Diplomarbeit Nr.2144

Einsatz einer XML-Anfragesprache für Transformationszwecke

Sascha Opletal

Betreuer: Uwe Heinkel
Institut für Parallele und Verteilte Systeme
Abteilung Anwendersoftware
Universität Stuttgart

Inhaltsverzeichnis

Kurzfassung **1**

1 Einführung **3**
1.1 Integration und Kopplung von Informationssystemen . . . 3
1.1.1 XML: Informationsaustausch . . . 4
1.1.2 Ein Beispielszenario . . . 6
1.2 SIES: Aufbau und Funktion . . . 9
1.2.1 Der Abhängigkeitsmanager . . . 9
1.2.2 Der Propagationsmanager . . . 11
1.2.3 Das Repository . . . 11
1.2.4 Message-Queues und Adaptoren . . . 12

2 Anforderungen **13**
2.1 Transformationen im SIES . . . 13
2.2 Strukturänderungen . . . 16
2.2.1 Filterung . . . 16
2.2.2 Anordnung/Sortierung . . . 17
2.2.3 Änderung der Modellierung . . . 17
2.2.4 Ersetzungen/Strukturänderungen . . . 19
2.2.5 Integration mehrerer Dokumente . . . 20
2.3 Datenänderungen . . . 20
2.3.1 Zusammenführung/Aufteilung . . . 21
2.3.2 Aggregationsfunktionen . . . 22
2.3.3 Typkonvertierungen . . . 22
2.3.4 Umrechnungen . . . 23
2.3.5 Selbstdefinierte Funktionen . . . 24
2.3.6 Verknüpfung von Daten . . . 24
2.3.7 Einfachere Verwendung als XSLT . . . 24
2.4 Methoden zur Durchführung von Transformationen . . . 25
2.4.1 Lokalisierung von Elementen . . . 25
2.4.2 Variablenbindung . . . 25
2.4.3 Bedingungen . . . 25
2.4.4 Iterationen . . . 26
2.5 Informationsextraktion mit Anfragesprachen . . . 26

2.6 Integrationsanforderungen . . . 26
2.6.1 Plattform . . . 26
2.6.2 Zugriff auf Dokumente . . . 27
2.6.3 Integration verschiedener Datenquellen . . . 27
2.6.4 Anfrageschnittstelle . . . 27
2.6.5 Externe Funktionen . . . 27
2.7 Zusammenfassung . . . 27

3 Überblick XML-Verarbeitung 29
3.1 Transformationssprachen . . . 29
3.1.1 XSL(T) . . . 30
3.1.2 XPath . . . 31
3.1.3 Transformationsbeispiel in XSLT: Materialstrom . . . 32
3.2 Anfragesprachen . . . 35
3.2.1 Datenmodelle . . . 36
3.2.2 XML Query Algebra . . . 38
3.2.3 Ursprung: SQL/OQL . . . 39
3.2.4 XML-QL . . . 40
3.2.5 YATL . . . 41
3.2.6 Lorel . . . 42
3.2.7 XQL . . . 42
3.2.8 Quilt . . . 43
3.2.9 XQuery . . . 44
3.2.10 XML-GL . . . 45
3.3 Weitere Sprachen . . . 47
3.4 Zusammenfassung und Bewertung . . . 48

4 Transformation mit XML-Anfragesprachen 51
4.1 Analysemethoden . . . 52
4.1.1 Addressierung des Ausgangsdokuments . . . 52
4.1.2 Navigation und Selektion . . . 54
4.1.3 Analyse ohne Strukturwissen . . . 56
4.1.4 Bedingungen . . . 56
4.1.5 Quantoren . . . 57
4.2 Transfermethoden . . . 57
4.2.1 Variablenbindung . . . 57
4.2.2 Verwendung: Variablenmodifikation/-Auflösung . . . 58
4.2.3 Iteration . . . 59
4.2.4 Typerzwingung . . . 59
4.2.5 Skolem-Funktionen . . . 60
4.2.6 Einschachtelung von Anfragen . . . 60
4.3 Generierung von XML-Strukturen . . . 60
4.3.1 Aufbau von Strukturen . . . 61
4.3.2 Update-Language . . . 62

4.4 Transformationsoperationen 62
4.4.1 Strukturänderungen 62
4.4.2 Datenänderungen 67
4.5 Weitere Methoden 70
4.5.1 Erweiterbarkeit: Externe Funktionen 70
4.5.2 Erweiterbarkeit: Namensräume 70
4.6 Eignungsbewertung 71
4.7 Transformationen mit XQuery 72
4.7.1 Löschen von Information 73
4.7.2 Modellierungsänderung 73

5 Implementation **74**
5.1 XQuery-Prozessoren: Qexo und Saxon 74
5.1.1 Erweiterung der Funktionen von Qexo 75
5.1.2 Erweiterung der Funktion von Saxon 76
5.1.3 Standalone Aufruf Saxon und Qexo 76
5.1.4 Konfiguration von Saxon 76
5.2 Integration in SIES 77
5.2.1 Anpassung des Abhängigkeitsmanagers 78
5.2.2 Erweiterung von XRL+ 78
5.2.3 Datenhaltung 80
5.2.4 Anpassung des Propagationsmanagers 80
5.3 Realisierung von SIES-Transformationen in XQuery 81
5.3.1 Umrechnung 81
5.3.2 Konvertierung und Anreicherung 82
5.3.3 Aggregation 83
5.3.4 Integration 84
5.3.5 Zyklisches Inkrementieren 84

6 Zusammenfassung und Ausblick **85**

Anhang A: Syntax von XQuery **86**

Anhang B: XRL+ Schema **94**

Literaturverzeichnis **97**

Kurzfassung

Im Rahmen des Sonderforschungsbereichs 467 - Wandlungsfähige Unternehmensstrukturen für die variantenreiche Serienfertigung - beschäftigt man sich mit der Fragestellung, wie sich in Zukunft Fertigungsunternehmen permanent an die sich immer schneller verändernden Marktverhältnisse anpassen können. Das Ziel des Teilprojekts A5 (Modellierung von und Exploration in komplexen Unternehmensinformationen) des SfBs ist die informationstechnische Unterstützung wandlungsfähiger Unternehmen, die das Konzept der Selbstorganisation anwenden. Dafür ist eine Infrastruktur notwendig, die mit Hilfe von Änderungspropagation die Daten eines wandlungsfähigen Unternehmens integriert. Dabei werden Datenänderungen erfasst und durch entsprechende Aufbereitung für die von der Änderung abhängigen Informationssysteme verfügbar gemacht.

Diese Infrastruktur, genannt SIES, setzt sich aus drei Komponenten zusammen: das Repository, der Abhängigkeitsmanager und der Propagationsmanager. Das Zusammenwirken dieser Komponenten erlaubt den Austausch und die Kommunikation zwischen heterogenen Informationssystemen. Eine wichtige Rolle in dieser Infrastruktur spielt die Transformation von Änderungsinformationen, so dass diese an die Anforderungen der Zielsysteme angepasst sind. Die Transformation der zu propagierenden Änderungsinformationen, die in XML repräsentiert werden, erfolgt mit XSLT.

Ziel dieser Diplomarbeit war die Untersuchung, inwiefern eine XML-Anfragesprache für den Zweck der Transformation von Änderungsinformationen eingesetzt werden kann. Hierzu wurden die auf dem Markt befindlichen XML-Anfragesprachen auf Eignung für den Einsatz als Transformationssprache untersucht. Es wurden zunächst Anforderungen an die Funktionalität von Anfragesprachen zusammengestellt, die das Ziel einer XML-Transformation und Produktion unterstützen sollen. Die Bestimmung der Anforderungen wurde unter besonderer Berücksichtigung der Erfordernisse des SIES durchgeführt. Ein Vergleich in Frage kommender und verfügbarer Anfragesprachen sollte klären, ob eine XML-Anfragesprache die gestellten Anforderungen erfüllen kann und ob eine Integration in das SIES möglich ist.

Es wurde festgestellt, dass XQuery, der gegenwärtige Vorschlag des W3C zur Standardisierung von Anfragesprachen, die meisten der gegebenen Anforderungen erfüllt. Die Umsetzung einer Reihe von Skripten der bisher eingesetzten Transformationssprache XSLT nach XQuery soll die Realisierbarkeit von Transformationen mittels der Anfragesprache zeigen. Die geplante Integration in das SIES ergab eine Reihe weiterer zu beachtender Anforderungen, die sich auf die Struktur des Systems beziehen. Hierzu wurden zwei geeignete XQuery-Implementationen, Saxon und Qexo untersucht. Aufgrund der gestellten Anforderungen der Implementierung erschien Saxon gegenüber Qexo als

geeigneter für den Einsatz im SIES.

Den Abschluß der Arbeit bildet eine detaillierte Beschreibung, wie die Anfragesprache XQuery als Komponente in das SIES integriert wurde. Das Ziel der Integration war, dass Transformationen sowohl in XSLT als auch in der ausgewählten XML-Anfragesprache formuliert und bearbeitet werden können. Hierdurch sollte eine Flexibilisierung der möglichen Transformationsbeschreibungen erreicht werden.

1 Einführung

Zur Datenhaltung in Unternehmen werden verschiedene Informationssysteme (IS) eingesetzt. Auf dem Markt gibt es eine Vielzahl solcher Informationssysteme, die unterschiedlichen Anforderungen an Modellierung und Abfrage von Daten genügen (z.B. relationale und objektorientierte Systeme).

In einem Unternehmen kann es vorkommen, dass in verschiedenen Abteilungen und Bereichen unterschiedliche Informationssysteme genutzt werden. Die Zusammensetzung eines Unternehmens spiegelt sich direkt in Struktur und Organisation der Datenhaltung wieder, die sich in der Unabhängigkeit und Heterogenität der verwendeten Informationssysteme zeigt. Dieses Datenhaltungsumfeld ist meist historisch gewachsen und kann entstehen, wenn jeweils auf einen Unternehmensbereich spezialisierte Informationssysteme zum Einsatz kommen.

Diese Situation kann aus zweckgebundenen Gründen entstehen, wie etwa aufgrund von speziellen Funktionen oder Datenmodellen die benötigt werden und nur in einer bestimmten Software vorhanden sind. Weiterhin kann es vorkommen, dass Unternehmenseinheiten bei einem Upgrade oder Systemwechsel aus Zeit und Kostengründen ihre Daten nicht migriert haben. Ein weiteres Hindernis besteht, auch wenn unternehmensweit die gleichen Informationssysteme eingesetzt werden, in unterschiedlichen Modellierungen der Daten. Ist die Modellierung der Daten zwischen zwei IS oder angeschlossenen Maschinen verschieden, kann der Austausch nur mittels geeigneter Anpassung der Daten erfolgen. Diese Situation ist auch aus dem B2B-Bereich (Business-to-Business) bekannt, der sich mit der Realisierung des Datenaustausches zwischen Unternehmen beschäftigt.

Das Ziel des SfB 467 ist es, die Datenbestände der einzelnen Informationssysteme sinnvoll zu verknüpfen und sie dennoch als autonome Systeme beizubehalten. Dabei entsteht ein Verbund von Informationssystemen, der durch eine automatische Änderungspropagation nach außen hin als ein funktionales Gesamtsystem wahrgenommen wird.

1.1 Integration und Kopplung von Informationssystemen

Um eine Integration von Informationssystemen mit verschiedenen Architekturen und Datenmodellen zu erreichen, braucht man eine Schnittstelle, durch die ein Datenaustausch ermöglicht wird. Eine Vorgehensweise ist es, eine globale Datenstruktur zu verwenden, die von allen beteiligten Informationssystemen unterstützt wird. Dies kann in einer begrenzten Zahl von Systemen gelingen, ist im allgemeinen Fall jedoch schwierig zu realisieren.

Datenänderungen in einer Datenbank können Auswirkungen auf Daten haben, die in anderen Systemen gespeichert sind. Aufgrund der verschiedenen Repräsentationen der

Information in den Datenbanken, muss eine Transformation vom Format der Ausgangsdatenbank in das der Zieldatenbank erfolgen. Dabei sind etwaige Datenabhängikeiten zu berücksichtigen, so dass Änderungsinformation an die richtigen Datenbanken weitergeleitet wird.

Der einfachste Fall ist die direkte Abhängigkeit, bei der ein Zielsystem in unmittelbarer Abhängigkeit eines Quellsystems steht. Durch eine Verkettung mehrerer voneinander abhängiger Systeme können transitive Abhängigkeiten entstehen. Es kann auch vorkommen, dass die Eingangsdaten einer Datenbank von mehreren Quellen abhängig sind und Teile der Ausgangsinformation aktiv zusammengestellt werden müssen.

Die Integration der Systeme kann eng oder lose organisiert werden. Unter einer engen Kopplung wird die direkte Verbindung von Datenbanken des gleichen Typs oder mit fest definierten Austauschregeln verstanden. Für jeden neu anzubindenden Typ muss eine neue Konvertierungsrichtlinie festgelegt werden. Es müssen dazu n*n Transformationen definiert werden.

Eine lose Kopplung beinhaltet die Überführung eines Eingabeformats in eine unabhängige Zwischenrepräsentation. Dabei übernimmt eine Vermittlungsschicht die Konvertierung der Daten mit Hilfe von Ein- und Ausgangstransformationen. Es werden nur noch 2*n Transformationsrichtlinien benötigt, je n für Ein- und Ausgabe.

1.1.1 XML: Informationsaustausch

Die Sprache XML (eXtensible Markup Language) ist ein vom W3C (World-Wide-Web-Consortium) definierter, plattformunabhängiger Standard zum Austausch von Dokumenten. XML dient als Metasprache zur Modellierung und Darstellung der Daten innerhalb eines solchen Dokuments.

XML ist eine vereinfachte Form von SGML (Standard Generalized Markup Language), die in den 80er Jahren zum Austausch von Dokumenten entwickelt wurde. Es stellte sich jedoch schnell heraus, das SGML hierzu ungeeignet ist, da es in der Anwendung zu komplex ist (vgl. [Eck 00]). HTML (HyperText Markup Language), die Sprache zur Beschreibung der Darstellung von Informationselementen auf Web-Seiten, wurde in SGML definiert. Der Markup-Satz von HTML ist jedoch statisch und kann nicht erweitert werden. Daher wurde XML geschaffen, um Daten frei zu beschreiben und austauschbar zu machen.

XML-Dokumente sind hierarchisch aufgebaut. Die Struktur-Elemente eines XML-Dokuments nennt man Tag. Ein Tag ist ein Bezeichner, der von spitzen Klammern umschlossen wird (s.Abb. 1.1). Zu jedem öffnenden Tag muss es einen abschliessenden geben. Mit XML kann man eigene Markup-Tags und Dokumentenstrukturen definieren und somit das Modell für die Daten erzeugen.

```
<siesdata>...</siesdata>
```

Abb. 1.1: Beispiel eines Tag

Das kleinste gültige Objekt eines XML-Dokuments ist der leere Tag. Als Sonderform kann beim leeren Tag durch das Fehlen des Elementinhalts ein schliessender Tag einge-

spart werden, indem ein abschliessender Schrägstrich hinzugefügt wird (s.Abb. 1.2).

```
<beispiel/>
```

Abb. 1.2: Der leere Tag

Tags umschliessen Informationseinheiten wie Textdaten oder weitere Tags. Attribute werden innerhalb eines Tags annotiert und bieten eine weitere Möglichkeit zum Ablegen von Information (s.Abb.1.3). XML-Dokumente besitzen einen einzelnen Wurzelknoten, in den die anderen Tags hierarchisch eingebettet sind.

```
<siesdata attributA="Wert" attributB="Wert"/>
```

Abb. 1.3: Attribute in einem Tag

Zur Validierung besitzt XML DTDs (Document Type Definitions) , in denen die Struktur eines Dokumentes abgelegt wird. Ein DTD-Validator kann anhand einer DTD überprüfen, ob ein Dokument gültig und syntaktisch wohlgeformt ist.

Als Beispiel einer XML-Kodierung sei in Abb.1.4 ein Ausschnitt aus einer Datensammlung dargestellt, die häufig bei der Evaluation von Anfragesprachen verwendet wird ([Fer 99]). Der Ausschnitt zeigt eine mögliche Modellierung der Daten des Sortiments eines fiktiven Buchladens. Ein XML-Dokument wird durch eine Zeile mit Information über verwendete XML-Version und Zeichensatz eingeleitet. Kommentare können in XML durch die Zeichenfolge <!– –> eingebettet werden. In diesem Beispiel ist <bib> das Wurzelelement, unter das mehrere Bücherelemente <book> untergeordnet werden. Für diese Daten wird anschließend in einem Beispielszenario eine mögliche Transformation gezeigt.

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

<!-- XML Beispiel zur Repraesentation von Daten -->

<bib>

    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>

    [...weitere book-Elemente...]
</bib>
```

Abb. 1.4: Beispielkodierung von XML-Daten

Man spricht bei XML-Dokumenten häufig auch von semi-strukturierten Daten. Die Haupteigenschaft semi-strukturierter Daten ist, dass die Daten selbstbeschreibend ab-

gelegt sind und nicht von einem festen Schema definiert werden. Da jedoch durch DTDs und XML-Schemas eine Struktur der XML-Daten definiert werden kann, ist es möglich auch streng einem Schema folgende Daten in XML darzustellen [Abi 00]. Auch kann durch das Schema ein semi-strukturiertes Dokument erzeugt werden, indem freigestellt wird, ob die Existenz bestimmter Elemente optional oder vorgeschrieben ist.

Dies ist im Gegensatz zu strukturierten Modellen wie dem Relationalen zu sehen, bei dem nicht vorhandene Daten durch einen expliziten "null"-Wert ausgedrückt werden müssen. XML selbst unterstützt keine Typisierung der Datenfelder. Es werden strukturell nur Elemente und Attribute unterschieden. Der textuelle Inhalt wird als CDATA bezeichnet.

Um XML-Strukturen innerhalb eines CDATA-Blocks abzulegen, kann eine Zeichenfolge explizit als CDATA annotiert werden (s.Abb. 1.5). Weitergehende Typisierungen und Validierungen sind nur durch eine externe Typprüfung mittels DTD oder Schema möglich.

```
<![CDATA[<greeting>Hello, world!</greeting>]]>
```

Abb. 1.5: Typisierung von CDATA

1.1.2 Ein Beispielszenario

Unter einer Transformation wird hier die Änderung der Darstellung von Information verstanden. Insbesondere umfasst dies Änderungen auf der strukturellen Ebene, um etwa bei zwei verschiedenen Arten der Modellierung die Information austauschbar zu machen.

Bei einer Transformation bleibt der Informationsgehalt insofern erhalten, dass in Abhängigkeit des Informationsbedarfs manchmal nur die für einen Empfänger relevante Teilinformation weitergegeben wird. Eine gängige Anwendung einer Transformation ist die Unterstützung des Datenaustausches zweier Unternehmen. Das folgende Szenario soll es einer Buchhandlung ermöglichen bei einem Verlag Bücher zu bestellen. Die jeweils verwendeten Datenformate werden im Folgenden gezeigt (s.Abb. 1.6 u. 1.7).

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

<bib>
    <book year="1992">
        <title>Advanced Programming in the Unix environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
    </book>
</bib>
```

Abb. 1.6: Daten der Buchhandlung

Um den Austausch zu ermöglichen muss eine Transformation des Datenformats der Buchhandlung in das Format des Verlags erfolgen. Information, die im Zielformat nicht benötigt wird, kann bei der Transformation ausgefiltert und somit unterdrückt werden. Eine Schwierigkeit ist jedoch der Umgang mit Informationen, die im jeweiligen Zielformat benötigt werden, im Ausgangsformat aber nicht in der gewünschten Form vorhanden sind und eventuell gezielt durch z.B. Aggregationsfunktionen oder Kontextwissen erzeugt werden müssen.

