gewünschte Anfragen effizienter zu gestalten, so sollten diese der Anwendung bekannt gegeben und ein Mechanismus vorgesehen werden, mit dessen Hilfe diese Möglichkeiten genutzt werden können:

Anforderung 4 *Anfrageoptimierung*

Der Wrapper soll die Möglichkeiten seiner Datenquelle in den Prozess der Anfrageverarbeitung und Optimierung mit einbringen können.

5.2 Eigenschaften der Quellen

Es gibt zahlreiche Eigenschaften, anhand derer die verschiedenen Datenquellen in Kategorien unterteilt werden können. Die wichtigsten sollen uns weitere Anforderungen liefern.

5.2.1 Datenmodell

Den Quellen liegen unterschiedliche Datenmodelle zu Grunde, die Anwendung soll jedoch auf einem einheitlichen Modell arbeiten. Daher müssen die Wrapper eine Konvertierung in ein Standardmodell vornehmen. Dieser Schritt ist notwendig, wenn eine Integration der Quellen vorgenommen werden soll, auf die sich die weitere Verarbeitung stützt:

Anforderung 5 *Datenmodell*

Der Wrapper muss die Daten seiner Quelle in das vorher festgelegte Standard-Format konvertieren, bevor er sie der Applikation anbietet, falls die Quelle nicht bereits im Standardformat vorliegt.

5.2.2 Kodierung

Wie bereits im Kapitel Datenmodelle (Kodierung der Dokumente) erwähnt, existieren im Web heute immer noch etliche unterschiedliche Kodierungen. Da wir alle von unseren Wrappern angebotenen Quellen zu einer *Einheitlichen Sicht* vereinigen wollen, müssen wir natürlich dafür sorgen, dass die verschiedenen Ausgangskodierungen in ein einheitliches Format gebracht werden. Wir fordern also:

Anforderung 6 *Datenkodierung*

Die Ausgangsdatenkodierung muss vom Wrapper in eine vorher festgelegte Standardkodierung gewandelt werden (sofern diese von der Ausgangskodierung abweicht), sodass die Applikation daraus eine *Einheitliche Sicht* machen kann.

5.3 Dynamische Aspekte der Quellen

5.3.1 Zugriffsmechanismus

Datenquellen unterscheiden sich in der Art der Zugriffsmöglichkeiten. Einige Quellen (etwa das Dateisystem oder Web-Server) stellt nur komplette Dokumente bereit, die als Ganzes gelesen werden müssen. Bei anderen Quellen werden die Daten durch ein System (etwa ein DBS) verwaltet, das die Anwendung unterstützen und entlasten kann, etwa durch das Anbieten von Anfrage- oder Auswahlmöglichkeiten. Auch das wollen wir uns zu Nutze machen:

Anforderung 7 *Zugriffsmechanismus*

Der Wrapper muss der Applikation mitteilen, welche Zugriffsmechanismen auf die Datenquellen er bereitstellt, bzw. in welchen Mengeneinheiten der die Daten anbietet (komplett materialisiert oder Fragmente des Komplettbestandes).

5.3.2 Kommunikationsprotokolle

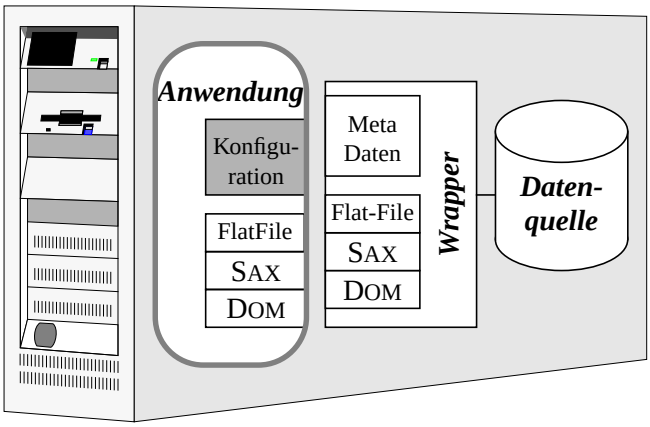
Je nachdem wo sich Wrapper und Datenquelle bezüglich der Anwendung befinden, müssen andere Mechanismen zum Anbinden der Datenquellen betrachtet werden. Handelt es sich um eine lokale Quelle, kann eine enge Bindung, etwa durch einen Funktionsaufruf, an die Anwen-

dung vorgenommen werden. Entfernte Quellen müssen hingegen über ein Netzwerkprotokoll, etwa einen „*Remote Procedure Call*“ (RPC), angesprochen werden. Es können folgende Fälle unterschieden werden:

1. ***Wrapper und Datenquelle lokal:***

   In diesem Fall befinden sich Sowohl Anwendung als auch die Quelle auf dem selben Rechner. Es kann also problemlos eine enge Bindung (etwa über Funktionsaufrufe) vorgenommen – werden (☞ Abbildung 20).

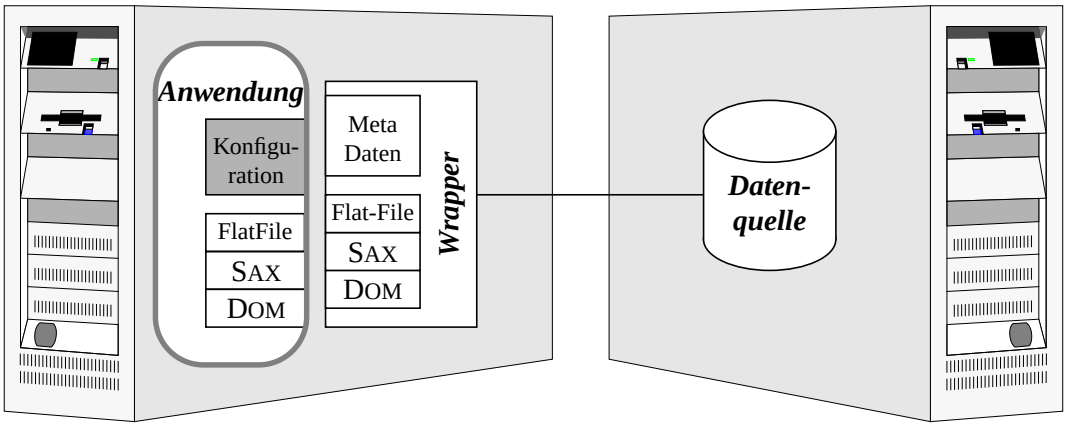
Abbildung 20 Wrapper und Datenquelle lokal



2. ***Wrapper lokal – Datenquelle auf entferntem Rechner:***

   In diesem Fall befinden sich die Daten auf einem entfernten Rechner. Der Wrapper befindet sich auf der Maschine, auf der auch die Anwendung läuft (☞ Abbildung 21).

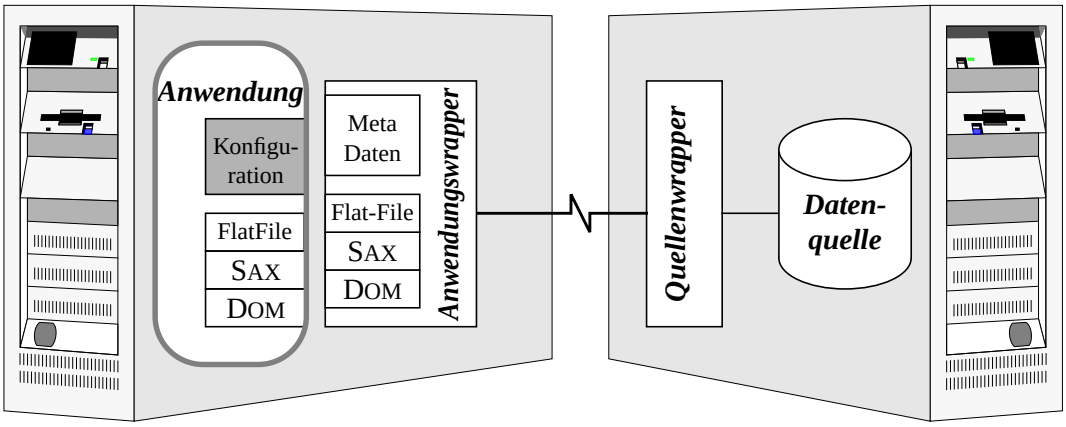
Abbildung 21 Wrapper lokal – Datenquelle auf entferntem Rechner



3. ***Wrapper teils lokal, teils bei der Datenquelle:***

   In diesem Fall findet die Kommunikation nicht nur zwischen Wrapper und Datenquelle statt, sondern auch der Wrapper ist geteilt. Der eine Teil bietet auf Anwendungsseite die Protokollschnittstellen zur Anwendung hin, der andere Teil kommuniziert auf dem Rechner der Datenquelle lokal mit der Quelle (☞ Abbildung 22).

Abbildung 22 Wrapper teils lokal, teils bei der Datenquelle

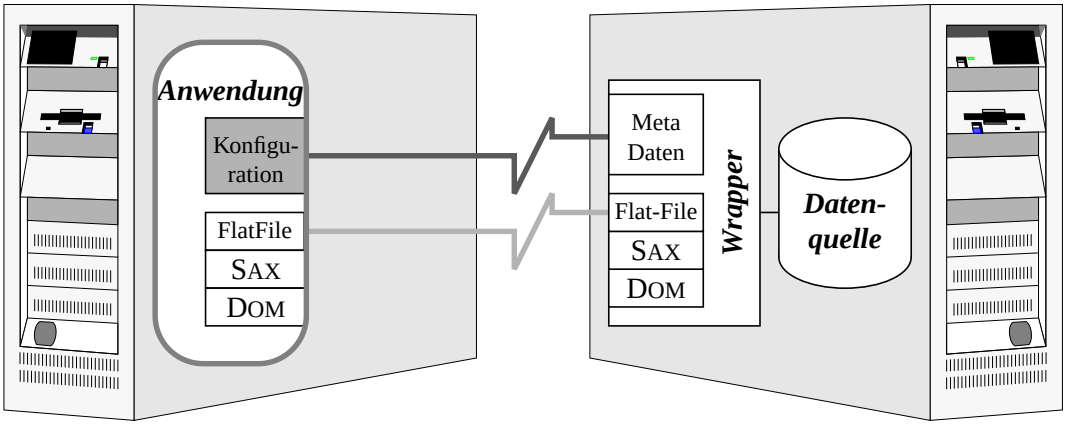


Die bisherigen Fälle sind jedoch aus Anwendungssicht identisch. Daher können wir sie für die Betrachtung der Anwendungs- ↔ Wrapper-Schnittstelle als einen auffassen.

4. ***Wrapper komplett auf dem Datenquellenrechner:***

   Wenn sich der Wrapper komplett auf dem Server-Rechner, also dem Rechner mit der Datenquelle befindet, muss alle Kommunikation über Netzwerkprotokolle laufen (☞ Abbildung 23). Daher müssen entsprechende Mechanismen vorgesehen werden.

Abbildung 23 Wrapper komplett bei der Datenquelle



Wir können nun folgende Anforderungen aus den verschiedenen Szenarien ableiten:

Anforderung 8 *Protokolle*

Je nach Lage des Wrappers aus sicht der Anwendung kann eine enge Bindung vorgenommen werden oder muss mittels Netzwerkprotokollen die Kommunikation erfolgen. Dies muss der Anwendung mitgeteilt werden.

5.3.3 Änderungsmöglichkeit

Ein weiterer wichtiger Aspekt ist die Möglichkeit, der Veränderung von Quelldaten. Im günstigsten Fall untersteht die Quelle dem Wrapper. Somit können die enthaltenen Daten erweitert, aktualisiert und geändert werden. Der Wrapper muss also nur ein Protokoll mit der Anwendung vereinbaren, wie die Änderungswünsche zur Quelle gelangen können.

Ist die Quelle jedoch nur ein Informationsanbieter, steht die Quelle nur lesend zur Verfügung. Ein Beispiel sind Nachrichtenanbieter, die es Interessenten ermöglichen Nachrichten zu einem bestimten Themengebiet (sog. *Channels*) mittels RSS *„Rich Site Summary“* abzurufen. Diese Daten können ohne Benachrichtigung des Wrappers gelöscht oder geändert werden. Auch in diesem Fall sollte die Anwendung sinnvoll reagieren und nicht in einen undefinierten Zustand gelangen:

Anforderung 9 *Änderungsmöglichkeit*

Der Wrapper muss der Applikation mitteilen, ob der von ihm verwalteten Datenbestand erweitert, geändert oder gelöscht werden kann. Ferner sollte mitgeteilt werden, ob der Datenbestand „statisch“ ist oder ohne Vorwarnung Änderungen vorkommen können.

5.3.4 Sicherheit

Bietet die Quelle bereits einen Sicherheitsmechanismus, sodass nicht alle Anwender auf die kompletten Daten zugreifen dürfen ohne sich als berechtigt zu authentifizieren (etwa bei einer Datenbank), so soll auch dieser Mechanismus der Anwendung angeboten werden:

Anforderung 10 *Sicherheitsmechanismen*

Der Wrapper soll der Anwendung mitteilen dass die Quelle Sicherheitsmechanismen (etwa Authentifizierung) unterstützt, bzw. deren Verwendung erwartet und ihr Möglichkeiten bieten, diese Mechanismen zu nutzen.

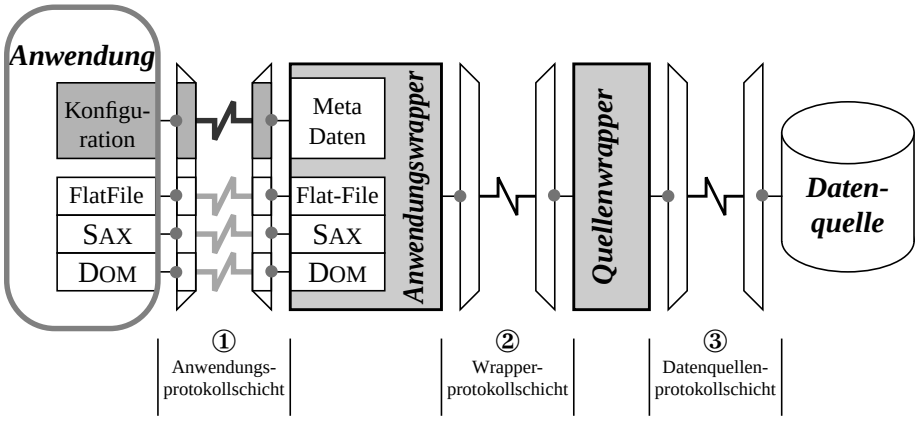
Kapitel 6 Konkretisierung

In diesem Kapitel sollen die notwendigen Wrapperschnittstellen betrachtet und festgelegt werden, wie an ihnen jeweils zu verfahren ist.

6.1 Die Protokollschichten

Aus den unterschiedlichen Szenarien, die im Kapitel Erarbeiten eines Wrapper-Frameworks (Kommunikationsprotokolle) in Abbildung 20 bis Abbildung 23 gezeigt wurden, lässt sich ein universelles Wrapper-Modell erstellen, in dem in die Kommunikation zwischen Anwendung und Datenquelle mittels Wrapper drei Protokollschichten eingezogen werden. Der Sinn dieser Maßnahme soll im Folgenden etwas genauer betrachtet werden.

Abbildung 24 Protokollschichten



Der Wrapper wird hierfür in zwei Teile gesplittet:

1. Der ***Anwendungswrapper*** ist der Teil, der die Kommunikation mit der Anwendung koordiniert. Er stellt die unterschiedlichen Verarbeitungsschnittstellen bereit und handelt diese über die Metadatenschnittstelle mit der Anwendung aus.
2. Der ***Quellenwrapper*** ist der explizit auf die jeweilige Quelle zugeschnittene Teil. Da die Möglichkeit bestehen soll unterschiedlichste Quelltypen anzusprechen, ist eine allgemeine Aussage über die Kommunikationsmethode nicht möglich.

Somit entstehen drei Schnittstellen, an denen die einzelnen Kommunikationspartner über ein adäquates Protokoll miteinander kommunizieren müssen.

1. Die ***Anwendungsprotokollschicht*** ① dient zur Kommunikation zwischen Anwendung und Anwendungswrapper.
2. Die ***Wrapperprotokollschicht*** ② wurde nur eingezogen, um die Möglichkeit zu haben, einen „verteilten Wrapper“ bieten zu können, der teilweise auf dem Anwendungsrechner, teilweise auf dem Datenquellenrechner implementiert ist.
3. Über die ***Datenquellenprotokollschicht*** ③ wird die eigentliche Kommunikation mit der Quelle realisiert. Da die Menge der Quelltypen zu groß ist, um sie sinnvoll zu normen ohne die Kommunikation unnötig einzuschränken, wird sie in den fogenden Betrachtungen keine große Rolle mehr spielen.

Für das eigentliche Framework müssen die zu den Knoten führenden Linien der jeweiligen Protokollschicht nun spezifiziert werden. Dies geschieht durch das festlegen von Funktions-, bzw. Methodenaufrufen die die Möglichkeiten der jeweiligen Schnittstelle netzwerktransparent machen. Somit muss erst für die Implementierung festgelegt werden, ob die Funktionalität des entsprechenden Wrappers direkt gekoppelt (also eine enge Bindung hergestellt) oder durch Binden an ein netzwerkfähiges Äquivalent lose gekoppelt werden soll.

Die mit dieser Maßnahme gewonnenen klaren Protokollschichten, ermöglichen es zur Laufzeit besser auf Bedürfnisse der Anwendung eingehen zu können. Wird etwa eine Quelle, die zu Beginn auf dem selben Rechner wie die Anwendung lief auf einen anderen Rechner verschoben, muss nicht der gesamte Wrapper neu implementiert werden. In diesem Fall würde es genügen, den Wrapper in der Wrapperprotokollschicht „aufzutrennen“ und es müssten nur die dort vorgesehenen Funktionen bzw. Methoden mit Netzwerkunterstützung neu implementiert werden.

Der Framework wird somit besser skalierbar. Der minimale Mehraufwand für den Wrapper-Programmier seht in keinem Verhältnis zur gewonnenen Flexibilität.

Im Folgenden sollen nun die einzelnen Protokollschichten genauer betrachtet werden.

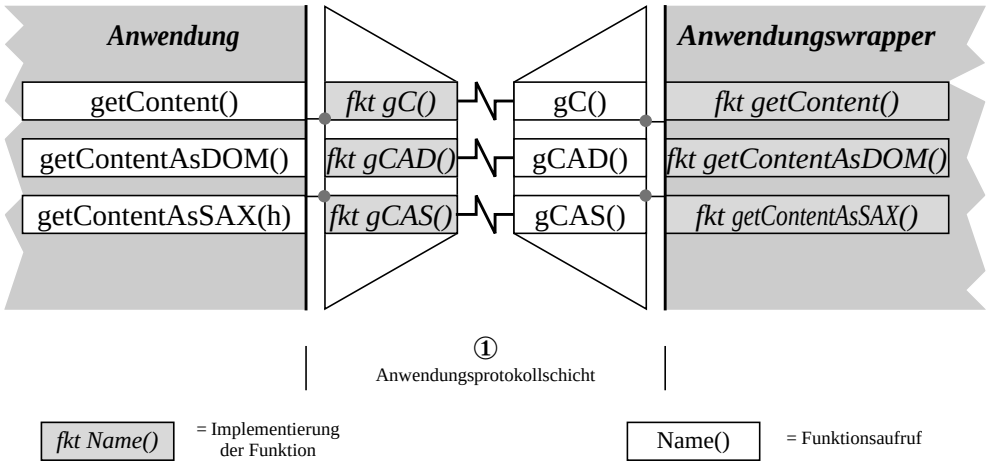
6.1.1 Die Anwendungsprotokollschicht

In dieser Schicht müssen für die unterschiedlichen Zugriffsschnittstellen sowie für den Metadatenaustausch entsprechende Funktions- bzw. Methodenaufrufe vorhanden sein. Weiterhin müssen Möglichkeiten vorgesehen werden, Anfragen und Aktualisierungen zur Quelle zu propagieren, sofern diese das unterstützt. Auf Anwendungsseite befindet sich ein Repository, das die XML-Dokumente und Dokumentfragmente entgegen nimmt.

In Abbildung Abbildung 25 sind die Schnittstellen für FlatFile, SAX und DOM dargestellt. `getContent()` dient zum übergeben des Dokuments als ganzes ohne Strukturierung. Mittels `getContentAsDOM()` wird ein DOM-Baum erstellt und der Anwendung übergeben. Durch den Aufruf von `getContentAsSAX(h)` wir der ContentHandler h als Parser der SAX Ereignisse festgelegt.

Die jeweiligen Funktions- bzw. Methodenaufrufe der Anwendung erhalten in der Anwendungsprotokollschicht eine entsprechende Implementierung. Bei einer losen Kopplung sind diese für die Netzwerkkommunikation verantwortlich, d.h. sie rufen auf dem Rechner, der den Anwendungswrapper beherbergt wieder eine entsprechende Funktion bzw. Methode auf. Bei enger Bin-

Abbildung 25 Anwendungsprotokollschicht



dung schrumpft die Anwendungsprotokollschicht auf die Identität, könnte somit in der Praxis wegfallen.

6.1.2 Die Wrapperprotokollschicht

Im günstigsten Fall kann von den unterschiedlichen Zugriffsmechanismen sowie Quelleneigenschaften vollkommen abstrahiert werden. Es muss einfach nur eine Möglichkeit für den bidirektionalen Datenaustausch vorgesehen werden.

Abbildung 26 Wrapperprotokollschicht

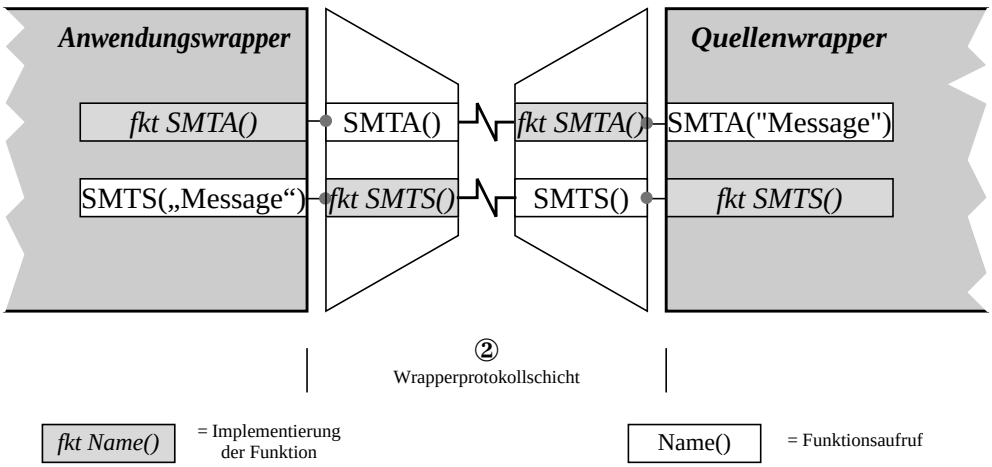


Abbildung 25 zeigt, wie dies mittels `SendMessageToSource`, und `SendMessageToApp` geschehen kann.

Konsequente Verwendung dieser Schnittstelle selbst in monolithischen Wrappern ermöglicht es, die Migration der Datenquelle von Anwendungsrechner auf einen anderen Rechner ohne komplette Reimplementierung. Es müssen lediglich die Funktionen bzw. Methoden `SendMessageToSource`, und `SendMessageToApp` durch netzwerkfähige Komponenten ersetzt werden.

Da wir unser Framework stark an das XML:DB-Framework anlehnen wollen (siehe Kapitel 4.6 auf Seite 28), existiert kein Platz, wo sich ein solcher Schnitt sinnvoll anbringen lassen würde. Eine Möglichkeit dies dennoch zu tun, wäre die Trennung in der Schnittstelle `Collection`. Dafür müssten alle Methoden, die auf Ressourcen zugreifen so implementiert werden, dass sie leicht durch netzwerktaugliche Implementierungen austauschbar sind.

6.1.3 Die Quellenprotokollschicht

Die Menge unterschiedlicher Quellen führt dazu, dass es nicht sinnvoll erscheint, hier eine Standardisierung vorzunehmen, da sonst die Kommunikation mit der Quelle unnötig eingeschränkt würde.

Da die Implementierung auf XML als Kommunikationsprotokoll, sowie als Datenformat arbeiten soll, sind XML-Quellen (bzw. XML-Schnittstellen der Quellen) natürlich hilfreich.

6.2 Abgrenzung: SHARX <-> XML:DB

Nach eingehender Betrachtung der API, die die XML:DB Initiative erarbeitet hat, stellte sich heraus, dass sie den Anforderungen, aus Kapitel 5 schon recht nahe kommt. Sie soll daher als Gerüst, für die Wrapper-API von SHARX dienen. Dennoch kommen durch die Forderung bei den heterogenen Quellen auch reine Bereitsteller von Daten zuzulassen, die keine Möglichkeiten der Einflussnahme bieten leicht andere Forderungen zu stande. Diese sollen nun analog zu den XML:DB Forderungen aus Kapitel 4.6 aufgelistet werden. Durch die Verwendung der XML:DB Schnittstelle und Erweiterung durch eine Metadatenschnittstelle, bleibt die ***Sprachenunabhängigkeit*** der API erhalten.

Auch bei SHARX muss eine ***Text-Schnittstelle*** bereitgestellt werden. Diese soll jedoch „intelligenter“ sein, als die von der XML:DB vorgesehene Variante. Wenn der Wrapper weiß, dass die Daten, die er in einem Dokument anbietet zu umfangreich sind, um sinnvoll komplett materialisiert zu werden, lässt er das Attribut „materialisierbar“ weg. Sollte die Anwendung nun dennoch versuchen, das komplette Dokument zu erfragen, kann der Wrapper mit einer Exception darauf reagieren. Um dennoch die kompletten verwalteten Daten anbieten zu können, muss der Wrapper entweder eine SAX-Schnittstelle bieten oder bei der Anfrageverarbeitung mithelfen, um die wirklich benötigten Teile zu suchen und nur diese zu propagieren. Die SHARX-Wrapper sollten ebenfalls die ***XML-Programmierschnittstellen*** für DOM und SAX anbieten.