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

<booklist>
       <book price="20 EUR">
          <title>Advanced Programming in the Unix environment</title>
          <author>W. Stevens</author>
          <style>paperback</style>
       </book>
</booklist>
```

Abb. 1.7: Daten des Verlags

Zu einem Austausch müssen die Modellierungsrichtlinien der Daten des Partners bekannt sein (durch DTD/XML Schema). Dies kann ermöglicht werden, indem die Richtlinien der beteiligten Parteien untereinander ausgetauscht werden.

Eine höhere Integrationsfähigkeit der Daten kann durch eine einheitliche Richtlinie erreicht werden, die zum Austausch verwendet wird. Dies setzt voraus, dass sich die beteiligten Parteien gemeinsam Gedanken über die Modellierung der Daten machen.

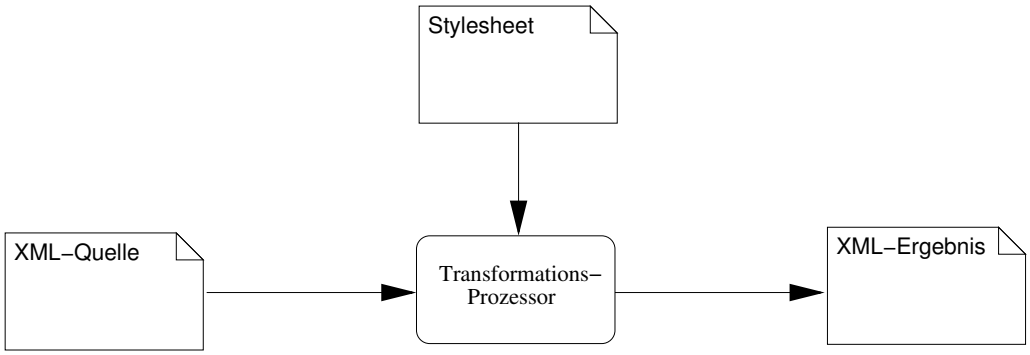


Abb. 1.8 : Ausführung einer Transformation

Die Transformation von Datenstrukturen eines Schemas in ein anderes erfolgt durch einen Transformationsprozessor. Dem Transformationsprozessor werden die zu verarbeitenden Daten und eine zugehörige Transformationsanweisung (bei XSLT genannt Stylesheet) übergeben (s.Abb. 1.8).

Der Transformationsprozessor gibt schließlich das Ergebnis der Transformation in Form eines XML-Dokuments zurück, so dass die enthaltene Information in eine Zieldatenbank integriert werden kann. Die Eingangsdaten werden in dieser Arbeit im Zusam-

menhang mit einer Transformation auch Quellstruktur, die Ausgangdaten Zielstruktur genannt.

Die verwendete Transformationssprache bestimmt gleichzeitig die möglichen Ausgabeformate, d.h. das Ergebnis kann je nach Sprache im XML-Format oder in einem beliebigen anderen dargestellt werden. Um jedoch eine Transformation auf XML-Daten durchzuführen, muss es einem Transformationsprozessor möglich sein, XML zu generieren.

1.2 SIES: Aufbau und Funktion

Das Stuttgarter Informations- und Explorations-System (SIES) wurde entworfen, um eine heterogene Sammlung von Informationssystemen zu einem funktionalen Gesamtsystem zu verbinden. Der Aufbau des SIES ist in Abb. 1.9 dargestellt.

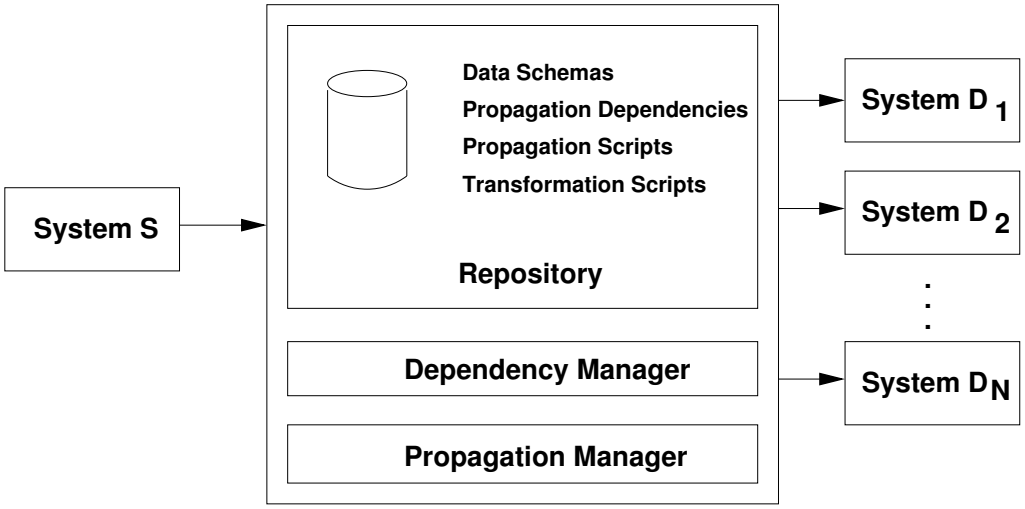


Abb. 1.9 : Aufbau des SIES (aus [Con 03])

Das SIES besteht aus drei Teilen, dem Repository, dem Abhängigkeits- und dem Propagations-Manager. Im Repository sind Abhängigkeitsdefinitionen, Daten-Schemata und Skripte zur Transformation von Daten abgelegt. Der Propagationsmanager sorgt dafür, dass abgefangene Datenänderungen in einem Informationssystem entsprechend aufbereitet an abhängige Zielsysteme weitergeleitet werden. Durch den Abhängigkeitsmanager wird der Datenfluss zwischen den Informationssystemen definiert. Dabei wird im Repository die zur Transformation und Filterung der Daten benötigte Information abgelegt, die bei der Bearbeitung der Informationsströme der Datenänderungen vom Propagationsmanager ausgwertet wird.

1.2.1 Der Abhängigkeitsmanager

Der Abhängigkeitsmanager dient zur Definition der informationellen Zusammenhänge, die eine Datenabhängigkeit beteiligter Systeme beeinhaltet, daher welche Quellsysteme auf welche Weise mit Zielsystemen zu verbinden sind. Dazu wird zunächst Schema-Information bereitgestellt, um sowohl Quell- als auch Zieldaten zu validieren. Anschliessend wird der Fluss der Daten modelliert. Dazu ist einerseits Routing-Information nötig, um den Weg der Daten vom Quell- zum Zielsystem zu beschreiben. Zum Anderen wird auch die benötigte Art der Umwandlung der Daten festgehalten, um eine Schema-konforme Integration in den Datenbestand des Zielsystems zu ermöglichen. Die Modellierung der Daten in der Zieldatenbank wird in durch speziell abgestimmte Transformationsanweisungen nachgebildet.

Abhängigkeitsmodellierung mit XRL+

Die Exchangeable Routing Language (XRL) dient zur Beschreibung von Prozessabläufen. Dabei wird der Workflow in Form von Dokumenten zwischen zwei Geschäftspartnern durch Routing-Information festgehalten [Aal 00]. Die Abhängigkeiten der Informationssysteme werden durch die modifizierte Sprache XRL+ dargestellt, die im Rahmen der Konzeption des Propagationsmanagers an die speziellen Anforderungen der Propagation angepasst wurde [Ker 01]. XRL+ bedient sich der XML-Syntax (s.Abb. 1.10).

```
<?xml version="1.0" encoding="UTF-8"?>
   <!DOCTYPE ROUTE SYSTEM "xrl.dtd">}

<xrl:ROUTE created_by="meinechr" creation_date="Oct 10, 2002" id=""
           xmlns:xrl="http://informatik.uni-stuttgart.de/XRL">
    <xrl:SEQUENCE>
        <xrl:WAIT sync="1">
            <xrl:MESSAGE_EVENT initial="yes" schema="Production Order"
                 system="Order Management" xml_out="om_data"/>
        </xrl:WAIT>
        <xrl:TRANSFORM xslt="po2tm" xml_in="om_data" xml_out="tm_data"/>
       <xrl:PROPAGATE schema="Transportation Matrix"
                      system="Facility Layout Design" xml="tm_data"/>
    </xrl:SEQUENCE>
</xrl:ROUTE>
```

Abb. 1.10 : Abhängigkeitsbeschreibung in XRL+

Es wurden unnötige Konstrukte wie <ANY_SEQUENCE> und <CHOICE> verworfen, da sie bei der Datenpropagation keinen Zweck erfüllen. Andere Mechanismen der Modellierung, wie die zur Synchronisation wurden vereinfacht. Ein aktuelles Schema von XRL+, bei dem auch die Ergebnisse der Implementation berücksichtigt sind, ist im Anhang B beigefügt.

Zentrale Elemente des angepassten XRL+ sind die neu definierten Elemente <FILTER>, <TRANSFORM>,<PROPAGATE> und <MESSAGE_EVENT> [Mit 03]. Die FILTER-Anweisung dient der Vorauswahl zu berücksichtigender Information und wird durch eine XPath-Bedingung realisiert (s. Kap. 3). Die TRANSFORM-Anweisung beschreibt die durchzuführende Umwandlung der Daten des Quellsystems in eine für das Zielsystem verständliche und integrierbare Form. Die Anweisungen PROPAGATE und MESSAGE_EVENT dienen der Behandlung der Ein- und Ausgangswarteschlangen, in denen zu transformierende Daten warten, um vom Propagationsmanger oder dem Zielsystem bearbeitet zu werden.

Eine Abhängigkeitsbeschreibung des SIES in XRL+ ist in Abb. 1.10 gezeigt. In diesem Abhängigkeits-Beispiel wird auf ein Ereignis der Quelle “Order Management” mit Namen “om_data” gewartet. Die Daten der Quelle “om_data” werden innerhalb der TRANSFORM-Anweisung durch eine in XSLT (s. Kap. 3) beschriebene Transformation aufbereitet. Das anzuwendende XSLT-Skript trägt den Namen “po2tm”. Anschließend

wird das Ergebnis der Transformation an das Ziel "Facility Layout Design" weitergeleitet und in den dortigen Datenbestand integriert.

1.2.2 Der Propagationsmanager

Die Aufgabe des Propagationsmanagers ist es, Datenänderungen in einem IS einem anderen IS bekannt und verfügbar zu machen. Der Propagationsmanager erzeugt die Information für das zu koppelnde Zielsystem durch Auswahl und Anwendung relevanter Transformationsskripte aus dem Repository.

Änderungen in der Datenbasis eines angeschlossenen IS werden dem Propagationsmanager über einen speziellen Adaptor mitgeteilt. Es gibt die Möglichkeit, den Export eines IS mittels aktiver oder passiver Adaptoren auszuführen. Aktive Adaptoren prüfen selbsttätig auf Änderungen in einer Datenbasis und stellen die anfallenden Daten dem Propagationsmanager bereit. Passive Adaptoren hingegen müssen von der Datenbank bei einer Änderung benachrichtigt werden. Die Auswahl eines Adaptors hängt von den vorhandenen Schnittstellen eines Informationssystems ab.

Die Ausgangsdaten werden durch den Adaptor aus der Quell-Datenbank entnommen und in eine system- und plattformunabhängige XML-Struktur umgewandelt. Auf diese XML-Strukur wendet der Propagationsmanager ein, gemäß der Art der vorhandenen Abhängigkeit, ausgewähltes Transformationsskript an.

Der Propagationsmanager prüft die Informationssysteme auf Abhängigkeit zu den festgestellten Datenänderungen und sorgt für eine Propagation der relevanten Änderungen an die Zielsysteme. Hierzu verfügt der Propagationsmanager über zur Datenabhängigkeit passende Skripte, die die Eingangsdaten filtern, transformieren und routen können. Eine Transformation der Eingangsdaten ist nötig, damit sie in einem gültigen Format an das Zielsystem übergeben werden können. Gleichzeitig können durch die Transformation Optimierungen und Verknüpfungen verschiedener Datenströme vorgenommen werden. Um Constraints behandeln zu können verfügt der Propagationsmanager über eine Filtermöglichkeit, mit der Eingangsdaten auf das Erfüllen von Kriterien überprüft werden können. Der Propagationsmanager wartet auf das Eintreffen relevanter Nachrichten ihn einer Eingangswarteschlange zu einer ihm bekannten Datenabhängigkeit. Nach der Bearbeitung einer Nachricht schreibt er das Ergebnis der zugehörigen Transformation in eine Ausgangswarteschlange, aus der die Adaptoren der Zielsysteme die Daten entnehmen und weiterverarbeiten. Gegenwärtig werden Transformationen in der Sprache XSLT (s. Kap. 3) beschrieben. Ziel dieser Arbeit ist die zusätzliche Integration einer XML-Anfragesprache, um die Ausdrucksmächtigkeit einer Transformation zu erhöhen und zu flexibilisieren.

1.2.3 Das Repository

Das Repository dient der persistenten Vorhaltung der zugehörigen Skripte einer Datenabhängigkeit und bildet dadurch eine eigene Datenbank, da der Propagationsmanager zwischen den Informationssystemen steht und von diesen unabhängig ist. Im Repository werden Abhängigkeiten, Propagations- und Transformationsskripte bereitgestellt. Zu

den einzelnen Skripten werden zusätzlich Schemata zur Validierung der Daten gespeichert.

Darüberhinaus ist Information über das Zusammenwirken der einzelnen Informationselemente wie die Zugehörigkeit eines Schemas zu einem Skript oder die Verknüpfung von System- und Routing-Information (XRL+) im Repository abgelegt.

1.2.4 Message-Queues und Adaptoren

Um die Daten zwischen den Informationssystemen und dem Propagationsmanager zu vermitteln wird die Spezifikation des Java Message Service (JMS) eingesetzt. Die im SIES eingesetzte Implementation des JMS names J2EE, ermöglicht es, sogenannte Message-Queues zu erstellen, über die Nachrichten synchron und asynchron ausgetauscht werden können. Die in den Message-Queues gelagerten Nachrichten werden persistent gespeichert, so dass im Fehlerfall keine Information verloren geht.

Die Verbindung zwischen den Informationssystemen und den Message-Queues wird über die Adaptoren bewerkstelligt. Die Adaptoren wandeln die Daten der Systeme aus deren spezifischen Format nach XML und zurück, so dass der Propagationsmanager mit einem einheitlichen Repräsentationsformat arbeiten kann.

2 Anforderungen

Die Anforderungen an eine XML-Anfragesprache sind im Kontext des SIES zu sehen. Es soll durch eine Anfragesprache mindestens die gleiche Funktionalität der XML-Transformation erreicht werden, wie sie bereits durch das eingesetzte XSLT (s.Kap. 3) ermöglicht wird.

Das Ziel ist eine Erweiterung der Ausdrucksstärke des Propagationsmanagers. XML-Anfragesprachen wurden im Gegensatz zu den Transformationssprachen auf die Extraktion von Information hin entwickelt. Im Laufe der Zeit entstanden neue Anforderungen, die bei der ursprünglichen Konzeption des Propagationsmanagers noch nicht berücksichtigt wurden. Es wurden einige wünschenswerte Funktionalitäten zusammengestellt, die den Propagationsmanager bereichern sollen, aber nicht direkt der Transformation zuzuordnen sind. Zunächst werden die benötigten Transformationsfunktionen betrachtet, gefolgt von Integrationsanforderungen.

2.1 Transformationen im SIES

Im Folgenden sollen die für den Einsatz im Propagationsmanager benötigten Transformationsfunktionalitäten zusammengetragen werden. Hierzu werden zunächst ausgehend von den bisherigen Möglichkeiten zur Transformation, basierend auf XSLT, die grundlegend notwendigen Mechanismen herausgearbeitet. Dazu wurde u.a. die Funktionalität der bestehenden Transformationsskripten des Propagationsmanagers untersucht.

Das Ziel ist eine Erweiterung der Ausdrucksstärke des Propagationsmanagers durch die Verwendung einer XML-Anfragesprache. Im Laufe der Zeit entstanden neue Anforderungen, die bei der ursprünglichen Konzeption des Propagationsmanagers (siehe [Ker 01]) noch nicht berücksichtigt wurden. Dabei kristallisierten sich eine Reihe von Funktionen heraus, die in den folgenden Versionen des Propagationsmanagers unterstützt werden sollen. Es wurden zusätzlich einige wünschenswerte Funktionalitäten aufgeworfen, die den Propagationsmanager bereichern sollen, aber nicht direkt der Transformation zuzuordnen sind.

Bei einer Transformation wird besonderes Augenmerk auf das Erzeugen einer XML-konformen Struktur gelegt, d.h. die Ergebnisse einer Transformation des Propagationsmanagers müssen gültige XML-Strukturen sein.

Die bisher bestehende Implementierung zur Tranformation basiert auf XSLT (s.Kap.3), da es eine Reihe von Anforderungen erfüllt, die bei der Konzeption des Propagationsmanagers zu berücksichtigen waren. Zum Zeitpunkt der Entwicklung des Propagationsmanagers wurden die Anforderungen durch XSLT ausreichend abgedeckt. Im Einzelnen waren dies (nach [Ker 01]):

- Erzeugen/Aufbrechen von XML-Strukturen
- Typumwandlungen / Bearbeitung von Zeichenketten
- numerische Berechnungen
- Integration verschiedener Datenquellen

Grundlegend bei einer Transformation ist das Zerlegen der Ausgangsstruktur und das Erzeugen der Zielstruktur. Die Erzeugung der Zielstruktur findet dabei gewöhnlich in Abhängigkeit der vorgefundenen Ausgangsstruktur statt. Das Auffinden von gesuchter Information in der Ausgangsstruktur steuert dabei die Erstellung der Zielstrukturen.

Obwohl XML eine strukturgebende Markup-Sprache ist, liegt es in der Hand des Benutzers, wie er seine Daten bei der Modellierung in XML durch strukturierende Elemente darstellt.

Meist wird die Entscheidung der Modellierung wesentlich von den zu modellierenden Daten bestimmt. Als Beispiel sei der Überweisungsvordruck einer Bank genannt. Dieser verfügt über einige Felder, die definierte und atomare Daten wie Nachnahmen, Konto-Nummer und Bankleitzahl enthalten. Diese sind typisierbar in z.B. Integer- und String-Typen und können als solche direkt in eine XML-Struktur übernommen werden.

Andere Felder wie der "Verwendungszweck" können jedoch beliebige Daten enthalten. Sollen auch mit solchen unstrukturierten Datensätzen Transformationen ermöglicht werden, so müssen Operationen wie Teilstringsuche, Ausschneiden von Strings und einige weitere unterstützt werden.

Auf der Seite der strukturierten Daten sollen Transformationen wie verschiedene Typumwandlungen in das erforderliche Format des Zielsystems unterstützt werden. Dabei können auch Berechnungen wie etwa die Umwandlung von Masseinheiten auftreten. Hierzu sollte es möglich sein, über die gewöhnlichen Aggregationsfunktionen hinaus, auch beliebige Funktionen selbst zu definieren.

Falls sich im Rahmen der Weiterentwicklung des Propagationssystems neue Transformationsanforderungen ergeben, die mit den Mitteln der gewählten Anfragesprache nicht zu realisieren sind, so sollte eine Möglichkeit zur Erweiterung gegeben sein.

Im Folgenden sollen nun die notwendigen Transformationsarten und Methoden zusammengestellt und an Beispielen genauer beschrieben werden. Zunächst wird der allgemeine Transformationsablauf betrachtet, gefolgt von Operationen die sich in die Kategorien Struktur- und Datenänderung einteilen lassen. Eine Strukturänderung bezeichnet dabei Anpassungen auf der strukturellen Ebene des XML-Dokuments, Datenänderungen beziehen sich auf die Informationsbereiche des XML-Dokuments, die nicht weiter strukturiert sind (Attributinhalt, Text in Elementen).

Aufbrechen und Erzeugen von XML-Strukturen

Bei einer Transformation werden Informationen von unterschiedlichen Stellen des Quelldokuments kombiniert. Eine XML-Struktur besteht aus einer hierarchischen Anordnung von Elementen. Jedes Element kann als Inhalt Daten oder eine Schachtelung weiterer

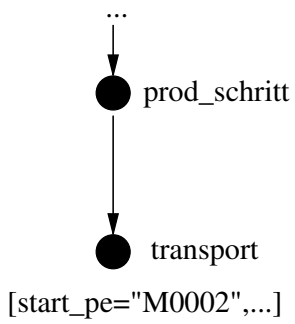
Elemente aufnehmen. Elemente können mit Attributen annotiert werden. Es muss daher bei der Analyse der Eingangsdaten auf alle XML-Strukturelemente wie Elementinhalt und Attribute zugegriffen werden können.

Die gewonnene Information wird schließlich in das Zieldokument integriert. Hierbei muss es wiederum möglich sein, die Struktur des Quelldokuments zu übernehmen, gegebenenfalls zu modifizieren oder vollständig neu aufzubauen. Die Validität der Struktur des Zieldokuments sollte bei der Erzeugung prüfbar sein, extern geschieht dies meist durch ein Schema.

Auswahl von Elementen der XML-Strukur

Um ein XML-Dokument in ein anderes zu transformieren, muss die Möglichkeit bestehen, in der Struktur des Ausgangsdokuments zu navigieren und einzelne XML-Strukturen auszuwählen und zu isolieren. Solche XML-Strukturen können Elemente, Attribute, Daten oder auch ganze Teilbäume des XML-Dokuments sein.

Diese Auswahl kann z.B. durch Angabe eines Pfades durch das Quelldokument erfolgen, wie in Abb. 2.1 dargestellt. Die Unterstützung von absoluten und relativen Pfaden ist wünschenswert.



Pfad: //prod_schritt/transport/[start_pe]

Abb. 2.1 : Pfadangabe mit Bedingung

Die Pfadangabe in Abb.2.1 benutzt einen Wildcard-Operator "//", so dass die Schachtelungtiefe der gesuchten Struktur im Quelldokument beliebig ist. Zusätzlich ist es notwendig, dass man Bedingungen bei der Navigation formulieren kann, um gezielt die Information herauszugreifen, die man benötigt. Dies ist im Beispiel durch die Bedingung [start_pe] angedeutet, die ein transport-Element nur dann berücksichtigt, wenn ein entsprechendes Attribut existiert. Die zugehörige XML-Struktur ist in Abb.2.2 dargestellt.

```
<?xml version="1.0" encoding="ISO8859-2"?>
<auftr_abw zyklus="1">
    <prod_schritt>
        <transport start_pe="M0002" ziel_pe="M0001">
            <teil nr="T00000001" anzahl="5000"/>
        </transport>
    </prod_schritt>

[...]