Aus der Tatsache, dass die Wrapper auch Quellen bereitstellen sollen, auf die sie nur lesenden Zugriff haben – etwa über Web-Server bereitgestellte Dokumente – müssen Forderungen resultieren, die von denen der XML:DB deutlich abweichen:

- ***Einfügen*** – Die Schnittstelle *soll* das Einfügen neuer Dokumente ermöglichen, wenn die Quelle es erlaubt. Beim Einfügen *soll* es möglich sein, eine eindeutige ID zu erzeugen.

- ***Suchen*** – Die Schnittstelle *muss* eine Möglichkeit vorsehen, zu erfahren, welche Dokumente vom Wrapper verwaltet werden.
- ***Löschen*** – Die Schnittestelle *soll* das löschen von Dokumenten ermöglichen, wenn die Quelle es erlaubt.
- ***Aktualisieren*** – Die Schnittstelle *soll* das aktualisieren von Dokumenten ermöglichen, wenn die Quelle es erlaubt.
- ***Anfrage-Verarbeitung*** – Die Schnittstelle *sollte* es ermöglichen, Anfragen zu verarbeiten, um damit die Anwendung entlasten und unterstützen zu können.
- ***Ansammlungen*** – Die Schnittstelle *muss* in der Lage sein, mit verschachtelten Ansammlungen arbeiten zu können.
- ***Transaktionen*** – Die Schnittstelle kann transaktionale Verarbeitung anbieten.

Es ist wichtig der Anwendung mitzuteilen, welche der Aktionen die Quelle nun wirklich unterstützt, da die API für jeden Wrapper alle Routinen vorsieht. Wir müssen also das XML:DB-Konzept erweitern um eine Metadatenschnittstelle. Über sie kommuniziert der Wrapper mit der Anwendung und propagiert zu Beginn seine Fähigkeiten. Ändern sich im laufenden Betrieb Eigenschaften, so hat er die Möglichkeit das auf dem selben Wege der Anwendung mitzuteilen.

Da die Kommunikation mittels XML stattfinden soll musste eine DTD erarbeitet werden, um die Kommunikation festzulegen (siehe Anhang B.1). Es ist somit sichergestellt, dass der Wrapper der Anwendung alles notwendige mitteilen kann.

Weiterhin sind in unserer API alle Aufrufe *transitiv*. Das bedeutet ein Aufruf der Methode `listResources` angewandt auf die aktuelle Ansammlung liefert daher auch die Ressourcen der Sohn- und Enkel-Ansammlungen – sofern diese existieren – und so fort. Die Anwendung muss sich somit nicht durch alle Ansammlungsebenen bewegen, um an alle Ressourcen zu gelangen, kann dies aber tun, um die Dokumentenmenge, auf der sie arbeiten möchte, einzuschränken.

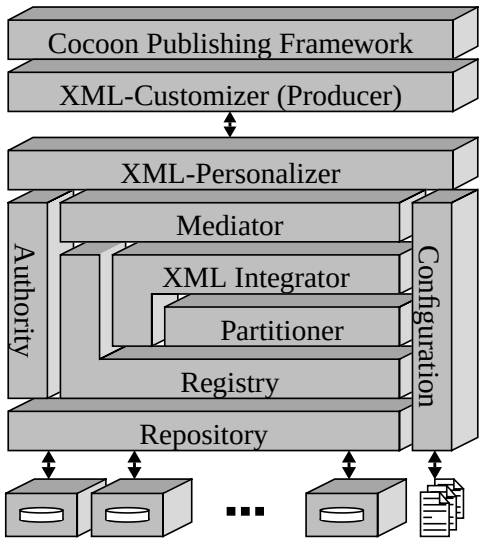
Kapitel 7 Implementierung

Die Anforderungen des vorangehenden Kapitels sollen nun umgesetzt werden. Unter Anderem werden auch grundlegende Implementierungsentscheidungen getroffen. Teilweise sind diese nicht direkt aus den Anforderungen ableitbar, jedoch notwendig um das Framework realisieren zu können.

7.1 SHARX – ein Web-Site-Management System

Wie bereits in der Einleitung erwähnt (☞ 1.2) ist SHARX ein Projekt zum datenorientierten, XML-basierten Web-Site-Management in hochpersonalisiertem Umfeld. Abbildung 27 zeigt schematisch den Aufbau des Systems.

Abbildung 27 SHARX – Realisierung



Die Basis des Systems bilden heterogene Datenquellen. Da diese für SHARX nur als austauschbare Module wahrgenommen werden sollen, sind sie durch Wrapper gekapselt. Um es dem System zu ermöglichen, effizient mit den Ausgangsdaten zu arbeiten, muss der Wrapper die

Eigenschaften und Möglichkeiten seiner Quelle über die Konfigurationsschnittstelle bereit stellen. Die beiden Partner – Anwendung und Wrapper – handeln die Zugriffsmöglichkeiten aus. Dabei ist auch auf die Authentifizierung der Anwender, bzw. zur Authorisierung der Datenzugriffe zu achten.

Wir wollen nun darauf eingehen, wie die Anforderungen aus Kapitel 5 sinnvoll in das Gesamtkonzept integriert werden können.

7.2 Grundlegende Implementierungs-Aspekte

Zu Beginn müssen einige grundsätzliche Entscheidungen getroffen werden:

7.2.1 Datenmodell

Da die *Einheitliche Sicht*, zu der die Dokumente der einzelnen Quellen vereinigt werden sollen ein XML Dokument sein soll, bietet es sich an, dieses als Standardformat zu wählen und dafür Sorge zu tragen, dass die Wrapper alle XML-Dokumente, bzw -Fragmente an das Repository propagieren. Somit wäre **Anforderung 5** (Datenmodell) erfüllt.

7.2.2 Kodierung

Um sicherzustellen, dass beim Vereinigen der Daten aus den einzelnen Quellen eine sinnvolle *Einheitliche Sicht* erreicht wird, muss ein Format gewählt werden, was die meisten, wenn nicht sogar alle Kodierungen der Quelldokumente beinhaltet. Hierfür eignet sich UTF-8. Daher müssen die Dokumente (bzw. Dokumentfragmente), die die Wrapper zum Erfüllen von **Anforderung 6** (Datenkodierung) an die Anwendung propagieren UTF-8-kodiert sein.

7.2.3 Programmiersprache

Um bei der Realisierung plattformunabhängig zu bleiben bietet es sich an, als Programmiersprache Java zu verwenden.

Dies erscheint sinnvoll, da das Framework stark an die XML:DB API („*Application Programming Interface*“) [XML:DB00] angelehnt werden soll, ähnlich wie etwa die dbXML Implementierung [Sta01]. Dieser Schritt erscheint sinnvoll, da die Anbindung von XML basierten Datenquellen dadurch wesentlich erleichtert wird.

Ein weiterer Vorteil dieser Wahl besteht in der Tatsache, dass unter Java alle Text-Operationen bereits mit Unicode geschehen, man also auch unabhängig von 8-Bit Kodierungskonventionen wird.

7.2.4 Metadaten

Die einzelnen Wrapper müssen mit der Anwendung in Kontakt treten können. Daher muss eine Konfigurationsschnittstelle seitens der Anwendung existieren, an die sich der Wrapper jeweils wenden kann, um sich bekannt zu machen, oder um mitzuteilen, dass sich an seiner Datenbasis oder seinen Fähigkeiten etwas geändert hat.

Mit dieser Entscheidung wird für **Anforderung 3** (Dynamische Anpassbarkeit) gesorgt. Auch hier soll die Kommunikation mittels UTF-8 kodierten XML-Dokumenten stattfinden.

Der Wrapper muss beim Aushandeln dieser Fähigkeiten grundlegend folgende Möglichkeiten bieten:

- Entweder das komplette Dokument ist materialisierbar. In diesem Fall genügt es eine FlatFile Übergabe anzubieten. Die Anwendung muss sich dann selbst um die Auswahl der benötigten Dokument-Fragmente kümmern,
- oder das Dokument ist nicht sinnvoll vollständig materialisierbar. In diesem Fall muss das Modul eine Verarbeitung von Dokumentteilen zulassen (etwa mittels SAX). Möglicherweise kann auch die Anfrage an die Datenquelle durchgereicht werden, sodass als Ergebnis eine materialisierbare Teilmenge erzeugt werden kann: **Anforderung 4** (Anfrageoptimierung).

Die Tatsache, dass (bei sinnvoll materialisierbaren Datenmengen) mindestens die Textschnittstelle existieren muss, zusätzlich jedoch auch DOM und SAX geboten werden können, soll unserer **Anforderung 2** (Erweiterbarkeit) genügen.

7.3 Die Kommunikationsschnittstelle

Wie bereits in Kapitel 6.2 erläutert, soll die API an die der XML:DB angelehnt werden. Durch diese Vorgabe einer klaren Schnittstelle sorgen wir für die Erfüllung der **Anforderung 1** (Erstellungskosten).

Wo bei der XML:DB die Rede von Datenbanken ist, werden wir jedoch den Begriff Datenquellen (engl. *Datasource*) verwenden, um klar zu stellen, dass wir nicht generell eine Datenbank, also ein Medium mit Änderungsmöglichkeit haben.

Da die Methoden der Schnittstelle `XMLResource` die Daten als XML bereitstellen, haben sie die Aufgabe der Schematransformation vorzunehmen. Sie müssen also die unterschiedlichen Ausgangsdatenformate in XML wandeln und die Kodierungskonvertierung vornehmen.

Das Verwalten von nichttextuellen Daten, etwa Bildern, Klängen oder Videos ist zwar im Rahmen dieser Arbeit nicht vorgesehen, doch mit der Schnittstelle `BinaryResource` ist es problemlos möglich auch diese in der jeweiligen Quelle vorhandenen Daten an das Repository anzubinden.

7.4 Die Konfigurationsschnittstelle

Der Wrapper teilt der Anwendung über die Konfigurationsschnittstelle Informationen in Form eines Metadaten-Dokuments über die von ihm verwaltete Quelle mit. Dieses Dokument genügt der DTD in Anhang B.1. Das Repository erhält somit Aussagen, ob die Dokumente dieser Quelle gelöscht, geändert oder neue hinzugefügt werden können. Der Wrapper informiert auch, über welche Mechanismen auf die Daten zugegriffen werden darf (DOM, Flat oder SAX). Sofern die Quelle zur Authentifizierung Name und Passwort benötigt, wird dies ebenfalls mitgeteilt.

Die durch die Quelle verwalteten Dokumente werden der Anwendung in einer Form einer Collection (vergleiche Abbildung 19) bekannt gegeben, sodass der Partitioner später auf diese Daten zurückgreifen kann, um seine Arbeit zu leisten.

7.5 Anwendungsbeispiele

In Anhang A werden alle API-Pakete mit einer kurzen Erklärung aufgelistet. Da es sinnvoller erscheint, die Verwendung anhand eines konkreten Beispiels zu erläutern, soll dies im Folgenden geschehen. Konkret soll die Publikationsliste einer Arbeitsgruppe verwaltet werden.

Die Verwaltung der Publikationen erfolgt im Dateisystem eines Rechners. Für jede Veröffentlichung wird ein XML-Dokument erstellt. Die Dokumente werden in einer Verzeichnishierarchie wie in Syntax 2 abgelegt und von Mitarbeitern manuell gepflegt.

Durch einen Wrapper sollen diese Daten nun einem Web-Site-Management-System zugänglich gemacht werden. Dafür muss der Wrapper die Wurzel dieser Verwaltungshierarchie kennen, um alle Dokumente finden zu können. Für die Anwendung ist es irrelevant, wie die reale Speicherhaltung aussieht, daher steht es dem Wrapper völlig frei, mit welcher Schachtelungstiefe er die Dokumente anbietet. Im konkreten Fall soll jede Verzeichnisbaumebene als eine `Collection` (vergleiche XML:DB in Kapitel 4.6) realisiert werden.

Syntax 2 *SHARX – Verzeichnisstruktur Publikationen*

```
/sharx/
  |- dtd/
  '- publications/
      |- article/
      |- book/
      |- chapter/
      |- disertation/
      |- talk/
      '- thesis/
          |- diploma/
          '- project/
```

Syntax 3 zeigt ein Beispieldokument, das – wie alle Publikationen – gegen die Publikations-DTD aus Anhang B.2 validiert werden kann.

Syntax 3 *Beispieldokument „Sie99.xml“*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE publication SYSTEM="/sharx/dtd/publication.dtd">

<publications>
  <thesis type="project“>
    <title>
      iWebDB/ED - Dateianbindung für ein integriertes
      Web-Dokumentenverwaltungssystem
    </title>
```

```
    <year>1999</year><month>10</month>

    <author>
      <name>Siegel, S.</name>
      <email href="mailto:siegel@informatik.uni-kl.de" />
      <address>
        Database and Information Systems Group,
        University of Kaiserslautern
        P.O. Box 3049,
        D-67653 Kaiserslautern
      </address>
    </author>
    <resources />
    <language>de</language>
  </thesis>
</publications>
```

In Syntax 4 ist das XML-Metadatenprotokoll eines Wrappers dargestellt, der dieses Dokument verwaltet und nun dem Repository propagiert.

Syntax 4 *Metadaten eines neuen Wrappers*

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE WRAPPERMESSAGE
SYSTEM "http://wwwagdbis.informatik.uni-kl.de/"
       "sharx/dtd/wrapper.dtd">

<wrapperMessage version="0.1" messageclass="initial">
  <wrapper>
    <name>Publications</name>

    <description>
    This datasource provides the  publication repository of
    the Database and Information Systems Group,
    University of Kaiserslautern
    </description>

    <!-- Wrapper der Dokumente bereitstellt -->
    <base href="sharx://wwwagdbis.informatik.uni-kl.de/" />

    <!-- Liefert Anfragen als FlatFile Ergebnis -->
    <query format="flat" />

    <!-- Kann Dokumente löschen -->
    <access ability="delete" />

    <!-- Kann Neue Dokumente anlegen -->
    <access ability="insert" />

    <!-- Kann Tupel-Aktualisierungen bearbeiten -->
    <access ability="tupdate" />
  </wrapper>
```

```
<service>
  <serviceName>XPathQueryService</serviceName>
  <serviceVersion>1.0</serviceVersion>
</sevice>

<collection id="sharx">
  <collection id="publications">
    <collection id="thesis">
      <collection id="project">
        <document matreialize="yes"
         id="publications/thesis/project/Sie99.xml">

          <!-- Das Dokument genügt folgender DTD        -->
          <!-- (diese Information kann durch das        -->
          <!-- Dokument überschrieben werden             -->
          <dtd href="http://wwwagdbis.informatik.uni-kl.de/“
                    "sharx/dtd/publications.dtd" />

          <size unit="byte">538</size>
        </document>
      </collection>
    </collection>
  </collection>
</collection>

</wrapperMessage>
```

Bei den folgenden Beispielen wird davon ausgegangen, dass die Datenquelle durch ihren Wrapper angemeldet wurde (Syntax 5). Es kann somit durch den Datenpfad auf die Dokumente zugegriffen werden. Einige Anfragen (etwa `getContentAsDOM()`) würde der Wrapper in der Praxis mit einer `NEGOTIATION_VIOLATION` Exception quittieren, da sich die Anwendung nicht an die Vereinbarung hielt, die als Metadaten propagiert wurden und stattdessen Operationen nutzen möchte, die die Quelle nicht unterstützt. Der Vollständigkeit wegen sollen die Anfragen dennoch aufgelistet werden.

Syntax 5 *Datenquelle wählen*

```
String CollectionURI =
     "sharx://donau.informatik.uni-kl.de/";
Collection collection =
     DatasourceManager.getCollection(CollectionURI, null);
```

In dem vorliegenden Beispiel wurde vom Wrapper als ID der Pfad des XML-Dokuments im Dateisystem gewählt (Syntax 4). Das muss nicht so sein. Der Wrapper hat bei der Vergabe seiner Dokument-Identifikatoren völlig freie Hand und kann die Werte vergeben, die er möchte.

7.5.1 DOM Dokumentensuche

Suche eines Dokuments anhand der Dokumenten-ID. Das Ergebnis soll als DOM Dokument geliefert werden.

Syntax 6 *DOM Dokumentensuche*

```
String id = "publications/thesis/project/Sie99.xml"
XMLResource resource =
     (XMLResource) collection.getResource(id);

Document doc = (Document) resource.getContentAsDOM();
```

7.5.2 Text Dokumentensuche

Suche eines Dokumentes anhand der Dokumenten-ID. Das Ergebnis soll als XML-Text geliefert werden.

Syntax 7 *Text Dokumentensuche*

```
String id = "publications/thesis/project/Sie99.xml"
XMLResource resource =
     (XMLResource) collection.getResource(id);

String doc = (String) resource.getContent();
```

7.5.3 SAX Dokumentensuche

Suche eines Dokumentes anhand der Dokumenten-ID. Ein SAX-Handler soll sich um die Verarbeitung der Dokumentereignisse kümmern.

Syntax 8 *SAX Dokumentensuche*

```
String id = "publications/thesis/project/Sie99.xml"
XMLResource resource =
     (XMLResource) collection.getResource(id);
// Ein eigener SAX Handler der sich um die Ereignisse
// kümmert wird benötigt:
ContentHandler handler = new MyContentHandler();
resource.getContentAsSAX(handle);
```

7.5.4 Dokument einfügen mittels DOM

Einfügen eines neuen DOM Dokuments anhand einer vorgegebenen ID.

Syntax 9 *Dokument einfügen mittels DOM*

```
Document document;

String id = "publications/thesis/project/Sie99.xml";

XMLResource resource =
     (XMLResource) collection.createResource(id,
                                XMLResource.RESOURCE_TYPE);
resource.setContentAsDOM(document);
collection.storeResource(resource);
```

7.5.5 XML Text-Dokument einfügen

Einfügen eines neuen XML Text-Dokuments anhand einer vorgegebenen ID.

Syntax 10 *XML Text-Dokument einfügen*

```
String document;

String id = "publications/thesis/project/Sie99.xml";

XMLResource resource =
     (XMLResource) collection.createResource(id,
                                XMLResource.RESOURCE_TYPE);
resource.setContent(document);
collection.storeResource(resource);
```

7.5.6 Dokument einfügen mittels SAX

Unter Verwendung eines SAX `ContentHandler`s ein neues Dokument mit einer vorgegebenen ID einfügen.

Syntax 11 *Dokument einfügen mittels SAX*

```
//Als Quelle soll hier eine Datei fungieren.
String fileName = "Sie99.xml";

String id="publications/thesis/project/Sie99.xml";
XMLResource resource =
     (XMLResource) collection.createResource(id,
                                XMLResource.RESOURCE_TYPE);

ContentHandler handler = resource.setContentAsSAX;
XMLReader reader = XMLReaderFactory.createXMLReader();
reader.setContentHandler(handler);
reader.parse(new InputSource(fileName));
collection.storeResource(resource);
```

7.5.7 Dokument löschen

Löschen eines Dokuments mit bekannter ID.

Syntax 12 *Dokument löschen*

```
String id="publications/thesis/project/Sie99.xml";

collection.removeResource(collection.getRessource(id));
```

7.5.8 Dokument aktualisieren mittels DOM

Aktualisieren eines existierenden Dokuments mit bekannter ID mittels DOM.

Syntax 13 *Dokument Aktualisieren mittels DOM*

```
String id="publications/thesis/project/Sie99.xml";

XMLResource resource =
     (XMLResource) collection.getResource(id);
Document document = (Document) resource.getContentAsDOM();

// Hier findet die Änderung durch die Anwendung statt

resource.setContent(document);
collection.storeResource(resource);
```

7.5.9 Text XML-Dokument aktualisieren

Aktualisieren eines existierenden XML Text-Dokuments mit bekannter ID.

Syntax 14 *Text XML-Dokument aktualisieren*

```
String id="publications/thesis/project/Sie99.xml";

XMLResource resource =
     (XMLResource) collection.getResource(id);
String document = (String) resource.getContent();

// Hier findet die Änderung durch die Anwendung statt

resource.setContent(document);
collection.storeResource(resource);
```

7.5.10 Dokument aktualisieren mittels SAX

Für die Aktualisierung eines existierenden Dokuments mit bekannter ID, wobei sich ein SAX-Handler um die Verarbeitung der Dokumentenereignisse kümmern soll, existiert API-bedingt kein geeigneter Ansatz.

7.5.11 Suchen mittels XPath

Suche aller Publikationen des Autors „Siegel, S.“ mittels XPath. Das Ergebnis soll als DOM Dokument geliefert werden.

Syntax 15 *Suchen mittels XPath*

```
String xpath="/sharx/publications/thesis[@type='project']/"
             "author/name='Siegel, S.'";
XPAthQueryService service =
     (XPathQueryService)
          collection.getService("XPathQueryService",
"1.0");
ResourceSet resultSet = service.query(xpath);
ResourceIterator results = resultSet.getIterator();
while (results.hasMoreResources()) {
     XMLResource resource =
          (XMLResource) results.nextReource();
     Node result = resource.getContentAsDOM();
}
```

7.5.12 Aktualisieren mittels XUpdate

Hinzufügen eines Schlagworts zur Projektarbeit von Stefan Siegel (`name['Siegel, S.']`).

Syntax 16 *Aktualisieren mittels XUpdate*

```
String xupdate = "<xu:modifications version\"1.0\"" +
     "xmlns:xu=\"http://www.xmldb.org/xupdate\">" +
     "<xu:insert-after " +
     " select=\"/publications/thesis[@type='project']/\"" +
     "        \"author[name='Siegel, S.']\">" +
     "<xu:element name="keyword">INFORMIX</xu:element>" +
     "</xu:insert-after>" +
     "</xu:modifications>";
XPAthQueryService service =
     (XUpdateQueryService)
          collection.getService("XUpdateQueryService",
"1.0");
service.update(xupdate);
```

Kapitel 8 Zusammenfassung und Ausblick

In diesem Kapitel soll der Inhalt der Arbeit zusammengefasst und Ansätze aufgezeigt werden, wo und wie weiterer an diesem Projekt gearbeitet werden kann.

8.1 Zusammenfassung

Diese Arbeit unternimmt den Versuch, ein Wrapper-Framework zu entwerfen, dass es ermöglicht heterogene Datenquellen transparent an ein XML-basiertes Web Site Management System – konkret SHARX – anzubinden. Die so entstandene Schnittstelle bildet das Fundament, auf dem zukünftig eine hochpersonalisierte Datenausgabe erfolgen soll. Hierfür wurde das Projekt SHARX kurz vorgestellt. Anschließend erfolgte mit STRUDEL und MIX eine Übersicht über ähnliche Projekte.

Das relationale und das semistrukturierte Datenmodell wurden vorgestellt, und deren Vor- und Nachteile für die Verwendung als Einheitliche Sicht diskutiert. Anschließend wurde XML als Vertreter semistrukturierter Datenmodelle eingeführt und mit XLink – Referenzen in Dokumenten – und XUpdate – Aktualisieren der Dokumente – zwei Dialekte vorgestellt, die im Rahmen des Frameworks Verwendung finden sollten.

Um die Daten in bereits standardisierten Formaten bereitstellen zu können, wurden dann die zwei großen XML Zugriffsschnittstellen, „DOM – Document Object Model“ und „SAX – Simple API for XML“ vorgestellt. Da die Realisierung des Frameworks in Java vorgesen war, wurde auch auf die „JAXP – Java APIs for XML Processing“ eingegangen.

Ein großer Bereich beschäftigte sich dann mit dem Anbinden von Datenquellen. Es wurden Problemfälle wie verteilte Quellen, heterogene Datenmodelle und unterschiedliche Kodierungen erläutert. Der Mediator Ansatz, 1991 von Gio Wiederhold als „Konzept für zukünftige Informatinssysteme“ propagiert, wurde vorgestellt und erläutert, weshalb er bei der Verwendung heterogener Datenquellen erweitert werden muss. Die Verwendung von Wrappern als Konsequenz der bei Wiederhold fehlenden Schematransformation wurde dann motiviert. Im Anschluss wurde Garlic – ein Middleware System – vorgestellt, dass genau diesen Ansatz verfolgt. Mit „SQL/Med – Verwalten externer Daten“ und „XML:DB – ein XML basiertes Datenbank-Framework“ wurden zwei Standardisierungsansätze präsentiert, die für die beiden Datenmodelle (relational und

semistrukturiert) den Versuch unternehmen, externe Daten im jeweiligen Modell transparent zur Verfügung zu stellen.

Danach wurde mit dem Erarbeiten eines Wrapper-Frameworks begonnen. Es wurden zehn Anforderungen definiert. Davon vier nichtfunktionale Anforderungen (Aussagen zu Erstellungskosten, Erweiterbarkeit, Dynamische Anpassbarkeit und Anfrageoptimierung), zwei die sich mit den Eigenschaften der Quellen beschäftigten (Datenmodell und Datenkodierung) und vier die auf dynamische Aspekte der Quellen eingingen (Zugriffsmechanismus, Änderungsmöglichkeit, Protokolle und Sicherheitsmechanismen).

In der folgenden Konkretisierung wurden die Schnittstellen etwas genauer betrachtet. Der Wrapper wurde dafür in zwei Teile geteilt: den Anwendungswrapper und den Quellenwrapper. Es wurde dann auf die dadurch entstehenden drei Protokollschichten eingegangen. Die Anwendungsprotokollschicht ist für die Kommunikation mit der Anwendung und das Bereitstellen der Daten verantwortlich. Die Wrapperprotokollschicht soll die Migration der Quelle auf einen anderen Rechner erleichtern, da mit konsequenter Verwendung einer solchen Schnittstelle das Aufspalten eines Wrappers, auf die Reprogrammierung weniger Methoden begrenzt werden kann. Die Quellenprotokollschicht wurde wegen der Vielzahl unterschiedlicher Quelltypen nicht näher beschrieben, um die Kommunikation in der Praxis nicht unnötig einzuschränken.

Anschließend wurde die Implementierung des Frameworks, also die Definition der API, anhand eines Beispiels erläutert. Eine detaillierte Übersicht dazu ist im Anhang A zu finden.

8.2 Ausblick

Der Wunsch individuelle, personalisierte Inhalte aus dem Web abrufen zu können nimmt immer mehr zu. Als Folge davon wird der Bedarf an Inhaltsanbietern immer weiter steigen. Da die Zahl an Altdatenbeständen jedoch nicht so schnell schrumpfen wird, bleibt das Problem heterogene Quellen anbinden zu müssen auf absehbare Zeit akut. Ein Teil der damit verbundenen Probleme wurde in dieser Arbeit beleuchtet und versucht dafür eine Lösung vorzustellen. Dennoch sind weitere Überlegungen notwendig, um die Anwendungen effizient mit Daten zu versorgen.