</auftr_abw>
```

Abb. 2.2: Navigation

Die Navigation bestimmt zugleich die Reihenfolge der Verarbeitung der Elemente des Quelldokuments. Bei der Erstellung einer Zielstruktur sollte daher die Reihenfolge der Information modifiziert werden können, beispielsweise durch eine Sortierung.

2.2 Strukturänderungen

In diesem Abschnitt werden Transformationen beschrieben, die eine Änderung des Strukturbaumes eines XML-Dokuments um Ziel haben. Die einzelnen Bereiche der Strukturänderungen umfassen:

- Filterung
- Anordnung/Sortierung
- Änderung der Modellierung
- Umbenennen
- Integration mehrerer Dokumente

Diese Transformationen können auch in Kombination auftreten und dadurch komplexere Transformationen bilden.

2.2.1 Filterung

Bei einer Transformation muss auf mehreren Stufen eine Möglichkeit zum Filtern der verarbeiteten Daten gegeben werden. Eine erste Filterung wird bei der Auswahl der zu transformierenden Elemente des Quelldokuments angewandt. Diese Filterung erfolgt im Rahmen einer Navigation durch das Quelldokument, welche relevante Information auffinden und extrahieren soll. Dabei gelangen nur diejenigen Strukturen in die weitere Bearbeitung, die gestellte Auswahlkriterien erfüllen.

Eine weitere Filterung wird im sogenannten Transferschritt bei der Erzeugung von Zwischenergebnissen benötigt. Nur generierte Strukturen, die die gestellten Kriterien erfüllen, sollen in die Endbetrachtung eingehen (z.B. Duplikatfilterung).

Schließlich muss eine Möglichkeit zur Filterung beim Aufbau der Ergebnisstruktur vorgesehen werden. Nach der Verarbeitung der Eingabestrukturen und Integration der Teilergebnisse zu einer Rückgabestruktur müssen nachträglich unerwünschte Resultate unterdrückt werden können, da erst nach der Integration entschieden werden kann, ob ein Ergebnis sinnvoll ist.

2.2.2 Anordnung/Sortierung

Die Anordnung bezieht sich auf die Abfolge der Ergebniselemente einer Transformationsanweisung. Attribute besitzen im dem XML-Datenmodell keine Reihenfolge und können auch nicht in der Abfolge beeinflusst werden.

Eine Reihenfolgenänderung von Elemente wird durch eine Sortierung erreicht, wobei verschiedene Kriterien wie Elementnamen, Daten- oder Attributinhalte zur Bestimmung herangezogen werden sollen. Die Sortierung soll alle Merkmale einer XML-Struktur berücksichtigen können, d.h. tiefer eingebettete Information soll auf einer höheren Sortierebene zur Entscheidung herangezogen werden.

Bei der Sortierung sollen alle Ebenen der Baumstruktur bearbeitet werden können, auch getrennt voneinander. Als weiteres Kriterium soll die Form der Elementanordnung des Ausgangsdokuments dienen, die beibehalten oder verändert werden kann. Falls bei der Tranformation neue Elemente generiert werden, so ist zu entscheiden, wie sie in eine solche Sortierung integriert werden können.

2.2.3 Änderung der Modellierung

In XML kann man abzulegende Daten durch Datenfelder, aber auch durch das gewählte Markup ausdrücken. Das Markup dient dabei eher der Strukturierung von Daten, kann aber selbst durch die Benennung Informationstragend sein. Zur Aufnahme von Daten sind Attribut- und Elementwerte verfügbar.

Bei der Modellierung der Daten muss entschieden werden, ob Daten als Inhalt von Attributen oder Elementen kodiert werden. Es gibt verschiedene Argumentationsmöglichkeiten für oder gegen die Verwendung von Attributen zur Ablage von Daten. Oft wird gesagt, dass Daten als Elemente und Meta-Daten als Attributwerte kodiert werden sollten. Schlüssiger erscheint eine Entscheidung aufgrund struktureller Eigenheiten der beiden Datenformen. Es können keine zwei Attribute mit gleichem Namen in einem Element vorkommen. Die Abfolge der Attribute ist implementationsabhängig und nicht durch ein Datenmodell vorgegeben, wogegen Elemente hierarchisch und in einer geordneten Struktur abgelegt werden.

Elemente können in sich geschachtelt und somit strukturiert werden, mit Attributen ist dies nicht möglich. Aus Anwendungssicht spricht bei Verwendung einer DTD die Validierung von Attributen, bei Verwendung von XML-Schema die Validierung von Elementen.

Eine DTD kann Elementinhalte nur als Text auffassen, den Inhalt von Attributen jedoch genauer definieren und einschränken. Dies war auch der Grund zur Entwicklung von XML-Schema, welches eine feine Typprüfung von Elementinhalten ermöglicht.

Da die Festlegung "Element oder Attribut" letztendlich jedoch eine Entscheidung des Modellierers ist, muss die Möglichkeit vorgesehen werden, Daten, die in einer der beiden Formen abgelegt sind, in der jeweils anderen auszudrücken.

Anpassung

Unter einer Anpassung werden kleine Änderungen verstanden, die die Struktur und Anordnung des Quelldokuments weitgehend beibehalten. Die Transformation erfordert hierbei keinen Neuaufbau des Zieldokuments, sondern richtet sich nach der Strukur des Quelldokuments. Eine Anpassung umfasst auf der Datenseite die Änderung von Namen von Attributen (s.Abb. 2.3) und Elementen.

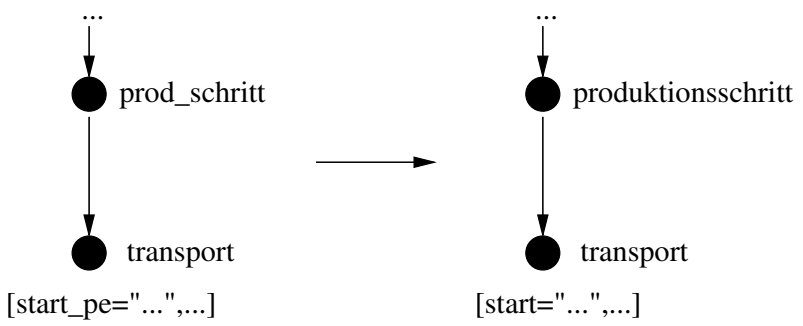


Abb. 2.3 : Namensänderungen

Anpassungen auf der strukturellen Ebene bezeichnen z.B. das Löschen von Elementen oder Attributen, wobei die umliegenden Strukturen nicht verändert werden (s.Abb.2.4). Dies kann auch als feine Filterung betrachtet werden, bei der gezielt Bestandteile der Struktur des Quelldokuments unterdrückt werden können.

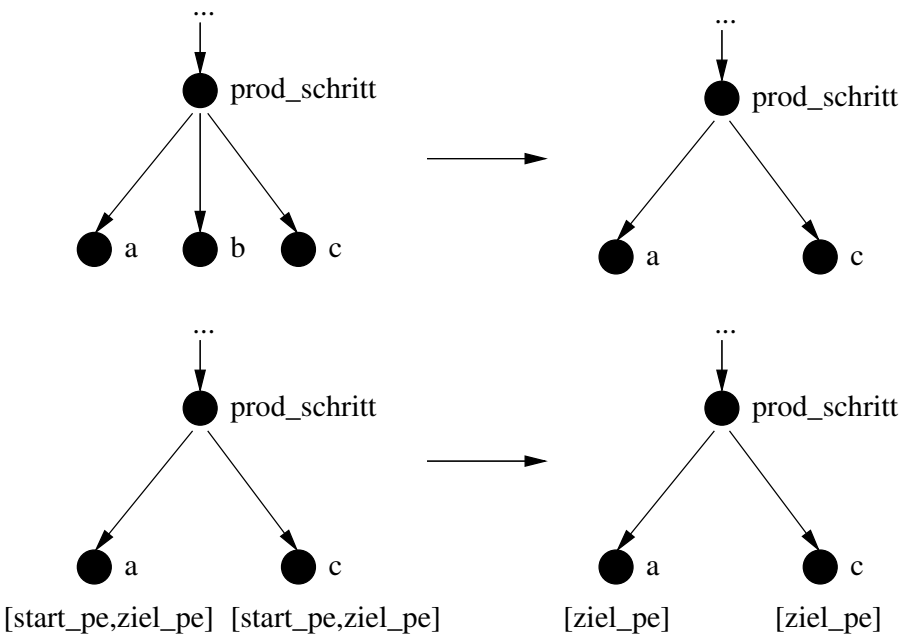


Abb. 2.4: Löschen von Information

Umwandlung

Bei einer Transformation wird oft zwischen zwei unterschiedlichen Modellierungen von Daten vermittelt. Die Transformation geschieht dabei durch gezielte Integration der Information in eine Zielstruktur. Einer strukturellen Transformationsfunktion kann z.B. eine Elementmenge übergegeben werden. Der Ergebnistyp hängt von der Funktion ab und kann eine veränderte Struktur der Elemente darstellen, indem aus einem Element ein Attribut entsteht (s.Abb. 2.5).

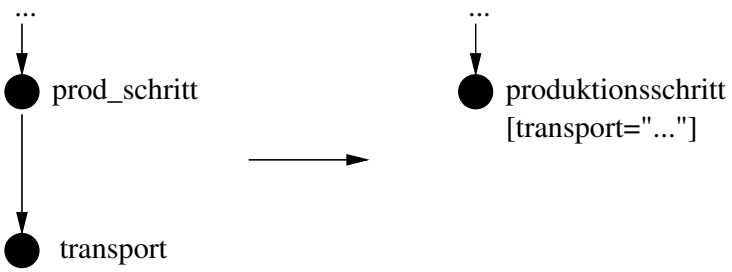


Abb. 2.5: Umwandlung

2.2.4 Ersetzungen/Strukturänderungen

Ausgewählte Strukturen sollten durch neue ersetzt werden können (s.Abb. 2.6). Dabei sollen sowohl einzelne Strukturelemente wie einzelne Elemente, Attribute oder Daten, als

auch komplexe Schachtelungen ersetzt werden können. Dies schliesst auch die Entfernung von Attributen oder Elementen ein.

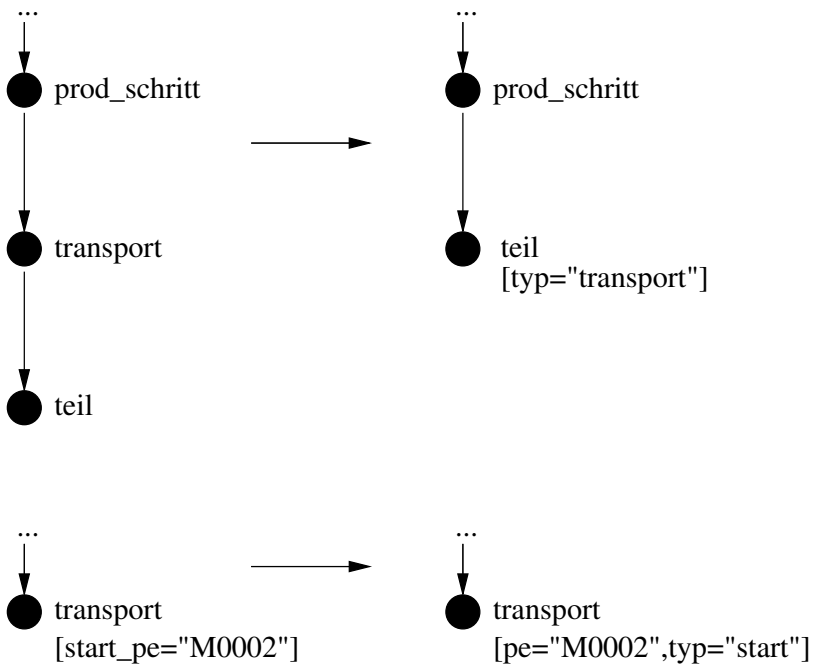


Abb. 2.6: Ersetzung

Manchmal ist es auch nötig, die Struktur des Zieldokuments völlig neu aufzubauen. Es muss eine Methodik bestehen, einen XML-konformen, hierarchischen Baum zu konstruieren. Hierzu muss das Instrumentarium der XML-Modellierung verfügbar sein. Es müssen Elemente erzeugt und belegt werden. Attribute müssen angelegt und mit Werten gefüllt werden. Eventuell müssen Validierungen und Typprüfungen durchgeführt werden.

2.2.5 Integration mehrerer Dokumente

Im Rahmen des Workflow-Modells des SIES können die Eingabedaten aus mehreren Quellen stammen. Es muss die Möglichkeit bestehen, diese Daten in einer integrierten Sicht einem Transformationsprozess zugänglich zu machen. Im Verlauf der Verarbeitung sollen beliebige Verknüpfungen dieser Eingabedaten möglich sein, dies schließt eine Navigation auf mehreren Dokumenten ein.

2.3 Datenänderungen

Als Datenfelder fasst man in XML die Attributwerte und den textuellen Inhalt von Elementen auf. Der Inhalt eines Elements kann aus PCDATA (parsed character data) oder weiteren eingeschachtelten Elementen bestehen (s. Abschnitt Änderung der Modellierung). PCDATA bezeichnet dabei allgemeine Zeichenketten, die keine Elementstrukturen enthalten dürfen.

Da die Datenfelder unstrukturiert und je nach Modell untypisiert sein können, muss man durch eine nachträgliche Atomisierung zwischen verschiedenen Datentypen wie Strings oder Zahlen unterscheiden können. Natürlich können Zahlen auch als Strings aufgefasst werden und einzelne Elemente von Strings können Zahlen sein. XML selbst lässt keine tiefere Typisierung der Datenfelder zu. Es liegt daher in der Interpretation der Daten, welche Transformationen angewandt werden können.

Mögliche Operationen auf unstrukturierten Textdaten umfassen:

- Zusammenführung
- Aufteilung
- Aggregation
- Berechnungen
- Umrechnungen
- Funktionen
- Typkonvertierungen
- Verknüpfung von Daten

Diese Operationen sind nicht orthogonal zu sehen, sondern bauen teilweise auf den anderen Operationen auf.

2.3.1 Zusammenführung/Aufteilung

Eine Zusammenführung kann nur mit zuvor extrahierten Daten erfolgen. Zur Extraktion von Information aus unstrukturierten Abschnitten sind einige Funktionen nötig, die Teilelemente nach verschiedenen Kriterien herausfiltern können (s.Abb. 2.7).

begins-with	Test auf Enthaltensein
ends-with	Test auf Enthaltensein
contains	Test auf Enthaltensein
substring	Extraktion
uppercase/lowercase	Konvertierung groß/klein
cut	Abschneiden
length	Rückgabe der Länge

Abb. 2.7: mögliche Funktionen auf Zeichenketten

Neben dem Verändern von Zeichenketten muss es möglich sein, Tests auf diesen durchzuführen. Dies ist notwendig, um im Falle von unbekannten unstrukturierten Daten die Art und Umfang der zu modifizierenden Zeichenfolge zu bestimmen.

2.3.2 Aggregationsfunktionen

Unter Aggregationsfunktionen wird der Umfang, wie er in Anfragesprachen wie dem SQL-Standard verfügbar ist, benötigt. Als Aggregationsfunktionen werden Funktionen bezeichnet, die aus einer Menge von Eingaben einen neuen Wert berechnen und somit die Wertemenge zu einer neuen Information verdichten. Den Umfang der standardisierten Aggregationsfunktionen ist in Abb. 2.8 gezeigt.

Funktion	Semantik
min	Minimum
max	Maximum
sum	Summe
count	Anzahl
avg	Durchschnitt

Abb. 2.8: Aggregationsfunktionen

2.3.3 Typkonvertierungen

Der textuelle Inhalt von Attributen und Elementen wird in XML grundsätzlich als CDATA aufgefasst. Die Interpretation als z.B. Integer oder Realzahl muss von einem weiterverarbeitenden Programm entschieden werden.

Einschränkungen zum Zeitpunkt der Belegung von Datenfeldern können mit sogenannter statischer Typisierung getroffen werden. Geläufiger ist jedoch die dynamische Typisierung z.B. mit Hilfe von XML Schema, gegen das ein erzeugtes XML-Dokument nachträglich validiert werden kann. Eine Transformation muss in jedem Fall zwischen den gelieferten und benötigten Datentypen vermitteln können. Dies ist in Abb. 2.9 dargestellt, wobei hier eine Konvertierung von Integer nach Real mit unterschiedlicher Semantik dargestellt ist.

```
1.6 -> 1 (Truncate)

1.6 -> 2 (Rundungen)
```

Abb. 2.9: Typkonvertierung Integer nach Real

Für die Typkonvertierung sollten möglichst viele atomare Typen durch das Datenmodell der Anfragesprache definiert sein und konvertiert werden können. Eine Auswahl der von XML-Schema überprüfbaren atomaren Typen ist zum Vergleich in Abb.2.10 dargestellt. Ist eine native Konvertierung nicht möglich, so muss diese mit Hilfe von Funktionen auf Zeichenketten und arithmetischen Operation nachgebildet werden können.

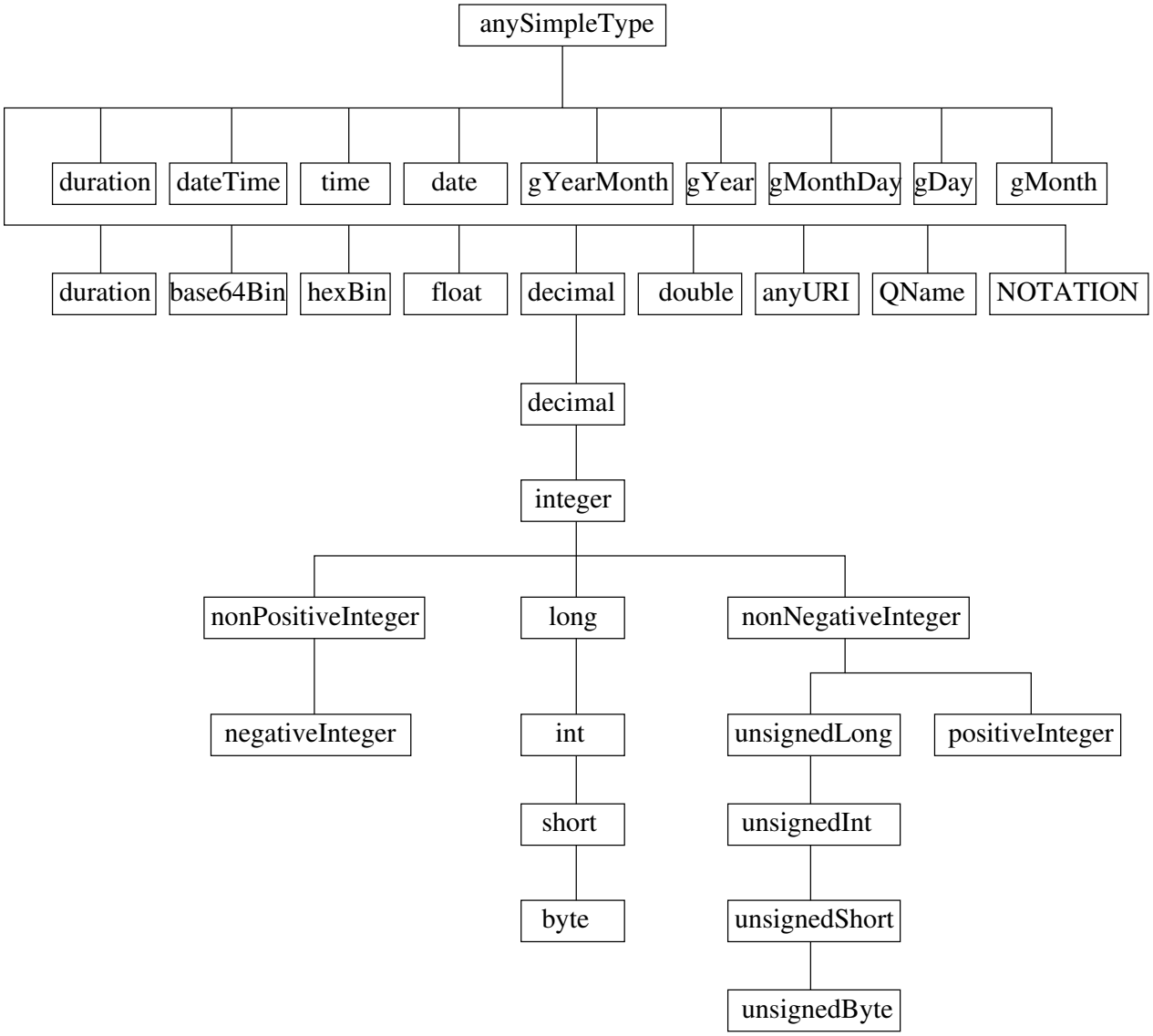


Abb. 2.10 : Datentypen von XML-Schema (aus [W3C 01/2])

2.3.4 Umrechnungen

Bei einer Umrechnung sind zwei Arten zu unterscheiden. Auf der einen Seite gibt es Umrechnungsarten, die durch berechnende Funktionen durchgeführt werden können, wie eine Währungsumrechnung. Bei einer Währungsumrechnung muss in Abhängigkeit der Repräsentation eventuell zunächst der Wert in Zahl und Maßeinheit aufgeteilt werden. Andere Umrechnungen, deren Semantik sich nicht numerisch beschreiben lässt, müssen sich vorgefertigter Tabellen bedienen. Als Beispiel sei hier die Konvertierung von Namen dargestellt (s.Abb. 2.11)

Quelle	Ziel
AMP	AM Prüfstand
U. Heinkel	Uwe Heinkel
S. Opletal	Sascha Opletal

Abb. 2.11: Umrechnungstabelle

2.3.5 Selbstdefinierte Funktionen

Um sich wiederholende, allgemeine Berechnungen durchführen zu können, sollte die Möglichkeit bestehen, selbstdefinierte Funktionen zu verwenden. Die selbstdefinierten Funktionen sollen in der Verwendung nicht von bestehenden Funktionen der Anfragesprache zu unterscheiden sein.

2.3.6 Verknüpfung von Daten

Eine Verknüpfung bezeichnet die Zusammenführung von Daten innerhalb eines oder von mehreren Dokumenten nach speziellen Kriterien. Man unterscheidet dabei Grouping und Join.

Grouping

Eine Gruppierung umfasst die gezielte Zusammenstellung von Datenelementen zweier Dokumente. Dies kann sich auf beliebige Information wie Attribute oder Elemente beziehen. Es wird eine Resstrukturierung der Information erzeugt.

Verbund/Join

Ein Join umfasst die Kombination von Datensätzen aufgrund einer Übereinstimmung zwischen der Information zweier Datensätze zu einem neuen. Danach kann in aus der neu angeordneten Struktur passende Information, die sich aus beiden Dokumenten zusammensetzt extrahiert werden.

2.3.7 Einfachere Verwendung als XSLT

XSLT-Skripte werden durch die nicht-sequentielle Abarbeitung der Eingangsdaten sehr schnell unübersichtlich und schwer zu verstehen. Die Ausführungsreihenfolge hängt von der Adressierung und Struktur der Eingangsdaten ab.

Es ist zu untersuchen, inwieweit XML-Anfragesprachen durch Verwendung einer Algebra zum Verständnis einer Transformationsformulierung beitragen.

2.4 Methoden zur Durchführung von Transformationen

Im Folgenden sollen die sprachliche Semantik zusammengestellt werden, die benötigt wird, um Transformationen beschreiben zu können. Dies umfasst Methoden wie Iterationen, Bedingungen und Variablenbindung.

2.4.1 Lokalisierung von Elementen

Um Elemente in der Ausgangsdatenmenge auszuwählen, benötigt man Pfadausdrücke. Wichtig ist hierbei, dass auch unvollständige und relative Angaben möglich sind. Hierdurch können Lokalisierungsvorgänge automatisiert werden und auch unbekannte Strukturen verarbeitet werden.

In XSLT wird hierzu die XPath-Language benutzt. XPath wurde 1999 vom W3C eingeführt, um innerhalb von XML-Dokumenten navigieren zu können. XPath unterstützt eine absolute, relative und unterspezifizierte Adressierung (s.Abb. 2.12).

```
/a/b/c Absolut
/a/parent/following-sibling Relativ
/a//b Unterspezifiziert
/a/b[Attribut=Wert]/c gesteuert durch Bedingung
```

Abb. 2.12: Lokalisierung von Elementen

Die Pfade können nicht nur strukturbezogen angegeben werden, sondern auch Bedingungen hinsichtlich des Inhalts des XML-Dokumentes enthalten.

2.4.2 Variablenbindung

Variablen dienen zur Berechnung und zum Speichern von Zwischenergebnissen, die nicht sofort in der Zielstruktur auftauchen sollen. Bei der Variablenbindung wird sowohl die Zuweisung von einzelnen Werten, als auch von Gruppen von Werten benötigt.

2.4.3 Bedingungen

Bedingungen dienen der Ablaufsteuerung. Es müssen Boolesche Ausdrücke in Form von Funktionen abprüfbar sein. Auch strukturelle Information soll zur Bestimmung der Alternative beitragen (s.Abb. 2.13).