Im Rahmen dieser Arbeit wurde eine Schnittstelle vorgesehen, um die Änderung der Metadaten an die Anwendung zu propagieren. Da die Dokumente jedoch nicht generell dem Wrapper unterstehen – etwa Dateien im Dateisystem – wäre es wünschenswert auch erkannte Änderungen in den Dokumenten selbst weiterreichen zu können, etwa wie ein Trigger bei relationalen DBS. Ein sinnvoller Einsatz wäre beispielsweise bei der Einführung eines Caches auf Ebene der *einheitlichen Sicht*. Der Wrapper könnte Änderungen seines Datenbestandes über die Kommunikationsschnittstelle propagieren und somit im einfachsten Fall mitteilen, dass der Cache-Inhalt zu invalidieren sei. Bei geschicktem Signalisieren der Änderung wäre es sogar denkbar, dass diese im Cache nachvollzogen werden kann, sodass eine kostspielige Neuanfrage über die eigentliche Datenschnittstelle vermieden werden kann.

Momentan ist für SHARX vorgesehen Bilder, Videos und sonstige nicht-XML Inhalte nur durch Hyperlinks in den Dokumenten zu repräsentiert. Es kann daher vorkommen, dass dem Anwender Dokumente generiert werden, die Verweise auf nicht(mehr) existierende Daten enthalten. In der API ist bereits eine Möglichkeit vorgesehen, auch diese Inhalte unter die Verwaltung des Wrappers zu stellen. Durch Verwendung der `BinaryResource`-Schnittstelle können verschiedenste

Dateien der Anwendung als Binär-Ressourcen bereitgestellt werden. Es wird dadurch möglich, auf Änderungen dieser Daten sofort sinnvoll zu reagieren.

Ein weiterer Aspekt, der untersucht werden müsste, ist, inwieweit es möglich ist, die Quellen in Klassen einzuteilen um diese durch eine Art generischen Wrapper anbinden zu können. So wäre beispielsweise ein OO-Wrapper denkbar, der es fertig brächte unter Zuhilfenahme geeigneter Informationen quasi halbautomatisch einen Wrapper zu konfigurieren. Entsprechend wäre auch ein Wrapper denkbar, der durch Abfrage der Metainformationen – etwa der Tabelle mit den Informationen der in dieser DB enthaltenen Tabellen – einer relationalen Datenbank diese anbinden kann, anstatt explizit auf eine spezielle Datenbank angepasst werden zu müssen.

Es bleibt auch zu klären, wie sinnvoll der Wrapper in der momentanen Implementierung in die Anfrageverarbeitung bzw. Optimierung eingebinden werden kann. Doch dies ist nur sinnvoll herauszufinden, wenn Mediator und Partitioner existieren (☞ Abbildung 27), die für die Erstellung der Teilanfragen zuständig sein sollen. Es wäre sicher hilfreich wenn eine einheitliche Anfragesprache – etwa XQuery – zur Verfügung stünde, doch dies ist mangels fehlender Standardisierung noch nicht möglich (es handelt sich noch um einen *„Working Draft“* des W3C).

Anhang A Framework Übersicht

Hier werden die notwendigen Pakete für das Wrapper-Framework von SHARX erklärt:

- `sharx.api`
- `sharx.api.base`
- `sharx.api.modules`
- `sharx.xml.parsers`

A.1 sharx.api

Das Paket `sharx.api` enthält alle notwendigen Klassen und Methoden, um ein Repository zu realisieren, an das Datenquellen mittels Wrapper ansgeschlossen werden können:

Klassenübersicht	
`Repository`	Methoden zur Datenquellenverwaltung.

A.1.1 Repository

Die Klasse `Repository` stellt Methoden zur Verfügung mit deren Hilfe sich neue Datenquellen registrieren können und auf bereits registrierte Quellen zugegriffen werden kann.

Methodenübersicht	
`static void`	**`deregisterDatasource(`**`Datasource datasource`**`)`** Meldet eine Datenquelleninstanz beim Repository ab.
`static Collection`	**`getCollection(`**`java.lang.String uri`**`)`** Liefert eine `Collection`-Instanz der durch die URI vorgegebenen Datenquelle.

static Collection	**getCollection(**java.lang.String uri, java.lang.String username, java.lang.String password**)** Liefert eine Collection-Instanz der durch die URI vorgegebenen Datenquelle.
static Datasource[]	**getDatasources()** Liefert eine Liste aller Datenquelleninstanzen, die beim Repository registriert sind.
static java.lang.String	**getProperty(**java.lang.String name**)** Liest eine Eigenschaft des Repositorys aus.
static void	**registerDatasource(**Datasource datasource**)** Registriert eine neue Datenquelleninstanz beim Repository.
static void	**setProperty(**java.lang.String name, java.lang.String value**)** Setzt eine Eigenschaft des Repositorys.
static int	**newMessage(**java.lang.String message**)** Nachricht einer Datenquelle.
static int	**newMessage(**org.w3c.dom.Node message**)** Nachricht einer Datenquelle.

von der Klasse java.lang.Object geerbte Methoden
equals, getClass, hashCode, notify, notifyAll, toString, wait

A.2 sharx.api.base

Das Paket sharx.api.base bildet die Basis der Repository-Wrapper-Schnittstelle:

Schnittstellenübersicht	
Collection	Repräsentiert eine Sammlung von XML-Dokumenten, wie sie in einer Datenquelle vorliegt.
Configurable	Bietet die Möglichkeit Objekteigenschaften einzustellen.
Datasource	Eine Kapselung einer Datenquelle durch den Wrapper, der die XML-Dokumente bereitstellt.

Resource	Ein Container für die in der Datenquelle gespeicherten Daten.
ResourceIterator	Diese Schnittstelle wird benötigt, um über eine Menge von Ressourcen zu iterieren.
ResourceSet	Ein Container für eine Menge von Ressourcen.
Service	Die Schnittstelle bietet einen Erweiterungsmechanismus, für die Implementierung von `Collection`. Sie bestehen aus `Service`-Instanzen, die ihre eignen Methoden besitzen, um die nötigen Aktionen ausführen zu können.

Klassenübersicht	
ErrorCodes	Die Klasse definiert die SHARX Fehlernummern, die zum Setzen der `errorCodes` Attribute einer `WrapperException` verwendet werden können.

Exceptionsübersicht	
WrapperException	Eine `WrapperException` wird immer dann ausgelöst, wenn in der Wrapper-API ein Fehler auftritt.

A.2.1 Collection

Eine `Collection` ist eine Ansammlung von Ressourcen, die in einer Datenquelle vorhanden sind. Eine Datenquelle kann eine Hierarchie dieser Ansammlungen als Vater-Sohn Ansammlung anbieten.

Methodenübersicht	
void	**`close()`** Gibt alle durch die Ansammlung vereinnahmten Ressourcen wieder frei.
java.lang.String	**`createId()`** Erzeugt eine eindeutige ID im Kontext der Ansammlung.
Resource	**`createResource(`**`java.lang.String id, java.lang.String type`**`)`** Erzeugt eine neue leere Ressource mit der vorgegebenen ID.

Collection	**getChildCollection(**java.lang.String name**)** Liefert eine Ansammlungsinstanz für die gewünschte Sohn-Ansammlung, falls diese existiert.
int	**getChildCollectionCount()** Liefert die Anzahl Sohn-Ansammlungen der Instanz, sofern welche existieren und 0 andernfalls.
java.lang.String	**getName()** Liefert den Namen der Ansammlungsinstanz.
Collection	**getParentCollection()** Liefert die Vater-Ansammlung der Instanz oder Null, wenn diese nicht existiert.
Resource	**getResource(**java.lang.String id**)** Liefert eine Ressource der Datenquelle.
int	**getResourceCount()** Liefert die Anzahl Ressourcen, die in dieser Ansammlung gespeichert sind bzw. 0, wenn die Ansammlung leer ist.
Service	**getService(**java.lang.String name, java.lang.String version**)** Liefert eine Service-Instanz mit dem gegebenen Namen und der gegebenen Version.
Service[]	**getServices()** Liefert alle der Ansammlung bekannten Dienste.
boolean	**isOpen()** Liefert TRUE, wenn die Ansammlung geöffnet ist, andernfalls wird FALSE zurückgeliefert.
java.lang.String[]	**listChildCollections()** Liefert eine Liste mit den Namen aller Sohn-Ansammlungen.
java.lang.String[]	**listResources()** Liefert eine Liste aller in der Ansammlung gespeicherten Ressourcen.
void	**removeResource(**Resource res**)** Entfernt die angegebene Ressource aus der Datenquelle.
void	**storeResource(**Resource res**)** Speichert die angegebene Ressource in der Datenquelle.

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), setProperty(java.lang.String name, java.lang.String value)

A.2.2 Configurable

Bietet die Möglichkeit die Eigenschaften eines Objekts zu konfigurieren.

Methodenübersicht	
java.lang.String	**getProperty(**java.lang.String name**)** Liefert den Wert der durch name identifizierten Eigenschaft.
void	**setProperty(**java.lang.String name, java.lang.String value**)** Setzt den Wert, der durch name identifizierten Eigenschaft, auf value.

A.2.3 Datasource

Mittels Datasource wird eine Datenquelle gekapselt, mit anderen Worten es handelt sich hier um den Wrapper, der die Datenquelle verwaltet. Jede Datenquelle benötigt ihre eigene Implementierung von Datasource, die am Repository angemeldet werden muss, bevor die Quelle angesprochen werden kann.

Methodenübersicht	
boolean	**acceptsURI(**java.lang.String uri**)** Gibt an, ob diese Datenquelleninstanz die angegebene URI verarbeiten kann oder nicht.
Collection	**getCollection(**java.lang.String uri, java.lang.String username, java.lang.String password**)** Liefert eine Ansammlungsinstanz anhand der in uri übergebenen URI.
java.lang.String	**getName()** Liefert den mit der Datenquelle assoziierten Namen.

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), setProperty(java.lang.String name, java.lang.String value)

A.2.4 Resource

Resource ist ein Container für Daten der Datenquelle. Rohdaten in Form von Resource sind nutzlos. Man benötigt eine konkrete Implementierung für das Verarbeiten eines speziellen Datentyps, um sinnvoll damit arbeiten zu können.

Methodenübersicht	
java.lang.Object	**getContent()** Gibt an, ob diese Datenquelleninstanz die angegebene URI verarbeiten kann oder nicht.
java.lang.String	**getId()** Liefert die eindeutige ID der Ressource, bzw. Null wenn es sich um eine anonyme Ressource handelt.
Collection	**getParentCollection()** Liefert die Ansammlungsinstanz, zu der die Ressource gehört.
java.lang.String	**getResourceTyp()** Liefert den Typ der Ressource.
void	**setContent(**java.lang.Object value**)** Setzt den Wert dieser Ressource.

A.2.5 ResourceIterator

Mittels ResourceIterator kann über eine Ressourcenmenge iteriert werden.

Methodenübersicht	
boolean	**hasMoreResources()** Liefert TRUE, solange weitere Ressourcen existieren, über die iteriert werden kann.
Resource	**nextResource()** Liefert die nächste Ressourceninstanz des Iterators.

A.2.6 ResourceSet

`ReourceSet` ist eine Ansammlung von Ressourcen. Es handelt sich hierbei nicht um eine Menge im mathematischen Sinn.

Methodenübersicht	
`void`	**`addResource(`**`Resource res`**`)`** Erweitert die Menge um die Ressourceninstanz `res`.
`void`	**`clear()`** Entfernt alle Ressourceninstanzen aus der Menge.
`ResourceIterator`	**`getIterator()`** Liefert einen Iterator für die Ressourceninstanzen der Menge.
`Resource`	**`getMembersAsResource()`** Liefert eine Ressource, die eine XML-Repräsentation aller in der Menge gespeicherten Ressourcen enthält.
`Resource`	**`getResource(`**`long index`**`)`** Liefert die Ressourceninstanz, die sich unter `index` in der Menge befindet.
`long`	**`getSize()`** Liefert die Anzahl Elemente der Menge.
`void`	**`removeResource(`**`long index`**`)`** Entfernt die Ressourceninstanz, die sich unter `index` in der Menge befindet.

A.2.7 Service

Die Schnittstelle Service bietet einen Erweiterungsmechanismus, für die Implementierung von `Collection`. Sie bestehen aus Service-Instanzen, die ihre eignen Methoden besitzen, um die nötigen Aktionen ausführen zu können.

Methodenübersicht	
`java.lang.String`	**`getName()`** Liefert den Namen, der mit dieser Service-Instanz assoziiert ist.

java.lang.String	**getVersion()** Liefert die Version des Service-Objekts.
void	**setCollection(**Collection col**)** Legt das Ansammlungs-Attribut des Service-Objekts fest.

von der Schnittstelle sharx.api.base.Configurable **geerbte Methoden**

getProperty(java.lang.String name),
setProperty(java.lang.String name, java.lang.String value)

A.2.8 ErrorCodes

ErrorCodes definiert die Sharx-Wrapper Fehlernummern, die verwendet werden können, um die errorCodes einer WrapperException zu setzen.