```
if ($part/@discounted)
  then $part/wholesale
  else $part/retail
```

Abb. 2.13: Bedingungen

2.4.4 Iterationen

Iterationen dienen der seriellen Abarbeitung von Elementen. Dabei wird die Abarbeitung von der Vorkommensreihenfolge der Elemente gesteuert.

2.5 Informationsextraktion mit Anfragesprachen

In [Fer 99] wurden anhand einiger Beispiele herausgearbeitet, welche Typen von Anfragen von einer XML-Anfragesprache unterstützt werden sollten. Die Beispiele umfassen:

- Selektion und Extraktion
- Flattening
- Beibehalten der Struktur
- Änderung der Schachtelung
- Grouping
- Kombinieren von Datenquellen
- Indexieren
- Sortieren
- Tag Variablen
- Reguläre Pfadausdrücke

Obwohl Fragestellung dieser Untersuchung auf Anfragen ausgerichtet ist, können einige der Methoden bei der Transformation Verwendung finden.

2.6 Integrationsanforderungen

Die Integrationsanforderungen beziehen sich auf die Eigenschaften, die eine konkrete Implementation einer Anfragesprache erfüllen sollte.

2.6.1 Plattform

Das vorhandene Propagationssystem wurde in Java entwickelt. Die Verwendung von Java gewährleistet die Unabhängigkeit von einer bestimmten Plattform (Betriebssystem/Prozessor), da Java-Programme durch eine virtuelle Maschine ausgeführt werden.

Zur Integration in dieses Umfeld sollte der Prozessor der Anfragesprache möglichst in Java geschrieben sein, um die Homogenität des Systems und die Plattformunabhängigkeit zu erhalten.

Alternativ besteht die Möglichkeit, einen Prozessor zu integrieren, der für eine Reihe von Plattformen verfügbar ist oder in einer Client-/Server-Architektur angesprochen werden kann.

2.6.2 Zugriff auf Dokumente

Gewöhnlich werden Datenquellen in einer verteilten Umgebung durch URIs angegeben. Im Rahmen der ursprünglichen Realisierung mittels eines XSLT-Prozessors wurde ein URI-Resolver eingesetzt, um so XML-Dokumente einfach adressieren zu können.

Diese Funktionalität soll bei der Integration eines Anfrageprozessors nach Möglichkeit beibehalten werden.

2.6.3 Integration verschiedener Datenquellen

Manche Transformationen stehen in Abhängigkeit zu mehreren Datensätzen. Es muss daher möglich sein, mehrere Datenquellen in einen Transformationslauf einzubeziehen und aus diesen ein Zieldokument zu generieren. Entfernte und lokale Dokumente sollen gleichzeitig Verwendet werden können.

2.6.4 Anfrageschnittstelle

Um die Anfragen selbst automatisiert zu transformieren, ist es von Vorteil wenn die Syntax der Anfragen XML-konform ist. Mit einer XML-konformen Anfragesprache können einfache Anpassungen mit dem Transformationsinstrumentarium, das hier vorgestellt wurde, durchgeführt werden.

Um neue Anfragen automatisch zu generieren, ist zu untersuchen, ob sich Anfragen in einzelne semantische Einheiten zergliedern lassen, aus denen bei Bedarf ein neue automatisiert zusammengestellt werden kann.

2.6.5 Externe Funktionen

Um eine Funktionalität bereitzustellen, die zum Zeitpunkt der Anforderungsanalyse noch nicht berücksichtigt wurde und die sich mit den Mitteln der Anfragesprache nicht beschreiben lässt, sollte die Möglichkeit einer Erweiterung der Anfragesprache gegeben sein.

Dies kann in Form einer Schnittstelle zu einer Programmiersprache realisiert werden. Zum Verständnis und zur Unterstützung der Abschätzung der Portabilität zwischen den Implementierungen der Anfragesprachen, sollten die externen Funktionen von den internen unterschieden werden können.

2.7 Zusammenfassung

Es wurden Anforderungen zur Funktionalität der XML-Anfragesprache und ihrer Implementation aufgestellt. Dabei wurden die Anforderungen der Anfragesprache unterschie-

den durch Transformationen auf der Strukturebene und Transformationen auf unstrukturierten Datenelementen. Die Anforderungen zur Integration ergaben sich aus der Art der bisherigen Implementation.

Diese Anforderungen sind die Bewertungsgrundlage bei der Auswahl einer Anfragesprache, sie sind möglichst vollständig zu erfüllen. Die Anforderungen zur Integration sind nicht zwingend zu erfüllen, da sie notfalls durch geeignete Änderungen der Komponenten des SIES umgangen werden können. Erstrebenswert wäre jedoch die Beibehaltung etablierter Strukturen des SIES.

3 Überblick XML-Verarbeitung

In diesem Kapitel soll die Vielfalt der Sprachen vorgestellt werden, die zur Transformation von XML-Dokumenten eingesetzt werden können. Dabei liegt der Schwerpunkt auf dem Umfang der verfügbaren Implementation. Es wird aber auch schon aufgrund offensichtlicher Eigenheiten der Sprache eine Vorauswahl hinsichtlich Eignung zur Transformation getroffen.

XML wurde als reines Datenrepräsentationsformat entwickelt und sollte eine Vereinfachung von SGML darstellen. Die Interpretation eines XML-Dokumentes erfolgt durch die verarbeitende Applikation, die in der Lage sein muss, die Struktur des XML-Dokuments auszuwerten. Mit der Entwicklung von XML als Alternative zu SGML, sowie der Evolution von HTML zu XHTML kam schnell der Wunsch zur Weiterverarbeitung von in XML kodierten Dokumenten auf. Es gibt zwei historische Linien in denen XML eingesetzt wird, die der Transformations- und die der Anfragesprachen. Die eine Seite kommt aus der Welt des WWW und HTML und entwickelte die Transformationssprachen mit dem Ziel, die Darstellung von Information auf Webseiten zu beeinflussen.

Die andere Seite stammt aus der Welt der Datenbanken und entwickelte aus Erfahrung mit relationalen und objektorientierten Datenbank-Anfragesprachen semantisch ähnliche Konzepte für XML [Kay 03]. Das Ziel hierbei ist es, den Inhalt von Datenbanken durch eine unabhängige Repräsentation austauschbar zu machen und Anfragen in ähnlicher Weise wie in bisherigen Datenbank-Anfragesprachen an XML-Dokumente zu stellen. Man stellte fest, dass XML je nach Art der Anwendung, Verarbeitung und Interpretation als Grundlage für die Bedürfnisse beider Gruppen dienlich sein kann.

3.1 Transformationssprachen

Die Transformationssprachen entstanden vor den XML-Anfragesprachen. Sie wurden jeweils aus speziellen Bedürfnissen heraus entwickelt, um XML besser nutzbar zu machen. Einige dieser Transformationssprachen sind:

- XSL(T)
- XMLTrans
- fxt
- STX
- XDuce/CDuce

Die Sprache XSLT wurde Zeitgleich mit der XML 1.0 Spezifikation vorgestellt und ist auf XML-Transformationen ausgerichtet. Als sich XML noch in der Draft-Phase befand, und es nicht absehbar war, was XSLT leisten würde, wurde die Sprache XMLTrans entworfen, um XML als Datenbasis für Wörterbücher zu verwenden. Zur Beschreibung von Transformationen verwendet XMLTrans reguläre Ausdrücke und ist somit leichter zu erlernen als XSLT. Es stellte sich heraus, dass XMLTrans nur eine Teilmenge der Transformationen des verabschiedeten XSLT beherrscht, was jedoch für den geplanten Einsatzzweck ausreichend war ([Wal 00]).

Da die Verarbeitungsreihenfolge von Elementen bei der Anwendung von XSLT nicht sequentiell erfolgt und auch vom Programmierer nicht immer durchschaubar und beeinflussbar ist, wurde fxt entwickelt, um den Zugriff auf Elemente exakt spezifizieren zu können [Ber 01]. Ein weiteres Ziel von fxt war es, durch einen deklarativen Ansatz eine verständlichere Transformationssprache zu erzeugen, als es im funktionalen Ansatz von XSLT der Fall ist.

STX erweitert XSLT 1.0 um Gruppen (<stx:group>), Puffer zur Zwischenspeicherung (<stx:result buffer>), Methoden zur Textbearbeitung (z.B. <stx:replace>). Viele der Konzepte von STX finden sich in der Spezifikation zum kommenden XSLT 2.0 wieder.

Die Transformationssprachen XDuce und CDuce beschäftigen sich hauptsächlich mit statischer Typprüfung. In Sprachen mit statischer Typprüfung ist es möglich, innerhalb der Transformationsregeln sicherzustellen, dass ein valides Dokument erzeugt wird. Bei dynamischer Typprüfung wird ein produziertes Ergebnisdokument erst nachträglich durch ein Schema validiert.

Für die hier genannten Sprachen gibt es eine Reihe von Implementierungen. Es soll hier nur der XSLT-Prozessor XALAN genannt werden, der im gegenwärtigen SIES eingesetzt wird (s.Abb. 3.1).

Name	Sprache	Quelle
XALAN	Java	http://xml.apache.org/xalan-j/

Abb. 3.1: XALAN XSLT-Prozessor

3.1.1 XSL(T)

Bei der Spezifikation von XML wurde nur an den Transformationaspekt bei der Konvertierung und Darstellung von Webseiten gedacht. Zur Aufbereitung der in einem XML-Dokument abgelegten Daten wurde die eXtensible Stylesheet Language (XSL) entwickelt, die aus drei unabhängigen Teilen besteht:

Teilsprache	Aufgabe
XSL-FO	Bereitstellung von Formatierungsinformation
XPath	Lokalisierung von Daten
XSLT	Formulierung von Transformationen

Dabei dient XSL-FO zur Angabe von Formatierungsinformation bei der Darstellung von XML-Dokumenten. Dies umfasst Angaben wie Schriftart, Farbe oder Seitenlayout. XPath ist eine Sprache zur Lokalisierung von Daten innerhalb von XML-Dokumenten. Das XPath-Konzept ist durch die Verwendung in vielen XML-Anwendungen sehr weit verbreitet.

XSLT und ähnliche Transformationssprachen wurden speziell zur Restrukturierung von XML-Dokumenten entwickelt. Meist wird XSLT zur Transformation eines XML-Dokuments in andere semi-strukturierte Sprachen wie HTML oder LaTeX benutzt. Es ist jedoch auch möglich, das Ergebnis der Transformation frei zu definieren.

XSLT arbeitet ereignisgesteuert. Das bedeutet, dass beim Durchwandern des Ausgangsdokumentes durch das Auffinden von Schlüsselelementen vordefinierte Transformationen ausgelöst werden. Es ist möglich Transformationsregeln direkt aufzurufen oder indirekt über ein sogenanntes Match-Pattern. Der Durchgang durch das Ausgangsdokument ist hierbei nicht immer sequentiell, da die Ausführungsreihenfolge der Transformation von der Reihenfolge der Daten des Zieldokuments entsprechen muss.

Dies beeinflusst auch die Abarbeitung von Iterationen, so dass deterministische Abfolgen von Element-Iterationen nicht vorausgesetzt werden können. Mit einigem Aufwand durch das Setzen von Prioritäten und Beachten aller möglichen Vorkommnisse können XSLT-Transformationen jedoch so konstruiert werden, dass nur gewünschte Abfolgen bearbeitet werden.

```
<xslt:template match="Pattern" >
 <xsl:for-each select="Pattern">
  <xsl:value-of select=node>
 </xsl:for-each>
</xsl:template>
```

Abb. 3.2: Template Ausdruck

In Abb.3.2 ist ein XSLT-Template Ausdruck dargestellt, durch den eine Transformationsregel definiert wird. Die Regel kann angewendet werden, wenn der aktuelle Kontext die Forderungen des Pattern-Ausdrucks erfüllt. Der Pattern-Ausdruck wird in XPath angegeben. Die Anwendung von Templates wird im aktuellen Kontext explizit angestossen, durch direkten (<xsl:call-template name=...>) oder indirekten Aufruf (<xsl:apply-templates>). Stehen mehrere Templates zur Auswahl, so wird automatisch das erste genommen. Falls Templates mit Prioritäten versehen sind, wird dasjenige mit der höchsten Priorität ausgewählt [Kay 00].

3.1.2 XPath

Zur Navigation in einem XML-Dokument wird die eigenständige Sprache XPath verwendet. XPath bietet:

- Auswahl von Teilstrukturen
- Auswahl von Knotenmengen

- Formulierung von Auswahlbedingungen

Die Auswahl von Knotenmengen und Teilstrukturen erfolgt über sogenannte "Location Paths" [W3C 99/2]. XPath definiert absolute (ab Wurzelelement) und relative (zu anderen Elementen) Location Paths, die sich an den Navigationachsen eines Elements orientieren. Eine Auswahl von möglichen Navigationsachsen eines XPath-Ausdrucks ist in Abb. 3.3 dargestellt.

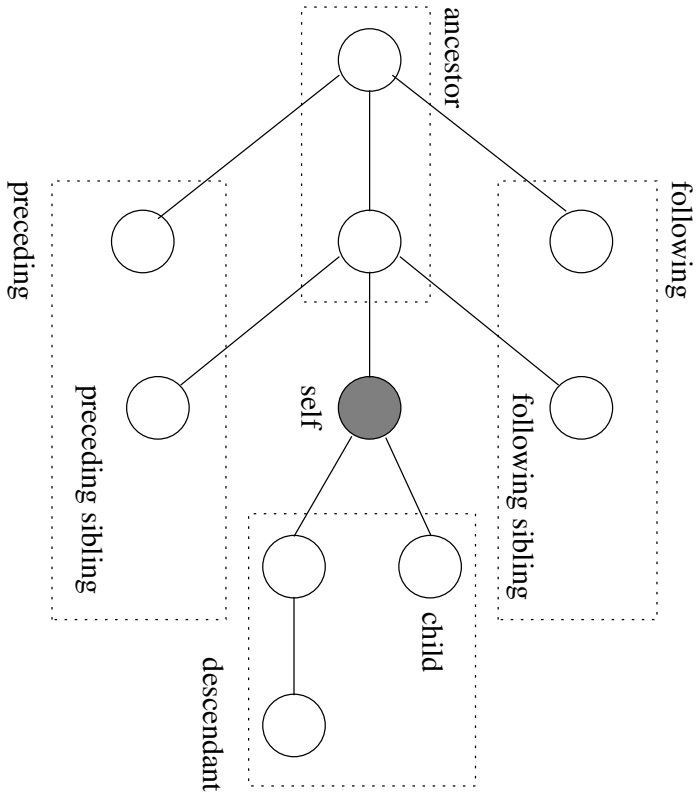


Abb. 3.3 : Navigation mit XPath

Neben der Navigation bietet XPath eine Reihe von Möglichkeiten, um Knotentests durchzuführen, mit denen die Auswahl einer Knotenmenge fein gesteuert werden kann. Hierzu sind eine Reihe von Funktionen zur String- und Zahlverarbeitung verfügbar.

3.1.3 Transformationsbeispiel in XSLT: Materialstrom

Eine Transformation des SIES soll die Anwendung von XSLT demonstrieren. Dabei wird einem IS ein Fertigungsauftrag entnommen, der von einem Zielsystem ausgeführt werden soll. Die Ausgangsdaten (s.Abb.3.4) beschreiben im Rahmen des Fertigungsauftrages die einzelnen Arbeitschritte (<Arbeitsgang>), die eine Menge von Material (<Teil>) zu durchlaufen hat.

```
<?xml version="1.0" encoding="UTF-8"?>
<Fertigungsauftrag Prioritaet="1" ID="1" Status="beendet">
    <Soll_start>2002-10-01T09:30:00</Soll_start>
    <Soll_ende>2002-10-01T16:00:00</Soll_ende>
    <Ist_start>2002-10-01T11:30:00</Ist_start>
    <Ist_ende>2002-10-01T09:30:00</Ist_ende>
    <Teil Teilenr="T001" Menge="5000"/>
    <Fertigungsschritte>
        <Arbeitsgang Nr="1" Operation="Bohren" Maschine="M007"
         Ruestzeit="00:02:00" Prozesszeit="00:01:00"/>
        <Arbeitsgang Nr="2" Operation="Fraesen" Maschine="M009"
         Ruestzeit="00:05:00" Prozesszeit="00:11:00"/>
        <Arbeitsgang Nr="3" Operation="Fraesen" Maschine="M010"
         Ruestzeit="00:05:00" Prozesszeit="00:13:00"/>
    </Fertigungsschritte>
</Fertigungsauftrag>
```

Abb. 3.4 : Ausgangsdaten

```
<?xml version="1.0" encoding="UTF-8"?>
<Materialstrommatrix>
<Transport Ziel="M007" Start="Lager" Menge="5000" Teilenr="T001" Nr="0"/>
<Transport Ziel="M009" Start="M007" Menge="5000" Teilenr="T001" Nr="1"/>
<Transport Ziel="M010" Start="M009" Menge="5000" Teilenr="T001" Nr="2"/>
<Transport Ziel="Fertigteillager" Start="M010" Menge="5000" Teilenr="T001"
 Nr="3"/>
</Materialstrommatrix>
```

Abb. 3.5 : Ergebnis

Die Ergebnisdaten (s.Abb.3.5) beschreiben die Transportwege (<transport>) zwischen den verschiedenen Maschinen (Attribut: Maschine). Ziel der Transformation ist es, die Ausgangsdaten geeignet aufzubereiten, so dass ein zusammenhängender Materialstrom angegeben werden kann (<materialstrom>)(s.Abb. 3.6).

Als Besonderheit ist hier zu erkennen, dass die Ausgangsdaten nicht die alleinige Quelle für Information ist. Das Wissen, dass ein Materialstrom am Lager beginnt und am Ende der Produktion in das Fertigteillager führt, ist nur im Transformationsskript enthalten.

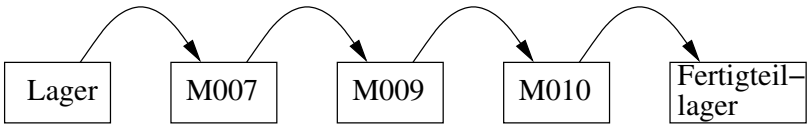


Abb. 3.6 : Materialstrom

Eine Transformation kann somit nicht nur informationsreduzierend oder -erhaltend sondern auch im Sinne einer Informationsanreicherung wirken. Die Möglichkeit der In-

formationsanreicherung ist dabei umfangreicher als durch Aggregationsfunktionen in den klassischen Datenbankanfragesprachen.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output encoding="UTF-8" indent="yes" method="xml"/>
    <xsl:variable name="TeileNr" select="/Fertigungsauftrag/Teil/@Teilenr"/>
    <xsl:variable name="Menge" select="/Fertigungsauftrag/Teil/@Menge"/>
    <xsl:template match="Fertigungsauftrag">
        <Materialstrommatrix>
            <xsl:apply-templates select="Fertigungsschritte"/>
        </Materialstrommatrix>
    </xsl:template>
    <xsl:template match="Fertigungsschritte">
        <Transport Nr="0" Teilenr="{$TeileNr}" Menge="{$Menge}" Start="Lager"
          Ziel="{Arbeitsgang/@Maschine}"/>
        <xsl:for-each select="Arbeitsgang">
            <xsl:variable name="Ziel">
                <xsl:choose>
                    <xsl:when test="position()=last()">Fertigteillager
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:value-of select="following::node()/@Maschine"/>
                    </xsl:otherwise>
                </xsl:choose>
                </xsl:variable>
                <Transport Nr="{@Nr}" Teilenr="{$TeileNr}" Menge="{$Menge}"
                  Start="{@Maschine}" Ziel="{$Ziel}"/>
        </xsl:for-each>
    </xsl:template>
</xsl:transform>
```

Abb. 3.7 : Transformationsskript in XSLT

Das Transformationsskript (s.Abb. 3.7) benutzt zur Transformation eine Iteration über die Arbeitsgänge. In Abhängigkeit der vorgefundenen Information die sowohl durch XSLT als auch durch XPath Bedingungen überprüft wird, wird das Ergebnis generiert.

Durch die gegenwärtige Bearbeitungsposition wird zudem ausgewählt, ob der Inhalt der Attribute “Start” und “Ziel” durch Information aus dem Quelldokument oder dem Transformationsskript (“Lager” und “Fertigteillager”) bestimmt wird.

3.2 Anfragesprachen

XSLT und die XML-Anfragesprachen sind aus verschiedenen Bedürfnissen heraus entwickelt worden. XSLT entstammt aus dem Hintergrund der Markupsprachen, die es zum Ziel haben, lesbare Textinformation aufzubereiten und in anderen lesbaren Text zu transformieren. Die XML-Anfragesprachen werden von Menschen entwickelt, die XML als Austauschformat für Datenbanken ansehen und eine dementsprechend andere Sichtweise auf die Funktionalität einer solchen Sprache haben.

Die Anfragesprachen sind im Gegensatz zu Transformationssprachen darauf ausgerichtet, die gesuchte Information mit möglichst einfachen Mitteln zu beschreiben. Eine Auswahl der bekannten Sprachen und ihre Herkunft ist in Abb.3.8 dargestellt.

Sprache	Herkunft
XML-QL	AT&T/INRIA u.a.
YATL	INRIA Frankreich
Lorel	Stanford University
XQL	GMD/Fuji u.a.
Quilt	IBM/Software AG/INRIA
XQuery	W3C
XML-GL	Politechnikum Mailand
XMAS	Univ. of California

Abb. 3.8 : XML-Anfragesprachen

XML entstand zwischen 1996 und 1998 zusammen mit XSLT, das zur Transformation und Formatierung von XML-Dokumenten gedacht war, um diese in einem Browser darstellen zu können. Mit der Verwendung von XML im Bereich der Datenbanken wurden ab 1997 die ersten Anfragesprachen entwickelt, um eine Modellsicht auf XML-Dokumente bereitzustellen, die sich an den herkömmlichen Datenbank-Anfragesprachen orientieren. Dabei wurden viele Konzepte vorgeschlagen, die aus verschiedenen Informationsbedürfnissen resultieren und die Entwicklung der Anfragesprachen untereinander beeinflusst haben.

Zunächst wurden einige Sprachen unabhängig voneinander entworfen, teilweise unter Berücksichtigung bereits etablierter Sprachen aus der Datenbankwelt. Dort werden Sprachen eingesetzt, die auf das jeweils zugrundeliegende Datenmodell zugeschnitten sind, wie SQL für das relationale und OQL für das objektorientierte Datenmodell. Die dabei entwickelten Sprachen bieten sehr unterschiedliche Möglichkeiten zur Abfrage von Daten. Später wurde auch wie in SQL92 die explizite Aufbereitung und Konstruktion von Ergebnissen berücksichtigt.

Die Sprache Quilt greift gezielt die Stärken einzelner Sprachen wie XPath, XQL, Lorel und Yatl auf und vereinigt sie zu einer neuen Sprache [Cha 00]. XQuery ist ein Standardisierungsversuch des W3C der Quilt als Grundlage nimmt und um Features wie Typerzwingung erweitert. Die Anforderungen an eine standardisierte Anfragesprache, wie vom W3C zusammengestellt, werden in XQuery realisiert.

In Abb. 3.9 ist der zeitliche Verlauf der Entwicklung und die Einflüsse der Sprachen

untereinander dargestellt.

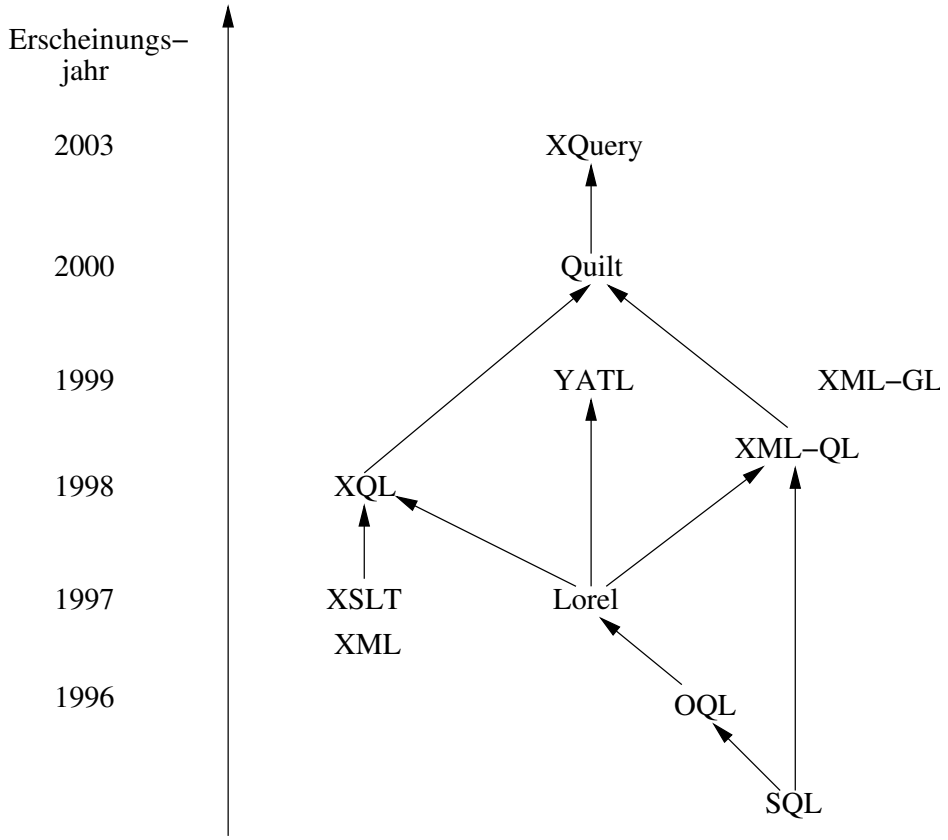


Abb. 3.9 : Entwicklung der Anfragesprachen

3.2.1 Datenmodelle

Eine Anfragesprache muss immer im Kontext des zugrundeliegenden Datenmodells gesehen werden. Normalerweise wird zunächst das Modell erstellt und danach durch die Generierung von Daten Instanzen des Modells gebildet.

Da XML vor den Anfragesprachen definiert wurde, beschreiben die Datenmodelle der Anfragesprachen im Nachhinein eine Interpretation der XML-Dokumente. Viele Sprachen bringen daher ihr eigenes Modell mit (s.Abb. 3.10).

Es werden im Folgenden anhand des OEM- und des XPath-Datenmodells die Auswirkungen auf die durchzuführenden Transformationen untersucht.

Sprache	XML-QL	YATL	Lorel	Quilt/XQuery	XML-GL
Modell	Graph	Ord. Tree	OEM	XPath/XQuery	XML-GDM

Abb. 3.10 : Datenmodelle der Anfragesprachen

Das Object Exchange Model (OEM) ist ein gerichteter Graph, der Objekte, die durch einen "object identifier" eindeutig gekennzeichnet sind, verbindet. Die Kanten verfügen

über Namen. Blattknoten symbolisieren atomare, innere Knoten komplexe Werte. Der Graph muss zusammenhängend sein, so dass alle Objekte über eine Folge von Kantennamen adressiert werden können. Das OEM eignet sich zur Modellierung von relationalen und hierarchischen Daten. Dadurch, dass die Modellierung kein festes Schema vorgibt, können auch semistructurierte Daten modelliert werden.

Das OEM, wie es in Lorel verwendet wird (s.Abb. 3.11) , wurde an XML angepasst, indem zyklische Verweise entfernt und die Annotation von Attributen ermöglich wurde. Es unterstützt eine Reihe atomarer Typen wie Integer, Real, String, GIF, HTML, AUDIO und Java [Abi 97].

Durch die Unterscheidung zwischen komplexen und atomaren Werten kann auf Grundlage des Datenmodells eine Typumwandlung für definierte Datentypen in der Sprache vorgesehen werden.

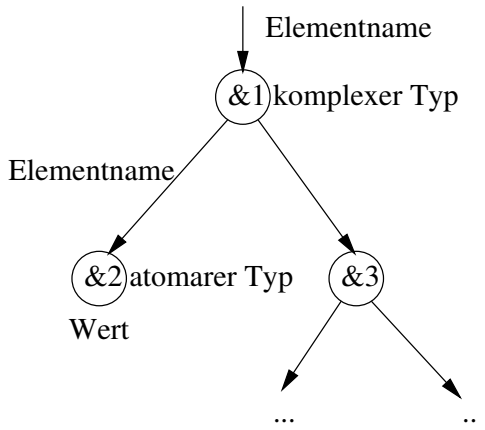


Abb. 3.11 : Object Exchange Model

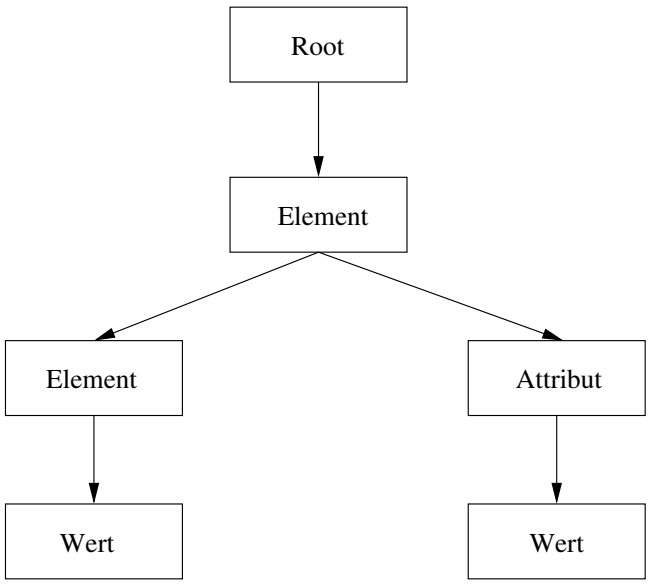


Abb. 3.12 : XQuery/XPath-Datenmodell

Das XQuery/XPath-Datenmodell ist ein Baum, dessen Knoten Bezeichner tragen (s.Abb. 3.12). Im Gegensatz zum OEM-Modell vergibt das XQuery/XPath-Modell auch für die inneren Knoten des Baumes verschiedene Typen. Die Knoten können vom Typ Root, Element, Text, Attribut, Namespace, Verarbeitungsanweisung und Kommentar sein. Dabei darf in einem konkreten Modell nur ein Root-Knoten vorkommen, der auf das XML-Dokument selbst zeigt ([W3C 03/4]).

Zusätzlich unterscheidet sich das XQuery/XPath-Datenmodell vom OEM-Modell dadurch, dass die Knoten des Baumes geordnet sind. Die Ordnung der Knoten entspricht der Anordnung im zugehörigen XML-Dokument und kann durch eine Sprache, die dieses Modell verwendet abgefragt und modifiziert werden. Da einige Transformationen des SIES von der Reihenfolge der Information in einem Dokument abhängig sind, ist dies ein großer Vorteil.

3.2.2 XML Query Algebra

Um auf einem Datenmodell Funktionen und Berechnungen durchzuführen, benötigt man eine Algebra. Jede Anfragesprache arbeitet mittels einer eigenen Algebra auf ihrem Datenmodell. Die Algebra definiert die formal-semantischen Grundlagen für die sprachlichen Konstrukte. Stellvertretend wird hier die XML Query Algebra des W3C vorgestellt, wie sie bei der Definition von sprachlichen Konstrukten in Quilt und XQuery verwendet wird ([W3C 01]). Es wird als Beispiel die Iteration mittels einer for-Schleife gezeigt:

```
  for b in bib0/book do
      book [ b/author, b/title ]

==> book [
      author [ "Abiteboul" ],
      author [ "Buneman" ],
      author [ "Suciu" ],
      title  [ "Data on the Web" ]
    ],
    book [
      author [ "Fernandez" ],
      author [ "Suciu" ],
      title  [ "XML Query" ]
    ]
```

Abb. 3.13 : Ablauf einer FOR-Schleife

Durch die vorgefundene Struktur wird nun in der for-Schleife iteriert und eine Typisierung vorgenommen (s.Abb. 3.13)

```
:   book [
      author[ String ]{1, *},
      title[ String ]
    ]{0, *}
```

Abb. 3.14 : Typisierung des Ergebnisses

Nach der Typisierung kann schließlich das Ergebnis aufgebaut werden. Durch eine Reihe von Gesetzen, die semantische Äquivalenzen beschreiben, können Ausdrücke umgewandelt werden (s.Abb. 3.15).

```
for v2 in (for v1 in e1 do e2) do e3
= for v1 in e1 do (for v2 in e2 do e3)
```

Abb. 3.15 : Assoziativ-Gesetz bei einer FOR-Schleife

Durch die Verwendung einer Algebra können, unter Beibehaltung der Semantik, Optimierungen der Anfrage durchgeführt werden.

3.2.3 Ursprung: SQL/OQL

Ein gängiges Konzept zur Formulierung von Anfragen ist die Anlehnung an Datenbankanfragesprachen wie SQL/OQL. SQL dient zur Abfrage von Daten die auf Grundlage des relationalen Datenmodells gespeichert sind. Die relationale Algebra definiert ein strukturiertes Datenmodell durch sogenannte Tupel. Nicht vorhandene Information wird hierbei explizit durch einen NULL-Ausdruck dargestellt. Dies ist ein wesentlicher Unterschied zu semi-strukturierten Modellen, wie sie durch XML ausgedrückt werden können.

Da den XML-Anfragesprachen nicht die relationale Algebra sondern z.B. die XML Query Algebra [W3C 01] zugrundeliegt, soll hier nur grob die Semantik der Sprachen wiedergegeben werden, um die Wurzeln der XML-Anfragesprachen zu beleuchten.

Der Rückgabewert einer SQL-Anfrage wie in Abb. 3.16 gezeigt, ist in jedem Fall ein Tupel der aus einer relational-algebraischen Transformation der Ausgangsdaten entstanden ist.

```
SELECT schluessel
FROM tabelle
WHERE ...
GROUP BY ...
```

Abb. 3.16 : SQL-Anweisung

Bei einer SQL-Anfrage wird auf den Daten einer Quelle **FROM/SELECT** eine Filteroperation angewandt **WHERE** und eine Anweisung zur Strukturierung des Ergebnisses gegeben **GROUP BY**.

Bei den XML-Anfragesprachen findet man jedoch noch eine Reihe weiterer Konzepte

zur Beschreibung von Anfragen. Dies resultiert aus den Unterschieden der Modellierung der Daten und beinhaltet meist prozedurale Komponenten. Auch ist offen, wie die Rückgabe einer XML-Anfrage zu gestalten ist, da die Anfrage selber meist keiner Algebra unterliegt. Durch den **CREATE VIEW**-Mechanismus von SQL92, der virtuelle Tabellen erstellt, ist es möglich, auch in SQL auf die Struktur des Ergebnisses Einfluss zu nehmen, indem das Endergebnis zwischengespeichert und weiterverarbeitet werden kann, bevor es zurückgegeben wird.

Einige Sprachen erlauben es, dass der Rückgabetyp beliebig und die Anfrage nicht in XML abgeschlossen ist, dies ist in SQL nicht möglich. Eine Ausnahme bilden einfache Sprachen wie XQL, die stets auf den Eingabedaten operieren und keine neuen Strukturen erzeugen können.

	relational	XML
Struktur	geordnet	heterogen
Schachtelung	flach	verschachtelt
Ordnung	ungeordnet	geordnet
Fehlende Information	explizit "null"	abwesend
Belegung	dicht	meist dünn

Abb. 3.17: Relationale- und XML-Daten

Ein Vergleich der typischen Eigenschaften von relationalen mit XML-Datensätzen ist in Abb.3.17 gezeigt. Diese Eigenschaften haben direkten Einfluss auf die XML-Anfragesprachen und deren Algebra.

3.2.4 XML-QL

XML-QL wurde im August 1998 vorgestellt. Das Ziel bei der Entwicklung der Sprache war es, neben der Extraktion von Daten, den Datenaustausch von Unternehmen mittels XML zu unterstützen. Hierzu wurde die Möglichkeit eingeführt, ein XML-konformes Ausgabedokument zu konstruieren. XML-QL ist "relational" abgeschlossen, was heißt, dass u.a. Joins unterstützt werden.

```
WHERE <bib><book>
        <publisher> <name>"Addison-Wesley" </> </>
        <title>  $t </>
        <author> $a </>
      </></> IN "bib.xml"
CONSTRUCT <result>
             <author> $a </>
             <title>  $t </>
          </>
```

Abb. 3.18: XML-QL

Eine XML-QL Anfrage (s.Abb. 3.18) besteht aus einem Lokalisierungsteil **WHERE**,

der das Ausgangsdokument nach einem bestimmten Muster filtert. Gleichzeitig werden die auszuwertenden Datenvorkommen in der Musterspezifizierung an Variablen gebunden, um sie später zugreifbar zu machen.

Die Konstruktion des Zieldokuments erfolgt in einem **CONSTRUCT**-Abschnitt, der die Struktur der zu erzeugenden Elemente beschreibt und die Werte der zuvor gebundenen Variablen einsetzt. Die Abarbeitung des Quelldokuments erfolgt hierbei sequentiell. Eine Besonderheit von XML-QL ist die Verwendung eines unbenannten Abschlußtags, welche das letzte Element schliessen. Dadurch entsteht keine semantische Mehrdeutigkeit, im Bereich der Schachtelung. Das Lesen von komplexen Anfragen wird jedoch teilweise erschwert.

Name	Sprache	Quelle
XMLQL	Java	http://www.research.att.com/sw/tools/xmlql/
XML::QL	Perl	http://cpan.develooper.com/

Abb. 3.19: Implementierungen von XML-QL

Es existiert eine Implementierung in Java (s.Abb. 3.19), was den Einsatz im SIES ermöglichen würde.

3.2.5 YATL

YATL wurde im Jahr 2000 entworfen, um als eine der ersten Sprachen die Anfrage, Konvertierung und Integration von XML-Daten zu vereinen. Die Integration von XML-Daten bezeichnet die Generierung eines XML-Dokuments aus mehreren einzelnen Dokumenten.

```
make
      results [ *result [ title [ $t ], $as ] ]

match "www.bn.com/bib.xml" with

      bib [ *book [ title [ $t ], *($as) author ]

where $t = "..."
```

Abb. 3.20: YATL

Name	Sprache	Quelle
YATL	OCaml	http://www-rocq.inria.fr/ cluet/

Abb. 3.21: Implementierungen von YATL

In YATL (s.Abb. 3.20) werden im Gegensatz zu XML-QL die Analyse und Filterung in voneinander getrennten Abschnitten definiert. Muster werden durch den **match**-Abschnitt, Filter durch den **where**-Abschnitt und die Konstruktion durch den **make**-

Abschnitt beschrieben. Die Verknüpfung der einzelnen Abschnitte erfolgt durch gebundene Variablen, die die Anzahl der zu konstruierenden Strukturen steuern.

Von YATL ist nur ein Prototyp in der Sprache OCaml verfügbar (s.Abb. 3.21), was die Integration in das SIES erschwert.

3.2.6 Lorel

Lorel (s.Abb. 3.22) war eine der ersten Sprachen, um semi-strukturierte Daten abzufragen [Abi 97]. Mit dem Aufkommen von XML wurde Lorel erweitert, um auch XML-Dokumente verarbeiten zu können. Diese Erweiterung umfasst die Konstruktion von XML-Elementen, was im ursprünglichen OEM Datenmodell nicht vorgesehen ist. Wesentliche Merkmale von Lorel sind die automatische Typumwandlung und flexible Pfadausdrücke.

```
select s from review.album a, a.song s
where a.musician = "Clapton"
order by document-order
```

Abb. 3.22: Lorel

Name	Plattform	Quelle
Lore	SunOS/Linux/OSF	http://www-db.stanford.edu/lore/

Abb. 3.23: Implementierungen von Lorel

Von Lorel existiert ein Prototyp (s.Abb. 3.23), der nicht mehr gepflegt wird. Er steht nur auf ausgewählten Unix-Plattformen zur Verfügung. Da kein Quellcode verfügbar ist, würde eine Client-Server Kommunikation mit einem Unix-System benötigt, um Lorel im Rahmen des SIES einzusetzen.

3.2.7 XQL

XQL wurde 1999 vorgestellt. XQL war als Ergänzung zu XSLT gedacht, indem es kurze, einfach zu verstehende Anfragen bieten soll, die auch innerhalb einer URL übergeben werden können. Die Sprache ist dementsprechend einfach aufgebaut und dient hauptsächlich der Abfrage von Information, wobei sie sich sehr stark an XPath orientiert.

```
document("http://www.bn.com")/bib {
  book[publisher/name="Addison-Wesley" and @year>1991]{
  @year | title
}
```

Abb. 3.24: XQL Anfrage

XQL (s.Abb.3.24) bietet nur eine Filterung und Ausgabe der ausgewählten Information. Es gibt keine Methode zum freien Aufbau einer Zielstruktur. Der Aufbau des

Ergebnisses ergibt sich direkt aus der Musterbeschreibung. Dies bedeutet auch, dass keine Strukturänderung z.B. in der Schachtelung von Elementen möglich sind.

```
document("http://...")/a->b {}
```

Abb. 3.25: XQL Renaming-Operator

Bestehende Elemente können durch einen Renaming-Operator (s.Abb. 3.25) umbenannt werden. Es können jedoch keine neuen Elemente erzeugt werden.

Eine Variablenbindung ist in XQL möglich. Die Bildung beliebiger Kreuzprodukte, deren Daten nicht innerhalb des selben Anfragepfades liegen, ist nicht möglich. In einer späteren Version von XQL wurde es ermöglicht, Joins von mehreren Dokumenten zu erzeugen. Allerdings war die Integration dieser Methodik nicht ausgereift und wurde auch nicht konsequent umgesetzt ([Rob 99]).

Name	Sprache/Plattform	Quelle
XQEnginge	Java	http://www.fatdog.com/
GMD-IPSI XQL	Java	http://xml.darmstadt.gmd.de/xql/
XML::XQL	Perl	http://www.erols.com/enno/xql/
Xtract	Haskell/Unix	http://www.cs.york.ac.uk/fp/Xtract/

Abb. 3.26: Implementierungen von XQL

Von XQL gibt es eine Reihe von Implementierungen die geeignet wären, um die Sprache in das SIES zu integrieren (s.Abb. 3.26).

3.2.8 Quilt

Quilt wurde im März 2000 vorgestellt. Das Ziel beim Entwurf war es, die Eigenschaften mehrerer, bisher veröffentlichter XML-Anfragesprachen in einem neuen Ansatz zu vereinigen. Der Entwurf von Quilt wurde beeinflusst durch XML-QL und XQL [Cha 00]. Zur Navigation im XML-Dokument wird XPath benutzt.

```
<publisher_list>
   FOR $p IN distinct(document("bib.xml")//publisher)
   RETURN
     <publisher>
        <name> $p/text() </name> ,
        FOR $b IN document("bib.xml")//book
                 [publisher = $p]
        RETURN <book> $b/title ,
                      $b/price
               </book> SORTBY(price DESCENDING)
     </publisher> SORTBY(name)
</publisher_list>
```

Abb. 3.27: Quilt

Anfragen werden in Quilt durch den FLWR-Ausdruck (FOR-LET-WHERE-RETURN) beschrieben (s.Abb. 3.27), wobei die Variablenbindung im **FOR-LET**-Bereich vorgenommen und das Ergebnis im **RETURN**-Bereich konstruiert wird. Filterung und Sortierungen sind möglich, aber optional, wie in den anderen Sprachen auch.

Name	Sprache	Quelle
Kweelt	Java	http://cheops.cis.upenn.edu/Kweelt/
Quilt Parser	JavaCC	http://www.cobase.cs.ucla.edu/projects/xpress/quilt/

Abb. 3.28: Implementierungen von Quilt

Aufgrund der Verfügbarkeit einer Java-Implementierung ist es möglich, Quilt in das SIES zu integrieren.

3.2.9 XQuery

XQuery ist der Standardisierungvorschlag des W3C für eine XML-Anfragesprache. Die Version 1.0 steht zur Zeit (Anfang 2004) kurz vor der Verabschiedung [W3C 03]. Um die Konzepte dieser Anfragesprache zu testen wurden eine Vielzahl von prototypischen Implementationen entwickelt, die mit dem Fortschreiten der Spezifikation laufend angepaßt werden.

XQuery ist eine Erweiterung der Sprache Quilt und ist daher syntaktisch sehr ähnlich.

```
<bib>
 {
  for $b in doc("data.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  order by $b/@year
  return
    <book year="{ $b/@year }">
     { $b/title }
    </book>
 }
</bib>
```

Abb. 3.29: XQuery

Bei XQuery (s.Abb. 3.29) wurde die Syntax des FLWR-Ausdrucks von Quilt leicht verändert. Man spricht nun von einem FLWOR-Ausdruck, da die Sortierungsanweisung **order by** parallel zu den anderen Ausdrücken steht und nicht wie in Quilt an den **return**-Ausdruck angehängt wird. XQuery ist eine deklarative Sprache, die prozedurale Elemente beinhaltet. Sie bietet eine große Zahl an vordefinierten Funktionen und erlaubt auch die Definition eigener Funktionen.

Die Erweiterung der Funktionalität gegenüber Quilt umfasst die automatische Typumwandlung bei zu vergleichenden Typen. Datentypen können durch reguläre Ausdrücke definiert werden und zur statischen Typprüfung bei der Konstruktion von Elementen

verwendet werden.

Name	Sprache/Plattform	Quelle
XStreamDB	Java	http://www.bluestream.com
Qexo	Java	http://www.gnu.org/software/qexo/
IPSI-XQ	Java	http://www.ipsi.fraunhofer.de/oasys
XQBE	Java	http://dbgroup.elet.polimi.it/xquery/
Saxon	Java	http://saxon.sourceforge.net/
RainbowCore	Java	http://davis.wpi.edu/ dsrg/rainbow/
Qizx/open	Java	http://www.xfra.net/qizxopen/
Galax	O'Caml/C/C++	http://db.bell-labs.com/galax/
XQuark	MySQL/Oracle/...	http://xquark.objectweb.org/
phpxmlclasses	PHP	http://sourceforge.net/projects/phpxmlclasses/

Abb. 3.30: Implementierungen von XQuery

Aufgrund der Standardisierungsbemühungen des W3C, welche auch XML und XSLT hevorgebracht haben, wird die Entwicklung und weitere Integration von Anfrageparadigma von XQuery sehr aktiv vorangetrieben. Es existieren daher eine ganze Reihe von Implementierungen, die im SIES eingesetzt werden können (s.Abb. 3.30).

3.2.10 XML-GL

Anfragen werden in XML-GL im Gegensatz zu allen anderen Sprachen graphisch formuliert. Das Ziel war es, die Formulierung von Anfragen zu vereinfachen und einen intuitiven Zugang zu ermöglichen. Bei einer XML-GL Anfrage werden in einer vertikal geteilten Ebene die graphischen Elemente der Anfrage miteinander verbunden und so die Anfrage aufgebaut.