Feldübersicht

static int	**COLLECTION_CLOSED** Wird gesetzt, wenn eine Methode auf eine geschlossene Ansammlung angewendet wird.
static int	**INVALID_COLLECTION** Wird gesetzt, wenn eine Ansammlung in einem ungültigen Zustand ist.
static int	**INVALID_DATASOURCE** Wird gesetzt, wenn eine Datenquelleninstanz beim Registrierungsversuch als ungültig erkannt wird.
static int	**INVALID_RESOURCE** Wird gesetzt, wenn die Ressource, auf die eine Operation ausgeführt werden soll, ungültig ist.
static int	**INVALID_URI** Wird gesetzt, wenn die angegebene URI ein ungültiges Format aufweist.
static int	**NEGOTIATION_VIOLATION** Wird gesetzt, wenn die Anwendung Methoden verwenden möchte, deren Unterstützung die konkrete Datenquelle nie versprochen hat.

static int	**NO_SUCH_COLLECTION** Wird gesetzt, wenn die angeforderte Ansammlung nicht gefunden werden kann.
static int	**NO_SUCH_DATASOURCE** Wird gesetzt, wenn die Datenquelle für die angegebene URI nicht gefunden werden kann.
static int	**NO_SUCH_RESOURCE** Wird gesetzt, wenn die gewünschte Ressource nicht gefunden werden kann.
static int	**NO_SUCH_SERVICE** Wird gesetzt, wenn der gewünschte Service nicht gefunden werden kann.
static int	**NOT_IMPLEMENTED** Wird gesetzt, wenn die API die gewünschte Implementierung nicht bereitstellt.
static int	**PERMISSION_DENIED** Wird gesetzt, wenn die gewünsche Ansammlung mangels Zugriffsrechten nicht an den Anfragenden geliefert werden können.
static int	**UNKNOWN_ERROR** Wird gesetzt, wenn ein Fehler auftrat, dessen Urasache jedoch nicht bekannt ist.
static int	**UNKNOWN_RESOURCE_TYPE** Wird gesetzt, wenn der gewünschte Ressourcen-Typ der API-Implementierung unbekannt ist.
static int	**VENDOR_ERROR** Wird gesetzt, wenn ein Fehler in der Datenquelle auftritt.
static int	**WRONG_CONTENT_TYPE** Wird gesetzt, wenn der Inhalt einer Ressource vom angegebenen Typ abweicht.

von der Klasse `java.lang.Object` geerbte Methoden

equals, getClass, hashCode, notify, notifyAll, toString, wait

A.2.9 WrapperException

```
java.lang.Object
  |
  +--java.lang.Throwable
        |
        +--java.lang.Exception
              |
              +--sharx.api.base.WrapperException
```

Eine `WrapperException` wird bei allen auftretenden Fehlern, die in Zusammenhang mit Wrappern stehen, ausgelöst. Sie liefert zwei Fehlernummern. Eine, die die eigenen Fehler maskiert und eine weitere, die die Fehler der zu Grunde liegenden Datenquelle weiterreicht. Tritt nur ein Datenquellenfehler auf, so muss `errorCode` auf den Wert `errorCodes.VENDOR_ERROR` gesetzt werden.

Feldübersicht	
`int`	**`errorCode`** Fehlernummer des Wrappers, wenn beim Arbeiten ein Fehler auftrat.
`int`	**`vendorErrorCode`** Fehlernummer, die die Datenquelle produzierte, wenn dort ein Fehler auftrat.

von der Klasse `java.lang.Throwable` geerbte Methoden
`fillInStackTrace, getLocalizedMessage, getMessage, printStackTrace, toString`

von der Klasse `java.lang.Object` geerbte Methoden
`equals, getClass, hashCode, notify, notifyAll, toString, wait`

A.3 sharx.api.modules

Das Paket `sharx.api.modules` stellt Erweiterungsmodule der Repository-Wrapper-Schnittstelle bereit:

Schnittstellenübersicht	
`BinaryResource`	Ressource zum Bereitstellen von Binärdaten aus der Datenquelle.

CollectionManagement-Service	Der `CollectionManagementService` ist ein Dienst, der die Verwaltung von Ansammlungen in einer Datenquelle ermöglicht.
TransactionService	Bietet die Möglichkeit, eine Reihe von Aktionen auf Ansammlungen als Transaktion zu kapseln.
XMLResource	Ressource zum Bereitstellen von XML aus der Datenquelle.
XPathQueryService	Beim `XPathQueryService` handelt es sich um einen Dienst, der es ermöglicht, XPath Anfragen auf eine einzelne XML-Ressource anzuwenden, die in Form einer Ansammlung vorliegt.
XUpdateQueryService	Beim `XUpdateQueryService` handelt es sich um einen Dienst, der es ermöglicht, XUpdate Anfragen auf eine einzelne XML-Ressource anzuwenden, die in Form einer Ansammlung vorliegt.

A.3.1 BinaryResource

Ressource zum Bereitstellen von Binärdaten, die neben Text in der Datenquelle vorhanden sind. Die Unterstützung dieses Moduls ist optional.

Die Standardmethode `getContent` liefert eine Bytekette (`byte[]`), die Standardmethode setContent erwartet eine Bytekette (`byte[]`).

Feldübersicht	
`static java.lang.String`	**RESOURCE_TYPE** Weist die Ressource als Binärdaten aus.

von der Schnittstelle `sharx.api.base.Resource` geerbte Methoden
`getContent()`, `getId()`, `getParentCollection()`, `getResourceTyp()`, `setContent(java.lang.Object value)`

A.3.2 CollectionManagementService

Der `CollectionManagementService` ist ein Dienst, der die Verwaltung von Ansammlungen in einer Datenquelle ermöglicht. Da die Datenquellen ein sehr unterschiedliches Vermögen

im Umgang mit Ansammlungen an den Tag legen, bietet diese Schnittstelle eine nur sehr rudimentäre Unterstützung.

Methodenübersicht	
Collection	**createCollection(**java.lang.String name**)** Erzeugt eine neue Ansammlung in der Datenquelle.
void	**removeCollection(**java.lang.String name**)** Entfernt die durch name vorgegebene Ansammlung aus der Datenquelle.

von der Schnittstelle sharx.api.base.Service geerbte Methoden
getName(), getVersion(), setCollection(Collection col)

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), **setProperty(java.lang.String name, java.lang.String value)**

A.3.3 TransactionService

Bietet die Möglichkeit eine Reihe von Aktionen auf Ansammlungen als Transaktion zu kapseln.

Methodenübersicht	
void	**begin()** Startet eine neue Transaktion.
void	**commit()** Beendet die Transaktion.
void	**rollback()** Bricht die Transaktion ab und stellt den Ursprungszustand wieder her.

von der Schnittstelle sharx.api.base.Service geerbte Methoden
getName(), getVersion(), setCollection(Collection col)

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), setProperty(java.lang.String name, java.lang.String value)

A.3.4 XMLResource

Bietet den Zugriff auf die XML-Ressourcen, die in der Datenquelle vorhanden sind. Auf eine XMLResource kann entweder als Text, mittels DOM oder SAX zugegriffen werden.

Die Verwendung von `getContent` und `setContent` arbeitet auf den Ressourcen als Text.

Feldübersicht	
static java.lang.String	**RESOURCE_TYPE** Weist die Ressource als XML aus.

Methodenübersicht	
org.w3c.dom.Node	**getContentAsDOM()** Liefert den Inhalt der Ressource als DOM Objekt.
void	**getContentAsSAX(**org.xml.sax.ContentHandler handler**)** Ermöglicht es der Anwendung mit dem ContentHandler handler den Inhalt der Datenquelle zu parsen.
java.lang.String	**getDocumentId()** Liefert die eindeutige ID des XML-Dokuments, zu dem diese Ressource gehört, bzw. Null wenn es kein Vaterdokument gibt.

void	**setContentAsDOM(**org.w3c.dom.Node content**)** Setzt den Inhalt von content als neuen Wert der Ressource.
org.xml.sax.ContentHandler	**setContentAsSAX()** Setzt den Inhalt der Ressource mit Hilfe eines SAX ContentHandlers.

von der Schnittstelle sharx.api.base.Resource **geerbte Methoden**

getContent(), getId(), getParentCollection(), getResourceTyp(), setContent(java.lang.Object value)

A.3.5 XPathQueryService

XPathQueryService bietet die Möglichkeit, XPath Anfragen über eine Collection oder eine einzelne Resource als Teil einer Collection zu stellen.

Methodenübersicht

void	**clearNamespaces()** Löscht das Namensraum-Register.
java.lang.String	**getNamespace(**java.lang.String prefix**)** Liefert die URI, mit der prefix assoziiert ist, aus dem internen Namensraum-Register.
ResourceSet	**query(**java.lang.String query**)** Führt die XPath-Anfrage query auf der Ansammlung aus.
ResourceSet	**queryResource(**java.lang.String id, java.lang.String query**)** Führt die XPath-Anfrage query auf einer XML-Ressource der Ansammlung aus.
void	**removeNamespace(**java.lang.String prefix**)** Entfernt den mit prefix assoziierten Namensraum aus dem Namensraum-Register.
void	**setNamespace(**java.lang.String prefix, java.lang.String uri**)** Setzt prefix als Namensraum für uri im Namensraum-Register.

von der Schnittstelle sharx.api.base.Service geerbte Methoden
getName(), getVersion(), setCollection(Collection col)

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), setProperty(java.lang.String name, java.lang.String value)

A.3.6 XUpdateQueryService

Beim XUpdateQueryService handelt es sich um einen Dienst, der es ermöglicht, XUpdate Anfragen auf eine einzelne XML-Ressource anzuwenden, die in Form einer Ansammlung vorliegt.

Methodenübersicht	
long	**update(**java.lang.String commands**)** Führt ein XUpdate auf der Ansammlung aus.
long	**updateResource(**java.lang.String id, java.lang.String commands**)** Führt ein XUpdate auf einer XML-Ressource der Ansammlung aus.

von der Schnittstelle sharx.api.base.Service geerbte Methoden
getName(), getVersion(), setCollection(Collection col)

von der Schnittstelle sharx.api.base.Configurable geerbte Methoden
getProperty(java.lang.String name), setProperty(java.lang.String name, java.lang.String value)

A.4 sharx.xml.parsers

A.4.1 sharx.xml.parsers.DocumentBuilder

`sharx.xml.parsers.DocumentBuilder`

Methodenübersicht	
abstract DomImplementation	**`get DomImplementation()`** Liefert eine Instanz eines `DOMImplementation`-Objekts.
abstract boolean	**`isNamespaceAware()`** Gibt an, ob diese Parserinstanz konfiguriert wurde, dass sie den Umgang mit XML-Namensräumen beherrscht.
abstract boolean	**`isValidatein()`** Gibt an, ob die Parserinstanz konfiguriert wurde, XML-Dokumente zu validieren.
abstract Document	**`newDocument()`** Liefert eine neue Instanz eines DOM `Document` Objekts, um damit einenen DOM Baum erzeugen zu können.
Document	**`parse(`**`java.io.File f`**`)`** Parst das Dokumet aus Datei `f` als XML-Dokument und liefert ein neues DOM `Document` Objekt zurück.
abstract Document	**`parse(`**`InputSource is`**`)`** Parst das Dokumet der angegebenen Quelle `is` als XML-Dokument und liefert ein neues DOM `Document` Objekt zurück.
Document	**`parse(`**`java.io.InputStream is`**`)`** Parst das Dokumet des angegebenen `InputStream`s als XML-Dokument und liefert ein neues DOM `Document` Objekt zurück.
Document	**`parse(`**`java.io.InputStream is, java.lang.String systemId`**`)`** Parst das Dokumet des angegebenen `InputStream`s als XML-Dokument und liefert ein neues DOM `Document` Objekt zurück.

Document	**parse(**java.lang.String uri**)** Parst den Inhalt der angegebenen `uri` als XML-Dokument und liefert ein neues DOM `Document` Objekt zurück.
abstract void	**setEntityResolver(**EntityResolver er**)** Angabe des `EntityResolver`s, der verwendet werden soll, Entitys im XML-Dokument aufzulösen.
abstract void	**setErrorHandler(**ErrorHandler eh**)** Angabe des `ErrorHandler`s der verwendet werden soll, um auf Feher im gegebenen XML-Dokument zu reagieren.