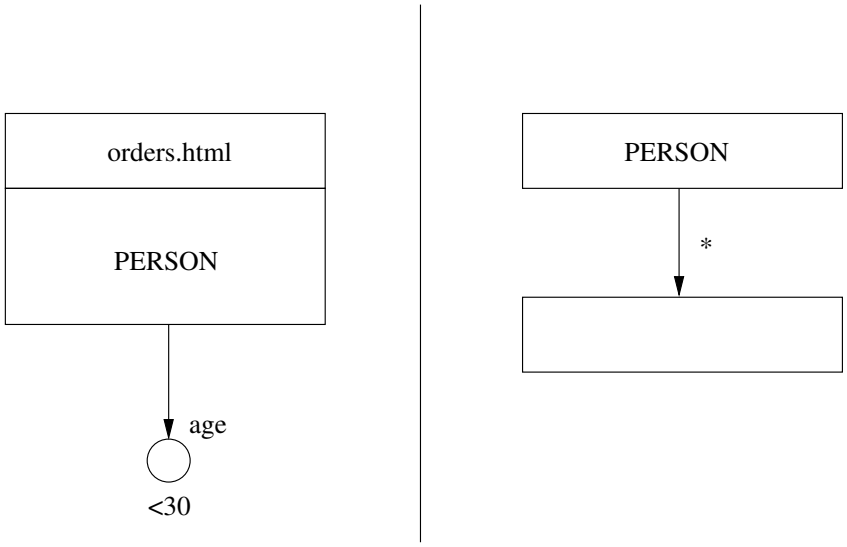


Abb. 3.31 : Einfache XML-GL Anfrage

Auf der linken Seite der Anfrageebene wird die Analyse des Quelldokuments, auf der rechten Seite die Konstruktion des Zieldokuments dargestellt (siehe Abb. 3.31). Elemente des Analyse- und Konstruktionsabschnitts werden implizit durch gleiche Namensgebung oder explizit durch Linien verbunden, falls die Namen sich nicht entsprechen.

Das Datenmodell von XML-GL umfasst abstrakte Objekte, Eigenschaften und Relationen der Objekte untereinander. In Bezug auf die Darstellung von XML-Daten in diesem Datenmodell werden XML-Elemente (Objekte) als Rechtecke, Eigenschaften (Attribute) als Kreise und die Beziehungen (Schachtelung der Elemente) untereinander als gerichtete Kanten dargestellt.

XML-GL Anfragen können sich auf ein oder mehrere XML-Dokumente beziehen und diese zu einem neuen Dokument vereinen.

Einfache Anfragen werden in XML-GL als Extract-Match-Clip Anfragen bezeichnet (s.Abb. 3.31). Der Extract-Teil wählt in Form einer Pfadangabe ausgehend von Name und Adresse des XML-Dokuments die zu betrachtenden Abschnitte aus. Der optionale Match-Teil besteht aus der Annotierung von Bedingungen an die einzelnen im Extract-Teil ausgewählten Elemente. Diejenigen Elemente, die die Bedingungen erfüllen, werden im Clip-Abschnitt weiterverarbeitet. Der Clip-Teil beschreibt eine Art Filterung der zurückzugebenden Elemente, indem beschrieben wird, welche Information beibehalten werden soll.

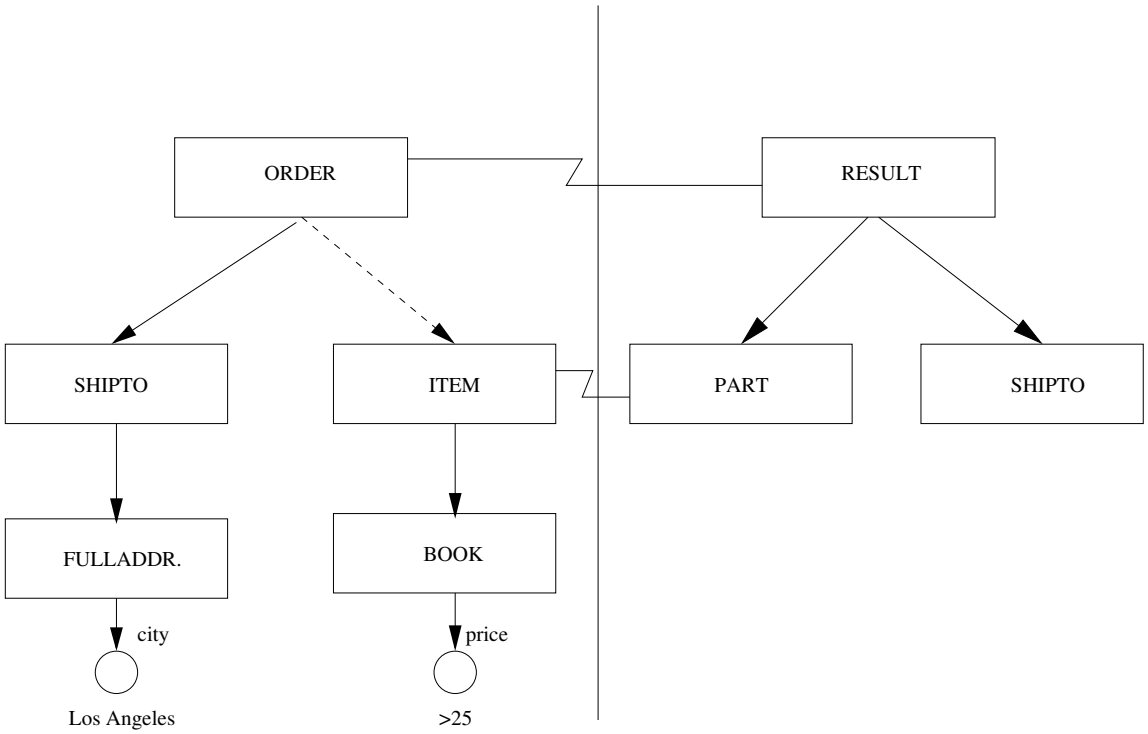


Abb. 3.32 : Komplexe XML-GL Anfrage

Bei Anfragen, die eine Umstrukturierung oder einen Neuaufbau des Ausgabedokuments beinhalten, spricht man von Extract-Match-Construct-Clip Anfragen (s.Abb. 3.32). Hierbei wird durch einen Construct-Teil die Möglichkeit gegeben, mittels Konstruktoren

neue Elemente zu erzeugen und bestehende umzustrukturieren.

Leider unterstützt XML-GL keine ineinander geschachtelten Anfragen und auch keine Funktionen [Cer 99]. Somit können komplexere Anfragen nur mit Mühe durch hintereinanderausführen und vorheriges zurechtlegen der Teildaten realisiert werden. Von XML-GL ist keine Implementation verfügbar. Lediglich prototypische Bemühungen, XML-GL Anfragen automatisch in die textuellen Sprachen Lorel, XQL und Quilt zu übersetzen sind bekannt, aber nicht frei erhältlich. Die Ausdruckmöglichkeiten von XML-GL werden durch einige dieser Sprachen mindestens erfüllt (Aufzählung, Quelle), so dass nur der Vorteil der Eingabeoberfläche hervorzuheben ist.

Eine Integration von XML-GL in SIES wäre daher mit einigem Programmieraufwand verbunden. Möglich wäre eine Übersetzung in einen bestehende textuelle Sprache.

3.3 Weitere Sprachen

Es gibt noch eine Reihe weiterer XML-Anfragesprachen, deren Zielsetzung jedoch zu spezialisiert ist, um sie als Transformationssprachen einzusetzen.

XMAS

XMAS wurde entwickelt, um in Interaktion mit einem Benutzer große Datenbestände zu bearbeiten. Die aktuelle Position des Benutzers in einem Datenbestand wird als **VIEW** bezeichnet. Eine Anfrage berücksichtigt nur die Daten der aktuellen **VIEW**. Bewegt sich der Benutzer so wird diese Information an die Operatoren der Anfrage weitergeleitet und es wird eine neues Ergebnis generiert ([Lud00]).

Die Syntax von XMAS ähnelt der von XML-QL.

```
CONSTRUCT <answer>
             <med_home> $H
                        $S {$S}
             </med_home> {$H}
          </answer> {}

WHERE homesSrc homes.home $H AND $H zip._ $V1
AND   schoolsSrc schools.school $S AND $S zip._ $V2
AND   $V1 = $V2
```

Abb. 3.33: XMAS

Name	Sprache/Plattform	Quelle
MIX	Java	http://www.db.ucsd.edu/Projects/MIX/

Abb. 3.34: Implementierungen von XMAS

XIRQL

XIRQL wurde entwickelt, um ein Information-Retrieval auf XML-Dokumenten durchzuführen [Fuh 01]. Dies wird unterstützt durch die Möglichkeit, eine Suche zu gewichten und die Ergebnisse bewerten zu lassen. Dabei kann die Suchanfrage vage gehalten werden. Vorraussetzung für eine XIRQL-Anfrage ist jedoch ein XML-Dokument, in dem Termwichtungen vorberechnet sind (s.Abb. 3.6).

```
<chapter> 0.3 XQL
  <section> 0.5 example </section>
  <section> 0.8 XQL 0.7 syntax </section>
</chapter>
```

Abb. 3.35: Dokument mit Termwichtung

3.4 Zusammenfassung und Bewertung

In diesem Kapitel wurden die verfügbaren Anfragesprachen und ihre Implementierungen vorgestellt. Die Übersicht über die zeitliche Entwicklung gibt Aufschluß über die konzeptionellen Abhängigkeiten und Erweiterung der einzelnen Sprachen.

In Abb.3.36 ist zusammenfassend die Verfügbarkeit von Implementierungen der einzelnen Sprachen dargestellt.

Da XQuery als der neue Standard des W3C für XML-Anfragesprachen konzipiert wurde, bietet es die umfangreichste Auswahl an Implementierungen. Lorel steht nur auf ausgewählten Unix-Plattformen zur Verfügung. Von XML-GL sind keine Implementierungen bekannt. Von den Autoren von XML-GL wurden jedoch einige Versuche unternommen, XML-GL Anfragen durch textuelle Anfragesprachen auszudrücken,

	XML-QL	YATL	Lorel	XQL	Quilt	XQuery	XML-GL
Java	X	-	-	X	X	X	-
C	-	-	-	-	-	X	-
C++	-	-	-	-	-	X	-
Cl./Server	-	-	-	-	-	X	-
Andere	-	X	X	X	-	X	-

Abb. 3.36: Vergleich der Implementierungen

Anfragen in XML-GL sind wegen des graphischen Ansatzes leicht verständlich. Die Anfragemöglichkeiten sind aufgrund fehlender Schachtelungen und Funktionen nicht so umfangreich wie in anderen Sprachen. Bei XQL ist die Möglichkeiten zur Konstruktion eines XML-Ausgabedokuments sehr beschränkt und nicht vom Eingabedokument unabhängig realisiert. XQL ist damit zur Transformation von XML-Dokumenten nicht geeignet.

Zur Vorauswahl der im Folgenden zu betrachtenden Sprachen sei auch auf eine Untersuchung des Funktionsumfangs einiger Sprachen verwiesen [Bon 01], der in Abb.3.37

dargestellt wird. XQL und XML-GL werden gehen aus oben genannten funktionalen Defiziten nicht in die weitere Untersuchung ein.

Das verwendete Datenmodell hat weitreichenden Einfluß auf die möglichen Operationen einer Anfragesprache. Durch die Definition atomarer Typen können automatische Typumwandlungen erfolgen. Eine Ordnung im Datenmodell, die aus dem zugrundeliegenden XML-Modell gewonnen wird, trägt zur Ausdruckstärke der Transformationen bei.

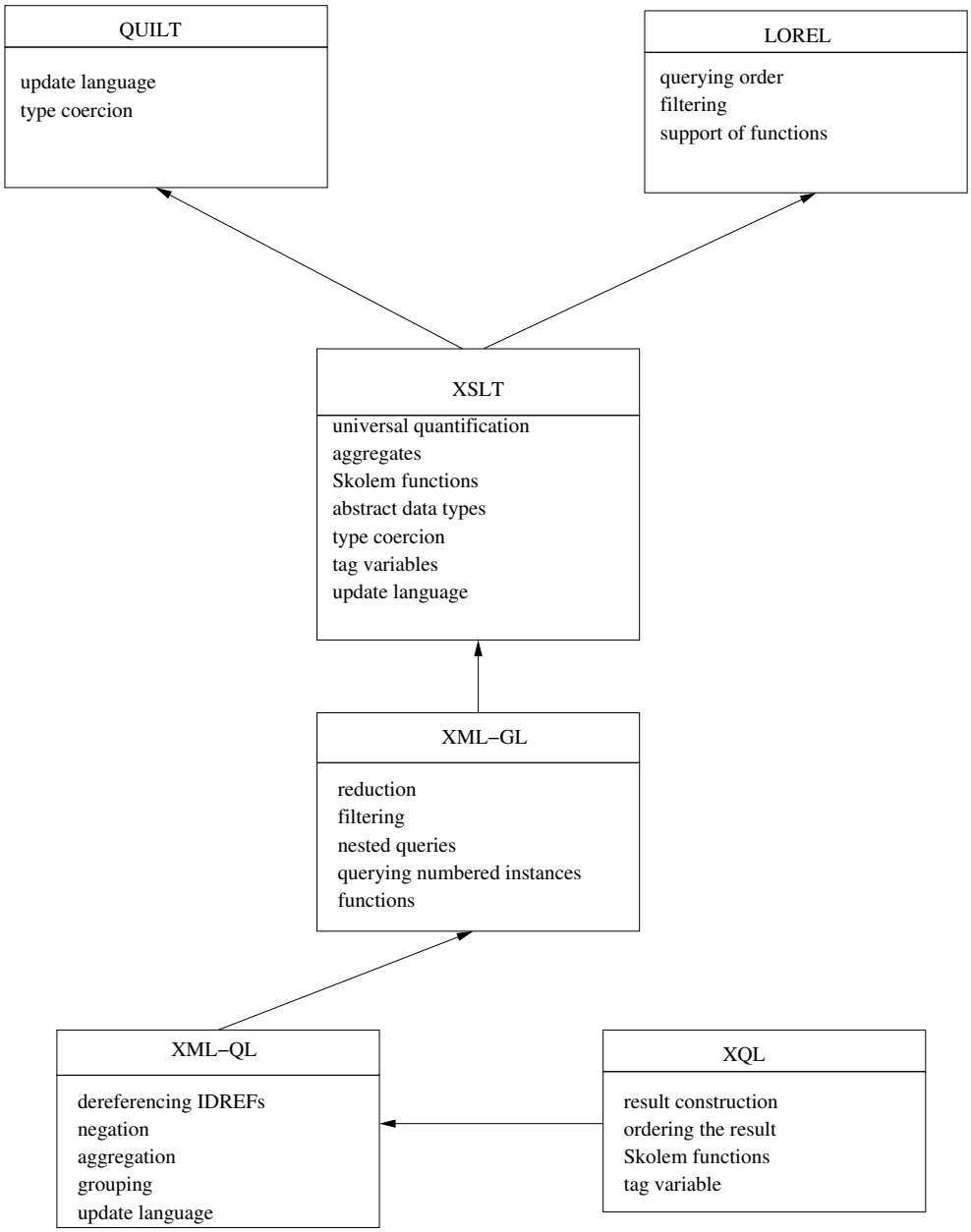


Abb. 3.37 : Klassifikation (nach [Bon 01])

Es wurden einige herausragende Anfragesprachen kurz vorgestellt, die entwickelt wur-

den, um XML als Datenbasis zu bearbeiten. Ein wesentliches Element, das für die Eignung zur Transformation von XML entscheidend ist, ist die Trennung von Analyse und Generierung. Dies wird im folgenden Kapitel genauer untersucht werden. Hauptkriterium dieses Kapitels war die Verfügbarkeit in Form einer Implementation und die Möglichkeit der Integration in SIES.

4 Transformation mit XML-Anfragesprachen

Im Folgenden werden die sprachlichen Konzepte von XML-Anfragesprachen untersucht, mit der Zielsetzung eine Transformation von XML-Dokumenten in XML-Dokumente durchzuführen.

Da XML-Anfragesprachen im Gegensatz zu Transformationssprachen wie XSLT, zur Abfrage von Information aus XML-Dokumenten entwickelt wurden, ist ihre Mächtigkeit primär auf diesen Zweck konzentriert. In vielen Anfragesprachen wird jedoch zusätzlich die Möglichkeit geboten, den Ergebnisaufbau zu beeinflussen, wodurch sich die Umsetzung einer Transformation mit einer Anfragesprache realisieren lässt. Soweit möglich, wird die Funktionalität zur Abfrage getrennt von den geeigneten Methoden zur Transformation behandelt, da eine hohe Mächtigkeit zur Formulierung von Anfragen bei einer Anfragesprache vorausgesetzt wird und nicht zentraler Gegenstand dieser Untersuchung sein soll.

Im Anschluss sollen relevante Abfragemöglichkeiten auch bei XML-Transformationen berücksichtigt werden. Es wird zunächst versucht, eine Menge von Konstrukten zu bestimmen, die von den meisten Sprachen unterstützt werden. Darauf aufbauend wird auf Besonderheiten der jeweiligen Sprachen eingegangen, sofern sich diese zur XML-Transformation eignen.

Die Transformation eines XML-Dokuments lässt sich aus Sicht der Struktur- und der Informationsebene in drei Schritten darstellen (s.Abb. 4.1).

	Struktur	Information
1.	Analyse	Selektion
2.	Transfer	Extraktion
3.	Generierung	Kombination

Abb. 4.1: Transformationsschritte

Im Analyseschritt wird versucht, in einer gegebenen Informationsstruktur die zur Beantwortung der Anfrage relevanten Abschnitte aufzufinden. Die hier enthaltene Information wird anschließend extrahiert und genauer geprüft, ob sie vorgegebenen Kriterien entspricht. Dies wird als Transferschritt bezeichnet und beinhaltet auch eine strukturelle oder inhaltliche Aufbereitung der Information. Zum Schluss wird die aufbereitete Information in gegebenenfalls neuer Darstellung als Antwort der Anfrage zurückgegeben. Der Datenfluss durch die Schritte eines sogenannten FLWR-Ausdrucks ist am Beispiel von Quilt in Abb. 4.2 dargestellt.

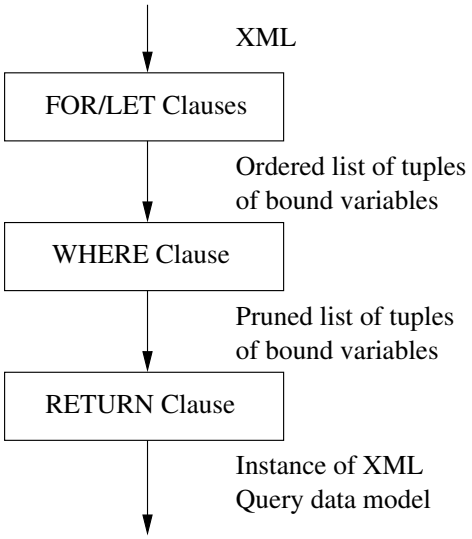


Abb. 4.2 : Datenfluss eines FLWR-Ausdrucks in Quilt (nach [Cha 00])

Dieser Datenfluss beschreibt die Kopplung der einzelnen sprachlichen Elemente wie Analyse (**FOR/LET**), Transfer (**WHERE**) und Rückgabe (**RETURN**).

Da jedoch Anfragen auch rekursiv in Anfragen eingebettet werden können, verlaufen die Grenzen der einzelnen Schritte nicht immer scharf. Auch können vielen Methoden der Anfragesprachen in unterschiedlichen Schritten eingesetzt werden. Es werden daher zunächst diejenigen Methoden beschrieben, die einer bestimmten Stufe der Transformation eindeutig zugeordnet werden können.

4.1 Analysemethoden

Im Analyseschritt wird versucht im Quelldokument Informationen zu lokalisieren. Hierzu kommen verschiedene Methoden wie Navigation, Filterung und Knotentests zum Einsatz. Ziel des Analyseschritts ist es, die relevanten Daten an Variablen zu binden, um sie im nachfolgenden Konstruktionsschritt einsetzen zu können.

4.1.1 Addressierung des Ausgangsdokuments

Bei der Bearbeitung muss einem Anfrageprozessor mitgeteilt werden, auf welches Dokument sich eine Anfrage bezieht, d.h. auf welcher Elementmenge eine Anfrage ausgeführt werden soll. Mit Angabe des Quelldokuments erfolgt gleichzeitig die Auswahl des Startknotens der Elementmenge.

Eine wichtiges Merkmal ist neben der Bearbeitung lokal verfügbarer Dokumente, die Referenzierung und Bearbeitung von Dokumenten, die in einem Rechnernetz verteilt sind. Diese Ressourcen werden durch einen URI identifiziert und gewöhnlich über das HTTP-Protokoll übertragen (s.Abb. 4.3).

```
  http://www.domain.com/pfad/datei.xml
Protokoll     DNS       Pfad+Ressource
```

Abb. 4.3: mögliche Zusammensetzung eines URI

Ein URI beginnt mit dem Namen des Übertragungsprotokolls, auch Schema genannt. Das Schema definiert die Interpretation der folgenden Angaben. Im Falle eines URL (Subklasse des URI zum Zugriff auf das Internet), folgen dem Schema die Ortsangabe des Rechners, auf dem sich die Ressource befindet, sowie der Pfad zum Zugriff.

```
match "http://www.domain.com/datei.xml" with Muster
```

Abb. 4.4: Dokumentenzugriff in YATL

Bei YATL ist der URI nach der **match**-Anweisung zu erkennen (s.Abb. 4.4), an die sich eine Musterspezifikation anschliesst, um relevante Strukturen zu beschreiben.

Bei Lorel ist es nicht möglich, einen URI anzugeben. Einstiegspunkt in ein Dokument ist hier der Dokumentenname ohne Dateityperweiterung. Der Dokumentenname ist somit der Startknoten, auf den sich weitere Pfadangaben beziehen. Dies bedeutet, dass zu bearbeitende Dokumente auf einem lokalen Dateisystem vorrätig sein müssen.

```
Pfadausdruck:
 from Knoten1.Knoten2 Alias Name

Datei: test.xml
       from test.elementA Alias testA
```

Abb. 4.5: Dokumentenzugriff in Lorel

Bei Lorel wird nicht der vollständige Dateiname übergeben, sondern der Name eines Startknotens, der dann auf einen Dateinamen abgebildet wird (Knoten1). Dem Knoten1 untergeordnete Knoten (Knoten2) bezeichnen als Pfadausdruck untergeordnete Strukturknoten innerhalb des Dokuments (s.Abb. 4.5).

In XQuery ist es möglich, einen Referenz-URI zu definieren (declare base, s.Abb. 4.6), auf den sich alle weiteren Dokumentzugriffe beziehen. Der Zugriff auf ein Dokument erfolgt über die Funktion fn:doc, die durch das eingebettete XPath bereitgestellt wird [W3C 03/2]. Der Rückgabewert der Funktion fn:doc ist ein Verweis auf den Wurzelknoten des XML-Dokuments, der als Ausgangspunkt für den XPath-Ausdruck dient.

```
XQuery:
 declare base-uri "http://example.org";

 for $a in doc("test.xml")/XPath-Ausdruck
 let $b in doc("test.xml")/XPath-Ausdruck
```

Abb. 4.6: Dokumentenauswahl in XQuery

Eine Spezialität des SIES ist es, dass hier die Dokumente über selbstdefinierte URIs

angesprochen werden. Die Auflösung der Adresse erfolgt über einen URI-Resolver. Die Verwendung eines URI-Resolvers ist in den Sprachen nicht definiert und erfordert daher eine Unterstützung oder Erweiterbarkeit seitens des verwendeten Anfrageprozessors. Mit der Ausnahme von Lorel unterstützen alle Sprachen zumindest die Auflösung eines HTTP-URI, so dass die Umsetzung des Zugriffs, falls nicht anders möglich, mit Hilfe eines HTTP-Servers durchgeführt werden kann. Dazu kann das zu bearbeitende Dokumente unter einem eindeutigen Namen verfügbar gemacht und danach dynamisch in eine Transformationsanfrage integriert werden.

Alle Anfragesprachen unterstützen die Verarbeitung lokal verfügbarer Dokumente. Dabei wird anstatt des URI der Name und Pfad des Dokuments in einem Dateisystem übergeben. Somit wäre es auch denkbar, die zu bearbeitenden Dokumente vorab zu beschaffen und in geeigneter Form durch Generierung dynamischer Namen bereitzustellen.

4.1.2 Navigation und Selektion

Zum Lokalisieren von relevanten Daten in einem Ausgangsdokument muss die Möglichkeit bestehen, durch die XML-Struktur zu navigieren. Bei der Navigation wird ein Pfadausdruck verwendet, der die Abfolge und Schachtelung relevanter Objekte des Ausgangsdokuments spezifiziert. Die Methodik der Navigation hängt dabei sehr stark vom Modell ab, das die Anfragesprache einem XML-Dokument zu Grunde legt.

Manche Sprachen wie Quilt oder XQuery betten zur Navigation die eigentständige Lokalisierungssprache XPath ein, deren zugrundeliegendes Datenmodell eine sehr flexible Navigation in einer hierarchischen Struktur bietet.

Da die Struktur des Ausgangsdokumentes unbekannt oder tief verschachtelt sein kann, ist es nötig, eine Lokalisierung auch unter Angabe von nicht vollständigen Pfadausdrücken vornehmen zu können. Die Möglichkeiten der einzelnen Sprachen sind in Abb. 4.7 dargestellt.

	absolute Navigation	relative Navigation	bedingte Navigation
XML-QL	Ja	Ja	Ja
YATL	Ja	Nein	partiell
Lorel	Ja	Ja	Ja
Quilt	Ja	Ja	Ja
XQuery	Ja	Ja	Ja

Abb. 4.7: Navigation

Die mächtigsten Navigationsfunktionen stellt Lorel bereit. Hierzu werden umfangreiche Pfadausdrücke definiert, die in allen Teilbereichen (select, from, where) einer Anfrage verwendet werden können. Pfade können durch reguläre Ausdrücke beschrieben werden (vgl. Navigation ohne XPath). Über unbekannte Schachtelungsstrukturen kann mit zwei Wildcard-Typen navigiert werden. Einmal ausgewählte Pfade können in Pfadvariablen abgelegt werden, mit deren Hilfe Pfade verglichen und auch ausgegeben werden können [Abi 97].

Navigation mit XPath

Quilt bettet zur Navigation XPath ein. XPath wählt Mengen von gültigen Knoten aus der angegebenen Menge (document) aus und gibt sie zurück. Die Spezifizierung eines Pfades erfolgt anhand des XPath-Datenmodells. Das bedeutet, dass durch die XPath-Sprache Verwandtschaftsbeziehung der Knoten untereinander abgeprüft werden können.

```
<result>
  (
    FOR $i IN document("items.xml")/items/item_tuple,
    LET $b := document("bids.xml")//bid_tuple[itemno = $i/itemno]
    RETURN $b
  )
</result>
```

Abb. 4.8: Navigation in Quilt

In diesem Beispiel (s.Abb. 4.8) werden die Elemente "item_tuple" und "bid_tuple" gesucht. XPath kennt dabei zwei allgemeine Pfadoperatoren, absolut ("/") und beliebige Tiefe ("//"). Das Element "item_tuple" wird dabei absolut adressiert, das Element "bid_tuple" kann in beliebiger Tiefe im Dokument bids.xml vorkommen und wird an die Variable b gebunden, wenn ein Knotentest (itemno = $i/itemno) als wahr ausgewertet wird.

Weiterhin kann durch die Funktionen des XPath-Datenmodells auf Elemente zugregriffen werden (z.b. following-sibling)(vgl.Kap.3). Quilt benutzt eine vereinfachte Form von XPath und erweitert es um Funktionen wie **BEFORE, AFTER**, **INTERSECT** und **EXCEPT** ([Cha 00]), deren Funktionalität erst in XPath 2.0 nachgerüstet wird [W3C 03/3].

Navigation ohne XPath

Am Beispiel von XML-QL wird die Navigation gezeigt, wenn nicht unter Verwendung von XPath, sondern durch Musterangabe navigiert wird. Bei einer XML-QL Anfrage wird das Ausgangsdokument durch den **WHERE**-Abschnitt gefiltert. Es wird nach einer Schachtelung nach Art des angegebenen Musters gesucht. Ein Muster besteht dabei aus der gleichen XML-Struktur, die im Quelldokument identifiziert werden soll. [Flo 99].

Dieses Muster kann an beliebiger Stelle im Quelldokument stehen, falls es nicht den Root-Knoten mit einbezieht. Alle Elemente des Musters, die keine Variablen sind, dienen als Navigationsbedingungen.

```
WHERE <$*>
        <part *> <name> $r </> <brand> Ford </> </> </>
       IN "www.a.b.c/bib.xml"
```

Abb. 4.9: Navigation in XML-QL

Die Beschreibung des Musters geschieht in XML-QL (s.Abb. 4.9) durch einen regulären

Pfadausdruck, in dem durch Variablen an den als relevant betrachteten Stellen die Daten des Quelldokuments gebunden werden. Zur Beschreibung des Pfadausdruckes stehen Disjunktion, Konkatenation, der Kleensche Stern-Operator und ein Wildcard-Ausdruck zur Verfügung [Flo 99].

4.1.3 Analyse ohne Strukturwissen

Nicht immer ist für das Ausgangsdokument eine DTD oder ein Schema vorhanden, das die Struktur eines validen XML-Dokuments beschreibt. Auch wird ein Schema meist ausschließlich zur Validierung von Dokumenten eingesetzt. Gewöhnlich wird daher eine Anfrage so geschrieben, dass sie nur auf bekannten Strukturen arbeiten kann und nicht strukturunabhängig ist.

Ein Problem hierbei ist es, die relevante Information trotz unbekannter Struktur des Quelldokuments aufzufinden. Sprachen wie XML-QL benötigen hierzu kein vorliegendes Schema, da sogenannte Tag-Variablen unterstützt werden [Flo 99/2]. Tag-Variablen dienen als Platzhalter für - und zur Abfrage von unbekannten Tags.

```
WHERE <bib>
        <$p> <title> $t </title> </>
      </> IN "..."
CONSTRUCT <$p> <title> $t </title> </>
```

Abb. 4.10: Tag-Variablen in XML-QL

In XML-QL wird eine Tag-Variable durch eine in spitzen Klammern umschlossene Variable ausgedrückt (s.Abb. 4.10) und kann somit sehr einfach in einem Muster verwendet werden. Tag-Variablen werden auch von Quilt und XQuery angeboten. Das Konzept der Tag-Variablen ist nur in den Anfragesprachen zu finden, in XSLT werden sie nicht unterstützt.

4.1.4 Bedingungen

Bei der Prüfung einer Bedingung muss ein Vergleich aller Strukturelemente eines XML-Baumes möglich sein.

```
FOR $h IN //holding
RETURN
    <holding>
    $h/title,
    IF $h/@type = "Journal"  THEN $h/editor
    ELSE $h/author
   </holding> SORTBY (title)
```

Abb. 4.11: Bedingungen in Quilt

In gezeigtem Beispiel (s.Abb. 4.11) in der Sprache Quilt wird der Wert eines Attributes überprüft und in dessen Abhängigkeit eine Entscheidung über die Rückgabestruktur getroffen [Cha 00]. Das Problem bei einem Vergleich ist, dass ohne eine Funktion zur Typerzwingung nur atomare Werte des jeweiligen der Sprache zu grundeliegenden Datenmodells verglichen werden können. Die Typerzwingung (s.Abschnitt 4.2.4) wird nur von Lorel und XQuery unterstützt. Wenn eine Sprache keine Typerzwingung beherrscht, so muss der Benutzer bei der Formulierung der Anfrage dafür Sorge tragen, dass zu vergleichende Daten entsprechend kompatibel sind oder explizit umgewandelt werden. Dies schränkt natürlich die Transformation unbekannter Daten ein.

4.1.5 Quantoren

Existenz und Allquantoren sind in den meisten Sprachen verfügbar. XML-QL kennt keinen Allquantor. Mit ihnen kann die Existenz bestimmter Knoten getestet werden (s.Abb. 4.12).

```
for $a in document("test.xml")/knotena
where every $a in $a/knotenb satisfies
      Boolesche-Funktion($a)
return $a
```

Abb. 4.12: Quantoren in XQuery

4.2 Transfermethoden

Ein wichtiges Element des Transfers ist die Extraktion und Speicherung der Information in einer Form, die von der Ausgangsstruktur unabhängig ist. Diese abstrakte Repräsentation erfordert die Zwischenspeicherung der Information in Variablen, deren Belegung in der Zielstruktur ausgewertet wird.

4.2.1 Variablenbindung

Da in den Anfragesprachen die Analyse und Generierung getrennt formuliert werden, bedarf es eines Bindeglieds, um Ergebnisse der einen Phase in die andere übertragen zu können. Die Verknüpfung zwischen Analyse (**match**) und Generierung (**make**) erfolgt in YATL wie auch in den anderen Sprachen über Variablen.

```
make
   d [ *b [ @c [ $y ] ] ]

match "http://www..." with
   a [ *b [ @c [ $y ] ] ]

where
   $y > ...
```

Abb. 4.13: Variablenbindung in YATL

Dabei wird die Variable an diejenigen Elemente gebunden, die durch das Muster der **match**-Anweisung im Quelldokument identifiziert werden (s.Abb. 4.13). In der Konstruktionsanweisung wird der Inhalt der Variablen ausgewertet und eingesetzt. In XML-GL ist dieser Mechanismus ähnlich, es wird jedoch ermöglicht, nicht nur einzelne durch Muster belegte Variablen, sondern auch die Muster selbst durch den **CONTENT_AS**-Operator zugreifbar zu machen.

YATL, XQuery und andere Sprachen unterscheiden bei der Variablenbindung Methoden zur sequentiellen und parallelen Bindung. Dies wird am Beispiel von XQuery gezeigt.

```
Sequentielle Bindung:
for $a in doc()/X-Path-Ausdruck

Parallele Bindung:
let $b := doc()/X-Path-Ausdruck
```

Abb. 4.14: Variablenbindung in XQuery

Mit einem for-Ausdruck wird eine Variable sequentiell belegt, und anschliessend durch die einzelnen möglichen Werte iteriert. Der let-Ausdruck bindet alle möglichen Werte als Menge auf einmal an die Variable, was einer parallelen oder Tupel-Bindung entspricht (s.Abb. 4.14).

4.2.2 Verwendung: Variablenmodifikation/-Auflösung

Variablen können Elemente oder atomare Werte beinhalten. Man kann durch Funktionen den Rückgabewert einer Variablen bestimmen. Z.b. gibt es in XPath die Funktion text(), durch die nur die textuelle Information eines Elementes berücksichtigt wird.

Bei der Spezifikation von XQuery wurde festgelegt, dass Variablen immer als Kopie ("copy by value") und nicht als Referenz übergeben werden. Dies hat zufolge, dass einmal belegte Variablen nicht mehr geändert werden können. Der Gedanke dahinter war, dass Seiteneffekte minimiert werden sollen ([W3C 03]). Man muss daher die Anfragen so gestalten, dass z.B. Werte einer Iteration vorher bestimmt und übergeben werden.

Da dies jedoch teilweise sehr umständlich ist, haben einige Implementationen einen

zusätzlichen **assign**-Operator eingeführt, der die Änderung von Variablenbelegungen ermöglicht.

4.2.3 Iteration

Für eine Transformation ist es wichtig, über die Elemente eines Quelldokuments iterieren zu können. In XQuery und Quilt wird dies über den FLW(O)R-Ausdruck ermöglicht . Dieser stellt eine **for**-Klausel bereit, mit der über die Ergebnisse eines XPath-Ausdruckes iteriert werden kann. Wichtig hierbei ist der Erhalt der Ordnung der Struktur des Quelldokuments, falls das zugrundeliegende Datenmodell eine solche beinhaltet und diese für die Konstruktion des Ergebnisses relevant ist. Eine Umsortierung der Elemente muss explizit definiert werden.

```
for $a in doc()
for $a in $b to $c
```

Abb. 4.15: Iteration in XQuery

4.2.4 Typerzwingung

Die Typerzwingung wird zum Vergleichen von Werten verschiedener Typen und anderen Funktionen (arithmetisch, Aggregation) benötigt. Man unterscheidet dabei den Vergleich atomarer Werte und komplexer Objekte. Die Typerzwingung ist nur in Lorel und XQuery vorhanden. Sie ist eine der wichtigsten Erweiterungen von XQuery gegenüber Quilt.

Sind bei einer Vergleichsoperation die Typen von atomaren Werten nicht kompatibel, so stellt die Typerzwingung ein Regelwerk zur Umwandlung der beteiligten Werte bereit. Die Regeln für Lorel sind in Abb. 4.16 gezeigt.

arg1/arg2	string	real	int
string	-	string → real	both → real
real		-	int → real
int			-

Abb. 4.16: Typerzwingung bei atomaren Typen in Lorel (aus [Abi 97]).

Alle anderen atomaren Typen sind in Lorel nicht vergleichbar (z.b. gif, audio) und liefern bei einem Vergleich **false**.

arg1/arg2	value	atomic object	set of objects	complex object
value	coerce	dereference	existential with ==	false
atomic object		value =	existential with ==	false
set of objects			existential with ==	false
complex object				value =

Abb. 4.17: Typerzwingung bei Werten von Objekten in Lorel (aus [Abi 97]).

In XQuery wird die Typerzwingung durch den **typeswitch**-Ausdruck ermöglicht [W3C 03].

```
typeswitch($customer/billing-address)
   case $a as element(*, USAddress) return $a/state
   case $a as element(*, CanadaAddress) return $a/province
   case $a as element(*, JapanAddress) return $a/prefecture
   default return "unknown"
```

Abb. 4.18: Dynamische Typen in XQuery

Durch die Angabe der Variable "$a" innerhalb der case-Anweisung wird diese deklariert. Innerhalb der ausgeführten case-Anweisung wird sie an das Auszuwertende Element gebunden und dadurch die Art der Rückgabe bestimmt.

4.2.5 Skolem-Funktionen

Durch die Verwendung von Skolem-Funktionen kann der Aufbau einer Struktur gezielt gesteuert werden. Dabei werden Objekten gesteuert durch Eigenschaften der Objekte ein eindeutiger Bezeichner zugewiesen. Treten bei einem zweiten Objekt die gleichen Eigenschaften auf, so wird der gleiche Bezeichner wie beim ersten Objekt vergeben. Dadurch kann zueinandergehörige Information integriert werden.

Skolemfunktionen werden von Quilt, XQuery, Lorel und XML-QL (s.Abb. 4.19) unterstützt.

```
CONSTRUCT <person ID=PersonID($fn,$ln)
           ...
          </>
```

Abb. 4.19: Skolem-Funktion in XML-QL

4.2.6 Einschachtelung von Anfragen

Eine komplexe Anfrage kann nicht immer in einem Frage-Block formuliert werden. Um die Notwendigkeit mehrer hintereinanderauszuführender Anfragen zu vermeiden, stellen die meisten Anfragesprachen die Möglichkeit, Anfragen in Anfrage einzubetten. Dabei dienen die eingebetteten Anfrage als Funktion, deren Rückgabewerte in der äusseren Anfrage verarbeitet werden.

Alle hier betrachteten Sprachen unterstützen die Schachtelung von Anfragen in Anfragen und die Verarbeitung von Rückgabewerten einer eingeschachtelten Anfrage.

4.3 Generierung von XML-Strukturen

Alle Anfragesprachen besitzen einen Rückgabeblock, in dem durch Konstruktoren neue Strukturen erzeugt werden können. Man unterscheidet dabei Element und Attribut-Konstruktoren.

4.3.1 Aufbau von Strukturen

Das Ergebnis einer Anfragesprache wird explizit in einem Rückgabeteil aufgebaut. Hierzu verfügen alle Anfragesprachen über Elementkonstruktoren, mit denen gemäß des zugrundeliegenden Datenmodells neue Strukturen aufgebaut oder bestehende übernommen werden können.

Lorel

Bei Lorel können in der select-Anweisung neue Elemente durch den xml()-Ausdruck erstellt werden. Dem Ausdruck werden Typ (optional), Name und Werte sowie weitere eingeschachtelte Elemente als Pfadausdrücke oder Variablen übergeben. Attribute werden nicht berücksichtigt (s.Abb. 4.20).

```
select xml(results: a)
from bib.book a
```

Abb. 4.20: Elementkonstruktor bei Lorel

Quilt/XQuery

Zu konstruierende Elemente werden in der **return**-Anweisung beschrieben. Dabei werden in geschweiften Klammern geschriebene Anweisungen und Variablen als sprachlicher Ausdruck interpretiert und das Ergebnis in die umgebende Struktur eingesetzt (s.Abb. 4.21).

```
for ...
let ...
where ..
return <element attribut="Wert">
        { XQuery }
       </element>
```

Abb. 4.21: Element- und Attributkonstruktor bei Quilt/XQuery

XML-QL

In XML-QL wird der Aufbau der Rückgabestrukturen durch die CONSTRUCT-Anweisung beschrieben. Darin werden die Werte der in der WHERE-Anweisung gebundenen Variablen eingesetzt.

Der Inhalt der CONSTRUCT-Anweisung beschreibt dabei ein Muster, das für alle existierenden Variablenbindungen erzeugt und ausgegeben wird.

YATL

Die Konstruktion des Ergebnisdokuments erfolgt in YATL in einer **make**-Anweisung. Hierbei wird nur die zulässige Elementschachtelung angegeben. Für jedes Element wird annotiert, ob es ein- oder mehrmals vorkommen darf. Die Anzahl der erzeugten Elemente hängt von der Anzahl möglicher sequentieller Variablenbelegungen ab, deren Werte in die Elemente eingesetzt werden.

4.3.2 Update-Language

Eine sogenannte Update-Language bezeichnet Methoden innerhalb einer Anfragesprache, um kleine Änderungen an einer Quellstruktur möglichst einfach auszudrücken. Die Menge der Operationen einer Update-Language umfasst das Einfügen, Löschen und Aktualisieren von Strukturen. Mit dem Begriff "Aktualisieren" sind strukturerhaltende Wertanpassungen gemeint.

Durch eine Update-Language kann bei kleinen Änderungen die manchmal aufwendige Rekonstruktion eines Quelldokuments durch einen Generierungsschritt vermieden werden. Von den hier untersuchten Sprachen verfügt nur Lorel über Update-Methoden. Für XQuery gibt es mehrere Vorschläge, die bisher noch nicht in den Standard aufgenommen wurden (Galax u. XQuery Update).

4.4 Transformationsoperationen

Als Transformationsoperationen bei einer Anfragesprache wird die Rekonstruktion der Information einer Ausgangsstruktur bezeichnet, so dass eine Transformation beschrieben wird. Dies bedeutet, dass sie unter Umständen durch das Zusammenspiel mehrerer Methoden einer Anfragesprache ausgedrückt werden muss.

Im Folgenden wird beschrieben, wie die einzelnen Transformationstypen, die in den Anforderung aufgestellt wurden, ausgedrückt werden können.

4.4.1 Strukturänderungen

Filterung

Alle Sprachen unterstützen die Filterung innerhalb einer **where**-Anweisung. Dabei ist es möglich auf den Inhalt von Elementen und Attributen zu prüfen. Als Beispiel wird die Verwendung in Lorel gezeigt (s.Abb. 4.22).

```
select xml(book:{@year: y, title:t})
from bib.book b, b.title t, b.year y
where b.publisher = "Addison-Wesley" and y > 1991)})
```

Abb. 4.22: Filterung in Lorel

In Sprachen, die XPath einbetten, wie XQuery und Quilt, ist es zusätzlich die Filterung bei der Elementauswahl möglich. Dies erlaubt bei der Integration eine einfache Vorfilterung der Quelldokumente, so dass sich die Ergebnisfilterung auf den **where**-Abschnitt beschränken kann.

Weiterhin kann eine Filterung durch die Update-Language erfolgen. Hier ist es möglich, einzelne Abschnitte eines Quelldokumentes zu entfernen.

Anordnung

Bei der Generierung von Ergebniselementen ist es wichtig, verschiedene Arten der Sortierung zu unterstützen:

- Sortierung von Rückgabestrukturen
- Erhalten der Ordnung des Eingabedokuments

Das Sortieren von Rückgabestrukturen wird von allen Sprachen ausser XQL unterstützt. Die Spezifikation von XQL sieht keine Möglichkeit zur Sortierung vor, manche Implementierungen erweitern jedoch XQL um einen SORT-Operator (TAMINO).

Alle Sprachen außer XQuery, Quilt und Yatl erfordern eine explizite Anweisung, wenn die Ordnung des Eingabedokuments beibehalten werden soll. Die Ordnung und Abfolge einer Variablenbindung wird bei diesen Sprachen nicht verändert. Bei Lorel ist die Ordnung des Ergebnisses willkürlich, wenn hierzu nichts in der Anfrage angegeben wird.

```
LOREL: order by document order

XQuery:
 order by $a/...

Quilt:
 SORTBY (name DESCENDING)
```

Das Argument einer Sortierung ist in Quilt eine Knotenmenge. Hier kann aufsteigend oder absteigend sortiert werden (ASCENDING/DESCENDING). Das gleiche gilt für XQuery.