A.4.2 sharx.xml.parsers.DocumentBuilderFactory

`sharx.xml.parsers.DocumentBuilderFactory`

Methodenübersicht	
abstract java.lang.Object	**getAttribute(**java.lang.String name**)** Ermöglicht es dem Anwender Attribute der zu Grunde liegenden Implementierung zu erhalten.
boolean	**isCoalescing()** Gibt an, ob die Factory so konfiguriert wurde, dass sie Parser erzeugt, die CDATA-Knoten in Textknoten wandelt und an benachbarte Textknoten, sofern diese existieren, anhängt.
boolean	**isExpandEntityReferences()** Gibt an, ob die Factory so konfiguriert wurde, dass sie Parser erzeugt, die Entity-Referenz-Knoten expantieren.
boolean	**isIgnoringComments()** Gibt an, ob die Factory so konfiguriert wurde, dass sie Kommentare ignoriert.
boolean	**isIgnoringElementContentWitespace()** Gibt an, ob die Factory so konfiguriert wurde, dass sie Parser erzeugt, die ignorierbare Freiraumzeichen in Elementen ignoriert.

boolean	**isNamespaceAware()** Gibt an, ob die Factory so konfiguriert wurde, dass sie den Umgang mit XML-Namensräumen beherrscht.
boolean	**isValidating()** Gibt an, ob die Factory so konfiguriert wurde, dass sie den XML Inhalt während des Parsens validiert.
abstract DocumentBuilder	**newDocumentBuilder()** Erzeut eine neue Instanz eines `DocumentBuilder` Objekts unter Verwendung aller bis Dato konfigurierten Parameter.
static DocumentBuilderFactory	**newInstance()** Liefert eine neue Instanz einer `DocumentBuilderFactory`.
abstract void	**setAttribute(**`java.lang.String name, java.lang.Object value`**)** Ermöglicht es dem Anwender Attributer der zu Grunde ligenden Implementation zu setzen.
void	**setCoalesing(**`boolean coalescing`**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, CDATA Knoten in Textknoten wandelt und sie an benachbarte Knoten hängt (falls es solche gibt).
void	**setExpandEntityReferences(**`boolean expandEntityRef`**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, Entity-Referenzknoten expandiert.
void	**setIgnoringComments(**`boolean ignoreComments`**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, Kommentare ignoriert oder nicht.
void	**setIgnoringElementContentWhitespace(**`boolean whitespace`**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, beim Parsen von XML-Dokumenten Leerraumzeichen in Element-Objekten entfernen muss oder nicht.

void	**setNamespaceAware(**boolean awareness**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, den Umgang mit XML-Namensräumen beherrscht, oder nicht.
void	**setValidating(**boolean validating**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, Dokumente beim Parsen auf ihre Wohlgeformtheit hin untersucht.

A.4.3 sharx.xml.parsers.FlatStream

sharx.xml.parsers.FlatStream

Methodenübersicht	
String	**getContent()** Liefert den Inhalt der XMLResoure als String im XML-Format.
XMLResource	**getResource(**java.io.File f**)** Bindet das Dokumet aus Datei f an eine XMLResource.
abstract XMLResource	**getResource(**InputSource is**)** Bindet den Inhalt der gegebenen Quelle is an eine XMLResource.
XMLResource	**getResource(**javaio.InputStream is**)** Bindet den Inhalt des gegebenen Datenstroms is an eine XMLResource.
XMLResource	**getResource(**java.lang.string uri**)** Bindet den Inhalt des gegebenen Ressourcen Identifikators (URI) an eine XMLResource.
abstract XMLResource	**newResource()** Liefert eine neue Instanz eines XMLResource Objekts.

A.4.4 sharx.xml.parsers.SAXParser

`sharx.xml.parsers.SAXParser`

Methodenübersicht	
`abstract parser`	**`getParser()`** Liefert den SAX Parser, der durch diese Klasse gekapselt ist, zurück.
`abstract java.lang.Object`	**`getProperty(`**`java.lang.String name`**`)`** Ermöglicht es dem Anwender Eigenschaften der zu Grunde liegenden Implementierung von `XMLReader` zu erhalten.
`abstract XMLReader`	**`getXMLReader()`** Liefert den XML Leser, der durch diese Klasse gekapselt ist zurück.
`abstract boolean`	**`isNamespaceAware()`** Gibt an, ob die Factory so konfiguriert wurde, dass sie den Umgang mit XML-Namensräumen beherrscht.
`abstract boolean`	**`isValidating()`** Gibt an, ob die Factory so konfiguriert wurde, dass sie den XML Inhalt während des parsens validiert.
`void`	**`parse(`**`java.io.File f, DefaultHandler dh`**`)`** Parst das Dokumet aus Datei f als XML-Dokument mit dem Standard-Handler `dh`.
`void`	**`parse(`**`java.io.File f, HandlerBase hb`**`)`** Parst das Dokumet aus Datei `f` als XML-Dokument mit dem speziellen Handler `hb`.
`void`	**`parse(`**`InputSource is, DefaultHandler dh`**`)`** Parst den Inhalt der gegebenen Quelle `is` als XML mit Hilfe des Standard-Handlers `dh`.
`void`	**`parse(`**`InputSource is, HandlerBase hb`**`)`** Parst den Inhalt der gegebenen Quelle `is` als XML mit Hilfe des speziellen Handler `hb`.
`void`	**`parse(`**`javaio.InputStream is, DefaultHandler dh`**`)`** Parst den Inhalt des gegebenen Datenstroms `is` als XML mit Hilfe des Standard-Handlers `dh`.

void	**parse(**javaio.InputStream is, DefaultHandler dh, java.lang.String systemId**)** Parst den Inhalt des gegebenen Datenstroms is als XML mit Hilfe des Standard-Handlers dh.
void	**parse(**javaio.InputStream is, HandlerBase hb**)** Parst den Inhalt des gegebenen Datenstroms is als XML mit Hilfe des speziellen Handler hb.
void	**parse(**javaio.InputStream is, HandlerBase hb, java.lang.String systemId**)** Parst den Inhalt des gegebenen Datenstroms is als XML mit Hilfe des speziellen Handler hb.
void	**parse(**java.lang.string uri, DefaultHandler dh**)** Parst den Inhalt des gegebenen Ressourcen Identifikators (URI) als XML mit Hilfe des Standard-Handlers dh.
void	**parse(**java.lang.string uri, HandlerBase hb**)** Parst den Inhalt des gegebenen Ressourcen Identifikators (URI) als XML mit Hilfe des speziellen Handler hb.
abstract void	**setProperty(**java.lang.String name, java.lang.Object value**)** Setzt den Wert der zu Grunde liegenden Impelentierung von XMLReader.

A.4.5 sharx.xml.parsers.SAXParserFactory

sharx.xml.parsers.SAXParserFactory

Methodenübersicht	
abstract boolean	**getFeature(**java.lang.String name**)** Gibt an, ob die zu Grunde liegenden Implementierung von org.xml.sax.XMLReader die entsprechende Eigendschaft besitzt.
boolean	**isNamespaceAware()** Gibt an, ob die Factory so konfiguriert wurde, dass sie den Umgang mit XML-Namensräumen beherrscht.
boolean	**isValidating()** Gibt an, ob die Factory so konfiguriert wurde, dass sie den XML Inhalt während des parsens validiert.

static SAXParserFactory	**newInstance()** Liefert eine neue Instanz einer SAXParserFactory.
abstract SAXParser	**newSAXParser()** Erzeut eine neue Instanz eines SAXParser Objekts unter Verwendung aller bis Dato konfigurierten Parameter.
abstract void	**setFeature(**java.lang.String name, boolean value**)** Setzt oder löscht die Eigenschaft der zu Grunde liegenden Implementierung von org.xml.sax.XMLReader.
void	**setNamespaceAware(**boolean awareness**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, den Umgang mit XML-Namensräumen beherrscht, oder nicht.
void	**setValidating(**boolean validating**)** Legt fest, ob die Parserinstanz, die durch diesen Kode erzeugt wird, Dokumente beim Parsen auf ihre Wohlgeformtheit hin untersucht.

A.4.6 sharx.xml.update.XUpdate

sharx.xml.update.XUpdate

Methodenübersicht	
abstract long	**update(**java.lang.String commands, InputSource is**)** Führt die XUpdate Befehle aus commands auf der gegebenen Quelle is aus und liefert die Anzahl geänderter Knoten zurück.
long	**update(**java.lang.String commands, sharx.xml.parser.DocumentBuilder.Document d**)** Führt die XUpdate Befehle aus commands auf dem DOM Dokument d aus und liefert die Anzahl geänderter Knoten zurück.
long	**update(**java.lang.String commands, sharx.xml.parser.FlatStream.XMLResource x**)** Führt die XUpdate Befehle aus commands auf der XMLResource x aus und liefert die Anzahl geänderter Knoten zurück.

Anhang B DTDs

B.1 DTD der Konfigurationsschnittstelle

```
1     <!--
2
3        SHARX - Wrapper Communication Messages (WCM) 0.1 initial DTD.
4
5        WRAPPERMESSAGE is an XML dialect to propagate or update Wrapper informations
6        to a running SHARX data repository.
7
8        Info about wrapperMessage can be found at http://wwwagdbis.informatik.uni-kl.de/sharx/
9        XML Info can be found at http://www.w3.org/XML/
10
11       Copyright © 2001 AG DBIS - Universität Kaiserslautern
12
13       Stefan Siegel - <siegel@informatik.uni-kl.de>
14
15    -->
16
17    <!ELEMENT wrapperMessage (wrapper, service*, collection*, updates*)>
18    <!ATTLIST wrapperMessage
19             version        CDATA    #FIXED "0.1">
20    <!ATTLIST wrapperMessage
21             messageclass   (initial |
22                             update)    #REQUIRED>
23
24    <!ELEMENT wrapper       (name, base, query+, description?, protected?, access*)>
25
26    <!ELEMENT name (#PCDATA)>
27
28    <!ELEMENT description (#PCDATA)>
29
30    <!ELEMENT base EMPTY>
31    <!ATTLIST base
32             href           CDATA       #REQUIRED>
33
34    <!ELEMENT query EMPTY>
35    <!ATTLIST query
36             format         (dom |
37                             flat |
38                             sax)       #REQUIRED>
39
40    <!ELEMENT protected EMPTY>
41
42    <!ELEMENT access EMPTY>
43    <!ATTLIST access
44             privilege      (delete |
45                             insert |
46                             read |
47                             tupdate |
48                             xupdate)   #REQUIRED>
49
50    <!ELEMENT service (serviceName serviceVersion)>
51
52    <!ELEMENT serviceName (#PCDATA)>
53
54    <!ELEMENT serviceVersion (#PCDATA)>
```

```
55
56    <!ELEMENT collection ((collection | document)+, access*)>
57    <!ATTLIST collenction
58              id              ID          #REQUIRED>
59
60    <!ELEMENT document (dtd?, size?, access*, property?)>
61    <!ATTLIST document
62              id              ID          #REQUIRED>
63
64    <!ELEMENT property EMPTY>
65    <!ATTLIST property
66              type            (materialize) #REQUIRED>
67
68
69    <!ELEMENT dtd EMPTY>
70    <!ATTLIST dtd
71              href            CDATA       #REQUIRED>
72
73    <!ELEMENT size (#PCDATA)>
74    <!ATTLIST size
75              unit            (byte
76                              |kilobyte
77                              |megabyte
78                              |gigabyte)  #DEFAULT "byte">
79
80    <!ELEMENT updates (xpathUpdate, xupdateUpdate)>
81
82    <!ELEMENT xpathUpdate (#PCDATA)>
83
84    <!ELEMENT xupdateUpdate (#PCDATA)>
```

B.2 Publikations-DTD

```
1     <!--
2
3        Publications list 0.1 initial DTD.
4
5        This ist an XML dialect to store informations about publications made
6        by working groups of german universities.
7
8        Copyright © 2001 AG DBIS - Universität Kaiserslautern
9
10       Marcus Flehmig - <flehmig@informatik.uni-kl.de>
11
12    -->
13
14    <!ELEMENT publications (article|talk|dissertation|thesis|book|chapter)*>
15
16    <!ENTITY % mandatoryfields    "title,year,month?,author+,resources,
17                                   language,project*,abstract?,keyword?">
18    <!ENTITY % articlefields      "pages?,published">
19    <!ENTITY % disfields          "cite?,supervisor">
20    <!ENTITY % thesisfields       "supervisor?,pages?,note?">
21    <!ENTITY % bookfields         "editor|address|pages|volume|number|publisher|
22                                   note|ref|crossref|isbn|series">
23
24    <!ELEMENT talk                (%mandatoryfields;)>
25
26    <!ELEMENT article             (%mandatoryfields;,%articlefields;)>
27    <!ATTLIST article
28                      type        CDATA  #IMPLIED
29    >
30
31    <!ELEMENT dissertation        (%mandatoryfields;,%disfields;,(%bookfields;)*)>
32    <!ATTLIST dissertation
33                      type        (professorial|doctoral)   #REQUIRED
34    >
35
36    <!ELEMENT thesis              (%mandatoryfields;,%thesisfields;)>
37    <!ATTLIST thesis
38                      type        (diploma|master|project)  #REQUIRED
```

```
39    >
40
41    <!ELEMENT book               (%mandatoryfields;,(%bookfields;)*)>
42    <!ELEMENT journal            (%mandatoryfields;,(%bookfields;)*)>
43    <!ELEMENT chapter            (%mandatoryfields;,(%bookfields;)*)>
44
45
46    <!ELEMENT resources          (paper|presentation|related)*>
47    <!ELEMENT paper              (#PCDATA)>
48    <!ATTLIST paper
49                       href        CDATA #IMPLIED
50    >
51    <!ELEMENT presentation       (#PCDATA)>
52    <!ATTLIST presentation
53                       href        CDATA #IMPLIED
54    >
55    <!ELEMENT related            (#PCDATA)>
56    <!ATTLIST related
57
58                       href        CDATA #IMPLIED
59    >
60
61    <!ELEMENT published          (#PCDATA|book|journal)*>
62    <!ATTLIST published
63                       accepted    CDATA #IMPLIED
64    >
65
66    <!ELEMENT author             (name,email,address?)>
67    <!ELEMENT name               (#PCDATA)>
68    <!ELEMENT email              EMPTY>
69    <!ATTLIST email
70                       href        CDATA #REQUIRED
71    >
72
73    <!ELEMENT editor             (#PCDATA)>
74    <!ELEMENT address            (#PCDATA)>
75
76    <!ELEMENT abstract           (#PCDATA)>
77
78    <!ELEMENT supervisor         (#PCDATA)>
79
80    <!ELEMENT title              (#PCDATA)>
81    <!ELEMENT pages              (#PCDATA)>
82    <!ELEMENT year               (#PCDATA)>
83    <!ELEMENT language           (#PCDATA)>
84    <!ELEMENT project            (#PCDATA)>
85    <!ELEMENT keyword            (#PCDATA)>
86    <!ELEMENT journal            (#PCDATA)>
87    <!ELEMENT volume             (#PCDATA)>
88    <!ELEMENT number             (#PCDATA)>
89    <!ELEMENT month              (#PCDATA)>
90    <!ELEMENT cite               (#PCDATA)>
91    <!ELEMENT publisher          (#PCDATA)>
92    <!ATTLIST publisher
93                       href        CDATA #IMPLIED
94    >
95    <!ELEMENT note               (#PCDATA)>
96
97    <!ELEMENT crossref           (#PCDATA)>
98    <!ELEMENT isbn               (#PCDATA)>
99    <!ELEMENT series             (#PCDATA)>
100   <!ATTLIST series
101                      href        CDATA #IMPLIED
102   >
103
104   <!ELEMENT ref (#PCDATA)>
105   <!ATTLIST ref
106                      href        CDATA #REQUIRED
107   >
108
```

Anhang C Abkürzungen

Hier eine Liste der in dieser Arbeit verwendeten Abkürzungen:

ACID	*Atomicity, Consistency, Isolation, Durability*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
ASCII	*American National Standard Code for Information Interchange*
BBQ	*Blended Browsing and Querying*
DB	*Datenbank*
DBMS	*Database Management System*
DBS	*Datenbanksystem*
DOM	*Document Object Model*
DTD	*Document Type Definition*
EBCDIC	*Extended Binary Coded Decimal Interchange Code*
FDW	*Foreign Data Wrapper*
HTML	*Hypertext Markup Language*
HTTP	*Hypertext Transfer Protocol*
ISO	*International Organization for Standardization*
JAXP	*Java API for XML Prcessing*
LOB	*Large Object*
MIME	*Multipurpose Internet Mail Extensions*
MIX	*Mediation of Information using XML*
MOM	*Message-Oriented Middleware*
OO	*Objektorientiert*
OR-DBS	*Objekrelationales Datenbanksystem*
POP	*Presentation-Oriented Publishing*
RPC	*Remote Procedure Call*
RSS	*Rich Site Summary*
SAX	*Simple API for XML*

SGML	*Standard Generalized Markup Language*
SHARX	
SOAP	*Simple Object Access Protocol*
SQL	*Simple Query Language*
UCS	*Universal Character Set*
URI	*Uniform Resource Identifier*
UTF	*Unicode Transformation Format*
W3C	*World Wide Web Consortium*
WWW	*World Wide Web*
XMAS	*XML Matching and Structuring*
XML	*Extensible Markup Language*
XSL	*Extensible Stylesheet Language*
XSLT	*XSL Transformations*

Anhang D Lizenzen

Da Teile dieser Arbeit auf bereits existierenden Implementierungen von Spezifiktationen und Standards aufbauen, sind im Folgenden die Lizenzen der abgedruckt, die dabei berührt werden.

D.1 The XML:DB Initiative Software License, Version 1.0

Copyright © 2000-2001 The XML:DB Initiative. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

   "This product includes software developed by the XML:DB Initiative (`http://www.xmldb.org/`)."

   Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The name "XML:DB Initiative" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact `info@xmldb.org`.
5. Products derived from this software may not be called "XML:DB", nor may "XML:DB" appear in their name, without prior written permission of the XML:DB Initiative.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABI-

LITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the XML:DB Initiative. For more information on the XML:DB Initiative, please see <http://www.xmldb.org/>.

D.2 The Apache Software License, Version 1.1

Copyright © 1999 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

   "This product includes software developed by the Apache Software Foundation (http://www.apache.org/)."

   Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.
4. The names "Xalan" and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache", nor may "Apache" appear in their name, without prior written permission of the Apache Software Foundation.

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This software consists of voluntary contributions made by many individuals on behalf of the Apache Software Foundation and was originally based on software copyright © 1999, Lotus Development Corporation., http://www.lotus.com/. For more information on the Apache Software Foundation, please see <http://www.apache.org/>.

D.3 W3C IPR SOFTWARE NOTICE

Copyright © 2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

The DOM bindings are published under the W3C Software Copyright Notice and License. The software license requires "Notice of any changes or modifications to the W3C files, including the date changes were made." Consequently, modified versions of the DOM bindings must document that they do not conform to the W3C standard; in the case of the IDL binding, the pragma prefix can no longer be 'w3c.org'; in the case of the Java binding, the package names can no longer be in the 'org.w3c' package.

Note: The original version of the W3C Software Copyright Notice and License could be found at http://www.w3.org/Consortium/Legal/copyright-software-19980720

Copyright © 1994-2000 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/

This W3C work (including software, documents, or other related items) is being provided by the copyright holders under the following license. By obtaining, using and/or copying this work, you (the licensee) agree that you have read, understood, and will comply with the following terms and conditions:

Permission to use, copy, and modify this software and its documentation, with or without modification, for any purpose and without fee or royalty is hereby granted, provided that you include the following on ALL copies of the software and documentation or portions thereof, including

modifications, that you make:

1. The full text of this NOTICE in a location viewable to users of the redistributed or derivative work.
2. Any pre-existing intellectual property disclaimers, notices, or terms and conditions. If none exist, a short notice of the following form (hypertext is preferred, text is permitted) should be used within the body of any redistributed or derivative code: "Copyright (c) [$date-of-software] World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. http://www.w3.org/Consortium/Legal/"
3. Notice of any changes or modifications to the W3C files, including the date changes were made. (We recommend you provide URIs to the location from which the code is derived.)

THIS SOFTWARE AND DOCUMENTATION IS PROVIDED "AS IS," AND COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE SOFTWARE OR DOCUMENTATION WILL NOT INFRINGE ANY THIRD PARTY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

COPYRIGHT HOLDERS WILL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY USE OF THE SOFTWARE OR DOCUMENTATION.

The name and trademarks of copyright holders may NOT be used in advertising or publicity pertaining to the software without specific, written prior permission. Title to copyright in this software and any associated documentation will at all times remain with copyright holders.

Literaturverzeichnis

[ABD+90] M. P. Atkinson, F. Bancilhon, D. DeWitt, K. R. Dittrich, D. Maier, S. B. Zdonik:
The Object-Oriented Database System Manifesto.
In ACM SIGMOD Conference Atlantic City, New Jersey, USA, 1990

[ABS00] S. Abiteboul, P. Buneman, D. Suciu:
Data on the Web – From Relations to Semistructured Data and XML.
Morgan Kaufmann Publishers, San Francisco, California 2000

[ASCII86] American National Standards Institiute:
7-Bit American National Standard Code for Information Interchange.
ANSI X3.4-1986 (R1992), Washington, DC 1986

[BGL+99] C.K. Brau, A. Gupta, B. Ludäher, R. Marciano, Y. Papakonstantinou, P. Velikhov, V. Chu:
XML-Based Information Meditiation with MIX.
In ACM SIGMOD Conference, Philadelphia, Pennsylvania, USA, 1999

[Cod69] E.F. Codd:
A relational Model of Data for Large shared Data Banks
In Communications of the ACM Vol. 13 No. 6 1970, 377-387

[Con00] D. Conoly:
Extensible Markup Language (XML) – Activity Statement,
`http://www.w3.org/XML/Activity/`

[DHN+99] D. DeWitt, Gang He, J. Naughton, J. Shanmugasundaram, C. Zhang:
Relational Databases for Querying XML Documents: Limitations and Opportunities
In VLDB Conference, Edinburgh, Scotland 1999

[DOM01] World Wide Web Consortium:
Document Object Model (DOM) Level 3 Core Specification, Version 1.0, W3C Working Draft
`http://www.w3c.org/TR/DOM-Level-3-Core/`, 2001

[EBCDIC] Industrial Business Machines:
Extended Binary Coded Decimal Interchange Code.
`http://www-1.ibm.com/servers/eserver/iseries/clientaccess/codepages/`

[FK99] D. Florescu, D. Kossmann:
A Performance Evaluation of Alternative Mapping Schhemes for Storing XML Data in a Relational Database.
In IEEE Data Engineering Bulletin 22:3, 1999, 27-34

[FFK+98] M. Fernández, D. Florescu, J. Kang, A. Levy, D. Suciu:
Catching the Boat with Strudel: Experiances with a Web-Site Management System.
In ACM SIGMOD Conference, Seattle, Washington, USA, 1998

[FFL+00] M. Fernández, D. Florescu, A. Levy, D. Suciu:
Declarative specification of web sites with Strudel.
In The VLDB Journal Volume 9:1, Springer 2000,38-55

[Fle01] M. Flehmig:
Data-Driven, XML-Based Web Management in Highly Personalized Environments.
In Proc. of the Int. Workshop on Information Integration on the Web, Rio de Janeiro, Brazil, 2001

[GR93] Gray, J. und Reuther, A.
Transaction Processing: Concepts and Techniques
Morgan Kaufmann, 1993

[HTML99] World Wide Web Consortium:
HTML: Hypertext Markup-Language 4.01 Specification, W3C Recommendation,
`http://www.w3.org/TR/html4/`, 1999

[ISO8859-1] International Organization for Standardization:
Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1
ISO/IEC 8859-1 JTC 1/SC 2, Switzerland 1998

[ISO8859-15] International Organization for Standardization:
Information technology – 8-bit single-byte coded graphic character sets – Part 15: Latin alphabet No. 9
ISO/IEC 8859-15 JTC 1/SC 2, Switzerland 1999

[ISO8879] International Organization for Standardization:
Information Processing – Text and office systems – Standard Generalized Markup Language (SGML)
ISO 8879 JTC 1/SC 34, Switzerland 1986

[ISO9075-9] International Organization for Standardization:
Database Language SQL – Part 9: SQL MED
ISO/IEC JTC 1/SC 21/WG 3 Database, Switzerland 2001

[ISO10646-1] International Organization for Standardization:
Information technology – Universal Multiple-Octet Coded Character Set (UCS) – Part 1: Architecture and Basic Multilingual Plane
ISO/IEC 10646-1 JTC 1/SC 2, Switzerland 2000

[JAXP01] Scott Boag, James Duncan Davidson, Rajiv Mordiani:
Java API for XML Prcessing, Version 1.1 Final Release
`http://java.sun.com/xml/download.html`, 2001

[RFC2616] *HTTP: Hypertext Transfer Protocol 1.1*,
Network Working Group – Request for Comments 2616,
`ftp://ftp.isi.edu/in-notes/rfc2616.txt`, 1999

[SAX01] D. Megginson:
SAX: History and Contributors
`http://www.megginson.com/SAX/SAX1/history.html`, 2001

[SAX02] D. Megginson:
SAX: Frequently-Asked Questions
`http://www.megginson.com/SAX/faq.html`, 2001

[SAX03] D. Megginson:
What is an Event-Based Interface
`http://www.megginson.com/SAX/event.html`, 2001

[Sic98] D. Suciu:
Semistructured Data and XML.
In 5th International Conference of Foundations of Data Organization (FODO'98), Kobe, Japan 1998

[SOAP00] World Wide Web Consortium:
SOAP: Simple Object Access Protocol 1.1, W3C Note,
`http://www.w3c.org/TR/SOAP/`, 2000

[SQL92] Digital Equipment Corporation:
Database Language SQL
ISO/IEC 9075:1992, Maynard, Massachusetts 1992

[Sta01] K. Staken:
dbXML DevelopersGuide 0.5
`http://www.dbxml.org/docs/DevelopersGuide.html`, 2001

[TRS97] M. Tork Roth, P. M. Schwarz:
Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources
In VLDB Conference, Athens, Greece 1997

[Wie92] G. Wiederhold:
Mediators in the architectures of Future Information Systems.
In IEEE Computer Magazine Volume 25:3, 1992, 38-49

[XLink01] World Wide Web Consortium:
XML Linking Language (XLink) Version 1.0. W3C Recommendation,
`http://www.w3c.org/TR/REC-xlink-20010627/`, 2000

[XML00] World Wide Web Consortium:
XML: Extensible Markup Language 1.0 (Second Edition). W3C Recommendation,
`http://www.w3c.org/TR/REC-xml-20001006/`, 2000

[XML:DB00] XML:DB – Initiative for XML Databases:
XML:DB Initiative: Enterprise Technologies for XML Databases,
`http://www.xmldb.org/`, 2000

[XML:DB01] XML:DB – Initiative for XML Databases:
XML:DB Initiative: Questions & Answers,
`http://www.xmldb.org/faqs.html`, 2001

[XPath99] World Wide Web Consortium:
XML Path Language (XPath) 1.0, W3C Reconedation,
`http://www.w3c.org/TR/xpath/`, 1999

[XUpdate00] XML:DB – Initiative for XML Databases:
XUpdate – XML Update Language, XML:DB Working Draft,
`http://www.xmldb.org/xupdate/xupdate-wd.html`, 2000