Änderung der Modellierung

Die Änderung der Modellierung ist in allen Sprachen möglich. Dabei muss das Ergebnis der Transformation explizit beschrieben werden und kann nur auf einem geeigneten Quelldokument erfolgen. Als Beispiel wird die Einbettung eines Element-Textes als Attribut und eines Attributes als Elementtext in XQuery gezeigt (s.Abb. 4.23).

```
for $b in doc()/...
where
return <ergebnis attributneu="$b/element/text()"
        $b/@attributalt
       </ergebnis>
```

Abb. 4.23: Modellierungsänderung in XQuery

Durch die Typ-Angabe bei der Element-Auswahl in XQuery (text()) kann angegeben werden, welcher Inhalt eines Elementes eingesetzt werden soll. Dies ist in z.B. XML-QL nicht möglich (s.Abb. 4.24).

```
WHERE <a>$t</a>
CONSTRUCT <b a=$t/>
```

Abb. 4.24: Modellierungsänderung in XML-QL

Ersetzungen/Anpassungen

Eine Ersetzung kann in allen Sprachen durchgeführt werden, die die Möglichkeit der Konstruktion von Elementen bieten. Hierbei wird die aufzubauende Struktur und die Einbettung der Daten des Ausgangsdokuments beschrieben. Am Beispiel von XML-QL wird die Änderung des Namens des Wurzelelements und eine Restrukturierung unter Auslassung des Elements <a> gezeigt (s.Abb. 4.25).

```
CONSTRUCT <result> {
WHERE
  <source>
    <a>
      <b>$t</b>
    </a>
  </source> IN "http://www...xml"

CONSTRUCT
   <b>$t</b>

} </result>
```

Abb. 4.25: Anpassungen in XML-QL

Integration mehrerer Dokumente

Zur Integration mehrer Dokumente muss die Möglichkeit bestehen mehr als ein Quelldokument zu adressieren und die Elemente der Dokumente zu einer neuen Struktur zusammenzusetzen.

Einfache Integration

Als Beispiel sei hier das Verfahren in XML-QL gezeigt (s.Abb. 4.26). Dokumente werden durch URIs angegeben, jedoch ohne Typ. Im Beispiel werden mehrere Muster/Dokument-Selektionen aneinandergehängt. Bei Variablen innerhalb der Mustermengen muss darauf geachtet werden, dass sie sich nicht gegenseitig überschreiben.

```
WHERE Muster IN "www.a.b.c/data.xml",
      Muster IN "www.irs.gov/taxpayers.xml"
CONSTRUCT
```

Abb. 4.26: Integration in XML-QL

Im Gegensatz zu XML-QL ist bei Lorel nicht einfach ersichtlich, auf welches Dokument sich eine Anfrage bezieht, da die Pfadausdrücke durch Aliasnamen ersetzt werden. Auch ist die Angabe von URIs nicht vorgesehen, eine Pfadangabe bezieht sich immer auf ein lokales XML-Dokument.

```
select xml(books-with-prices:{
 select xml(book-with-prices: { title: t,
                                price-amazon: pa,
                                price-bn: pb }
 from bib.book b, b.title tb, b.price pb,
   reviews.entry e, e.title ta, e.price pa
```

Abb. 4.27: Integration in Lorel

Funktionale Integration

Unter einer funktionalen Integration werden spezielle Verknüpfungsformen der Dokumente, wie Joins und Grouping verstanden.

Join

Join und Grouping sind Operationen der relationalen Algebra. Da XML als unabhängiges Datenaustauschformat eingesetzt werden soll, wurden auch hier ähnliche Operationen eingeführt. In SQL werden einige verschiedene Semantiken für Joins definiert (JOIN, LEFT OUTER JOIN, FULL OUTER JOIN). Am vollständigsten werden diese Semantiken von Quilt und XQuery erfüllt [Cha 00]. Als Quelle für einen Join können in XML ein oder mehrere Dokumente dienen.

XML-QL

```
WHERE <person>
       <name>$a
      </> IN "www.a.b.c/data.xml",
      <taxpayer>
       <name>$a</>
       <ssn>$ssn
      </> IN "www.irs.gov/taxpayers.xml"
CONSTRUCT <result> $a $ssn </>
```

Durch die zweimalige Verwendung von $a wird ein Join ausgedrückt. $a wird zunächst durch Daten des Dokuments data.xml belegt. Der Wert von $a dient nun im Dokument taxpayers.xml als Muster.

YATL

```
make
   d [ *b [ @c [ $y ] ] ]

match "http://www..." with
   a [ *b [ @c [ $y ] ] ]

match "http://www..." with
   d [ *e [ @f [ $z ] ] ]

where
   @f = @c
```

Der Join wird wie bei XML-QL durch Vorbelegung einer Muster-Variablen erzeugt.

Lorel

```
select xml(books-with-prices:{
 select xml(book-with-prices: { title: t,
                                price-amazon: pa,
                                price-bn: pb }
 from bib.book b, b.title tb, b.price pb,
   reviews.entry e, e.title ta, e.price pa

 where tb = ta)}
```

Lorel verwendet einen expliziten Vergleich der Elemente, deren Information zusammengefügt wird. Diese Vorgehensweise ist aus SQL bekannt.

XQuery

```
<result>
    FOR $i in doc("items.xml")//item_tuple,
        $u in doc("users.xml")//user_tuple

    WHERE $i/offered_by = $u/userid

    RETURN
      <interesting_item>
        $i/itemno,
        $i/description,
        <seller> $u/name/text() </seller>
      </interesting_item>
</result>
```

Grouping

Ein Grouping ist in allen Sprachen möglich. Als Beispiel sei hier das Verfahren in Quilt auf demselben Dokument gezeigt. Dabei werden durch die DISTINCT-Funktion doppelte Einträge vermieden.

```
<result>
  (
    FOR $i IN DISTINCT document("bids.xml'')//itemno
    LET $b := document("bids.xml")//bid_tuple[itemno = $i]
    RETURN
      <highbid>
       $i,
       <bid> max($b/bid_amount) </bid>
      </highbid> SORTBY(itemno)
  )
</result>
```

4.4.2 Datenänderungen

Zusammenführung/Aufteilung

XQuery

Die XQuery-Spezifikation enthält umfangreiche Methoden zur Bearbeitung von Zeichenketten. In Abb. 4.28 wird ein Ausschnitt der Funktionen gezeigt, der sich mit den aufgestellten Anforderungen deckt [W3C 03/2]. Es werden noch weitaus mehr Funktionen bereitgestellt.

fn:concat	Konkatenation
fn:substring	Rückgabe eines Teil-Strings
fn:string-length	Rückgabe der Länge eines Strings
fn:upper-case	
fn:lower-case	
fn:contains	Test auf Teilstring
fn:starts-with	Test auf Teilstring
fn:ends-with	Test auf Teilstring
fn:substring-before	Rückgabe eines Teil-Strings
fn:substring-after	Rückgabe eines Teil-Strings

Abb. 4.28: Zeichenkettenfunktionen in XQuery

Von den anderen Sprachen ist nicht bekannt, ob sie solche Funktionen anbieten. Lediglich YATL bietet einen concat-Operator, der Zeichenketten verknüpft.

Aggregationsfunktionen

	Lorel	XQuery	XQL	XML-QL	Quilt	YATL
Minimum	X	X	-	X	X	-
Maximum	X	X	-	X	X	-
Summe	X	X	-	X	X	-
Anzahl	X	X	X	X	X	-
Durchschnitt	X	X	-	X	X	-

Abb. 4.29: Aggregationsfunktionen

Aggregationsfunktionen erlauben es durch die Generierung von Information, Daten zu Transformieren oder Entscheidungen bei einer Transformation zu unterstützen (s.Abb. 4.30). Über Aggregationsfunktionen in YATL ist nichts bekannt. In XML-QL sind Aggregationen erst in der letzten Spezifikation enthalten.

Am Beispiel von XQuery

```
<a>
{sum(for $x in (3,4,5) return $x)}

</a>
```

Abb. 4.30: Anpassungen in XML-QL

Typkonvertierung

Man unterscheidet implizite und explizite Typumwandlungen von Werten. Dabei werden nur atomare Typen des jeweiligen Datenmodells einer Anfragesprache betrachtet.

Komplexe und selbstdefinierte Typen werden im Abschnitt "Komplexe Umrechnungen" behandelt. Implizite Typumwandlungen bezeichnen die automatische Umwandlung z.B. von Parametern einer Funktion. Das heißt, wenn eine Konvertierung nötig und möglich ist, wird sie durchgeführt. Implizite Typumwandlungen sind nur in Lorel und XQuery möglich (siehe Typerzwingung 4.2.4).

Von expliziten Typumwandlungen spricht man bei der Transformation von Eingabe- in Ausgabedaten. Hier fordert der Benutzer eine Konvertierung an, die zur Weiterverarbeitung der Ausgabedaten beiträgt. Die ist z.Bsp. der Fall, wenn die Zieldatenbank in einem Datenfeld einen speziellen Typ erwartet, der in der Ausgangsdatenbank durch einen anderen Typ dargestellt wird.

Explizite Umwandlungen werden nur von XQuery (mittels "cast as") als Sprachelemente unterstützt. Andere Sprachen können explizite Typumwandlungen mit Hilfe von Funktionen nachbilden (s.Abb. 4.31).

```
xs:boolean
xs:byte
xs:date
xs:dateTime
xs:decimal
xs:double
xs:float
xs:int
xs:integer
xs:long
xs:short
xs:string
xs:time
```

Abb. 4.31: cast as-Typen in XQuery

Komplexe Umrechnungen

Eine komplexe Umrechnung erfordert die Zerlegung eines Textfeldes in Wert und Einheit. Umrechnungen sind mit allen Sprachen möglich, die über Funktionen und Bedingungen/Alternativen verfügen. Weiterhin sind umfangreiche Datenoperationen auf unstrukturierten Daten nötig. XML-QL verfügt über Funktionen, aber nicht über Bedingungen und Datenoperationen. Auch in Lorel sind Umrechnungen nicht möglich, da keine Funktionen definiert werden können.

Benutzerdefinierte Funktionen

Benutzerdefinierte Funktionen dienen der Vereinfachung komplexer Anfragen. Lorel und YATL unterstützen keine solchen Funktionen.

Die Deklaration von Funktionen in einigen Sprachen ist hier dargestellt:

```
XML-QL:

  FUNCTION XML-QL-Query

Quilt:

  FUNCTION name (parameters) RETURNS output

XQuery:

declare function local:summary($emps as element(employee)*)
   as element(dept)*
{
   for $d in fn:distinct-values($emps/deptno)
   let $e := $emps[deptno = $d]
   return
      <dept>
         <deptno>{$d}</deptno>
         <headcount> {fn:count($e)} </headcount>
         <payroll> {fn:sum($e/salary)} </payroll>
      </dept>
};

local:summary(fn:doc("acme_corp.xml")//employee[location = "Denver"])
```

Abb. 4.32: Benutzerdefinierte Funktionen

4.5 Weitere Methoden

4.5.1 Erweiterbarkeit: Externe Funktionen

Externe Funktionen stehen außerhalb der Sprachdefinition und sind daher Implementationsabhängig. Sie werden meist über Namespaces eingebunden. Eine generelle Untersuchung kann daher erst nach Auswahl einer Sprache stattfinden. Externe Funktionen werden z.B. durch die Implementationen von XQuery names Saxon und Qexo unterstützt.

4.5.2 Erweiterbarkeit: Namensräume

Durch die Vorbelegung von Zeichenfolgen durch bestimmte Schlüsselworte der Anfragesprachen wird die freie Verwendung von Begriffen oder gleichlautenden Bezeichnern in im gleichen Kontext eingeschränkt. Eine Abhilfe können hier Namespaces sein.

Ein Namespace muss vor der Verwendung deklariert werden.

```
declare namespace foo = "http://example.org";

<foo:bar> Lentils </foo:bar>

<namespace:name>
```

Abb. 4.33: Namespacedeklaration und Anwendung in Quilt

Namespaces werden von Lorel und XML-QL nicht unterstützt.

4.6 Eignungsbewertung

Es wurden die Methoden verschiedener XML-Anfragesprachen zum Einsatz bei einer Transformation untersucht. Abschliessend soll nun festgehalten werden, welche Sprache am geeignetsten zum Einsatz im Propagationsmanager ist. Die Sprachen sind aufgrund ihres Datenmodells und Einsatzzwecke unterschiedlich gut zur Transformation geeignet.

	Analyse	Transfer	Konstruktion	Erweiterbarkeit
XML-QL	-	+	+	+/-
YATL	+/-	-	-	-
Lorel	+/-	+	-	-
Quilt	+	+/-	+	+
XQuery	+	+	+	+

Abb. 4.34 : Eignung in Teilbereichen

Die Bewertung der Transformationsfähigkeiten der jeweiligen Sprachen ist in Abb. 4.34 dargestellt. Allen verbliebenen und untersuchten Sprachen gemeinsam ist die Auftrennung in einen Analyse und Konstruktionsteil. Die Sortierung von Ergebnissen wird von allen Sprachen ermöglicht. Die Navigationsmöglichkeiten sind bei YATL nicht in demselben Umfang gegeben, wie in den anderen Sprachen. Durch die Einbindung der unabhängigen Sprache XPath bei Quilt und XQuery entsteht die größtmögliche Freiheit in der Navigation. Auch das von Quilt und XQuery benutzte Datenmodell bietet umfangreiche Möglichkeiten der Navigation. Durch das von Lorel verwendete Datenmodell sind keine Anfragen hinsichtlich der Anordnung von Elementen möglich.

YATL, Quilt und XML-QL fehlt die explizite und implizite Typumwandlung. Lorel unterstützt keine benutzerdefinierten Funktionen. Tag-Variablen werden von XML-QL, YATL, Lorel, Quilt und XQuery unterstützt und eignen sich zur Analyse unbekannter Strukturen. Auch hinsichtlich der Konstruktion ist Lorel nicht zur Transformation geeignet (fehlende Attribute). Die Erweiterbarkeit der Sprachen wird durch externe und interne benutzerdefinierte Funktionen erreicht. Interne Funktionen sind in Lorel und YATL nicht vorgesehen.

	XML-QL	YATL	Lorel	Quilt	XQuery
Filterung	+/-	+/-	+/-	+	+
Anordnung	+/-	+	-	+	+
Modellierung	+/-	-	-	+	+
Funktionen	+	+	+	+	+
Anpassungen	+	+/-	+	+	+
Integration	+/-	+	-	+	+

Abb. 4.35 : Eignung zur Strukturänderung

	XML-QL	YATL	Lorel	Quilt	XQuery
Datenoperationen	-	+/-	-	-	+
Aggregation	+/-	-	+	+	+
Umrechnungen	-	+/-	-	-	+
Typkonvertierung	-	-	+	+/-	+
Funktionen	+	+	-	+	+
Integration	+	+/-	+/-	+	+

Abb. 4.36 : Eignung zur Datenänderung

In Abb. 4.35 und Abb. 4.36 ist die Eignung der Sprachen auf den Gebieten Struktur- und Datenänderung dargestellt.

Die Filterung wird am umfangreichsten durch diejenigen Sprachen ermöglicht, die XPath einbetten. Hier wird die XML-Struktur vollständig unterstützt, was auch XML-Attribute beinhaltet. Das Datenmodell von Lorel ist ungeeignet, um Fragen zur Ordnung von Dokumenten zu beantworten. Operationen auf unstrukturierten Daten werden nur von XQuery vollständig unterstützt. Lorel und YATL sind zur Änderung der Modellierung ungeeignet, da hier keine XML-Attribute unterstützt werden. Über der Aggregatfunktionen ist in YATL nichts bekannt. Implementation von XML-QL bieten Aggregationsfunktionen nicht an. Lorel, Quilt und XQuery definieren umfangreiche Datentypen, in die Werte explizit umgewandelt werden können. Lorel und XQuery unterstützen zusätzlich die automatische Typerzwingung. Die Integration von verteilten XML-Dokumenten wird in Lorel dadurch erschwert, dass keine URIs verwendet werden können. Die umfangreichsten Semantiken von Joins können in Quilt und XQuery ausgedrückt werden. Groupings sind in allen Sprachen möglich.

XQuery erfüllt die meisten der gestellten Anforderungen und wird ihm Rahmen der Implementation in das SIES integriert werden.

4.7 Transformationen mit XQuery

Abschliessend sollen einige vollständige Transformationsbeispiele in XQuery angegeben werden, um die Anwendbarkeit zu demonstrieren. Ein Abriss der XQuery-Syntax ist in Anhang A beigefügt. Die Anfragen orientieren sich an den entsprechenden in den Anforderungen gezeigten Beispielen und Strukturen.

4.7.1 Löschen von Information

Das Löschen von Information wird in XQuery durch eine Neukonstruktion der Eingabestruktur unter Auslassung zu löschender Information erreicht (s.Abb. 4.37).

```
for $a in doc("xml")/.../prod_schritt

return <prod_schritt>
        $a/a
        $a/c
       </prod_schritt>
```

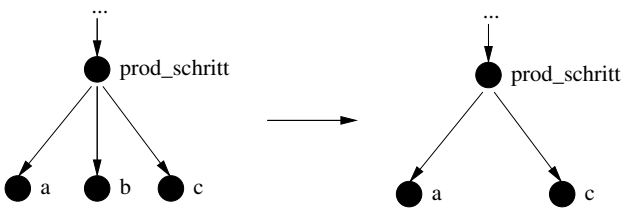


Abb. 4.37 : Löschen von Information

4.7.2 Modellierungsänderung

Eine Änderung der Modellierung geschieht durch einsetzen des Wertes der ursprünglichen Struktur in einen passenden Konstruktor (Element/Attribut) (s.Abb. 4.38).

```
for $a in doc("xml")/.../prod_schritt

return <prod_schritt transport="{$a/transport/text()}">

       </prod_schritt>
```

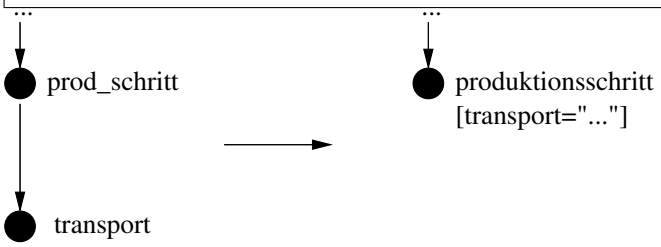


Abb. 4.38 : Modellierungsänderung

5 Implementation

Die Implementation beinhaltete zunächst die Analyse der bereits bestehenden Strukturen, um den notwendigen Umfang der Anpassungen zu ermitteln. Dabei wurde untersucht, wie der XSLT-Prozessor in die Struktur des SIES eingebunden ist, wie die Datenhaltung realisiert ist und wie die Kommunikation mit den Komponenten bewerkstelligt wird.

Anschliessend wurde überlegt, wie ein XQuery-Prozessor parallel zum XSLT-Prozessor integriert werden könnte, so dass weiterhin XSLT-Transformationen durchgeführt werden können. Dies hat den Hintergrund, dass in großem Umfang entsprechende XQuery-Anfragen mit der Funktionalität bisheriger XSLT-Skripte erst erzeugt werden müssen.

Die Implementation umfasste schließlich die Auswahl eines geeigneten XQuery-Prozessors und dessen Integration, welche im Folgenden beschrieben wird.

5.1 XQuery-Prozessoren: Qexo und Saxon

In die engere Auswahl kamen die plattformunabhängigen XQuery-Prozessoren Qexo und Saxon, Information über weitere verfügbare Prozessoren ist Kap.3 zu entnehmen. Die Bezugsquellen sind Abb. 5.1 angegeben.

Name	Quelle
Qexo	http://www.gnu.org/software/qexo/
Saxon	http://saxon.sourceforge.net/

Abb. 5.1 : Bezugsquellen der XQuery-Prozessoren

Zu Berücksichtigen waren die Vollständigkeit der Umsetzung der XQuery Spezifikation und die Ingtegrierbarkeit gemäß der aufgestellten Anforderungen.

Das Ergebnis des Vergleichs der beiden Prozessoren ist in Abb. 5.2 dargestellt.

	Qexo	Saxon
Verfügbarkeit	+	+
XQuery Kompatibilität	+/-	+
Erweiterbarkeit	+	+
Konfiguration	-	+
Anforderungskonformität	-	+

Abb. 5.2 : Eigenschaften der XQuery-Prozessoren

Beide Prozessoren sind im Quelltext und als Java-Bytecode verfügbar und können

daher direkt eingebunden werden. Beide werden ständig gewartet, so dass auch erwartet werden kann, dass Fehler schnell behoben werden. Da die Entwicklung der XQuery-Sprache in einem frühen Stadium ist und an der genauen Spezifikation noch gearbeitet wird, unterstützen viele der frei verfügbaren XQuery Interpreter die Vorgaben des XQuery-1.0 Standards nur zu einem Teil. Die XQuery-Kompatibilität ist bei Saxon höher als bei Qexo. Die Defizite von Qexo in Bezug auf die XQuery Spezifikation sind auf der zugehörigen Webseite dargestellt. Am schwerwiegendsten für den Einsatz im SIES ist das Fehlen der Sortierfunktion. Im Gegensatz zu anderen XQuery-Prozessoren unterstützen beide hier betrachteten jedoch die Einbindung von Java-Methoden in eine Anfrage. Mit dieser Methode lassen sich auf einfache Weise die fehlenden Funktionen nachrüsten. Durch die Möglichkeit auch Java-Methoden innerhalb der Anfragen zu integrieren können fehlende (z.B. Sortierfunktion bei Qexo) oder neu benötigte leicht nachgerüstet werden. Die dadurch entstehenden Anfragen entsprechen jedoch nicht mehr zum XQuery-1.0 Standard. Die Erweiterbarkeit und nützliche Funktionen sollen bei beiden Prozessoren hier kurz beschrieben werden.

Die Wahl des XQuery-Prozessors schließslich aufgrund der umfangreichen Konfigurationsmöglichkeiten auf das Opensource Projekt Saxon. Saxon ist unter der Mozilla-Lizenz frei verfügbar. Saxon kann auch als Servlet ausgeführt werden und so auf einer physikalisch abgetrennten Maschine laufen.

5.1.1 Erweiterung der Funktionen von Qexo

Qexo kann auf einfache Art mit JAVA-Funktionen erweitert werden. Die Funktionen müssen statisch definiert werden und passende Rückgabetypen besitzen. Dies ist in Abb. 5.3 dargestellt.

```
package sies.utils;
public class StringUtil {
        public static int indexOf(String src, String pat) {
                return src.indexOf(pat);
        }
        public static String substr(String src, int from, int to) {
                return src.substring(from,to);
        }
}
declare namespace SIES  = "class:sies.utils.StringUtil"
 for $b in doc("contains.xml")/test
  return
    <result>
      { $b/text() }
      {
        if ( SIES:indexOf("a b c","b")) then yes else no      }
    </result>
```

Abb. 5.3 : Erweiterbarkeit von Qexo

Das Beispiel zeigt, wie eine Funktion zur Extraktion eines Teilstrings integriert werden kann.

5.1.2 Erweiterung der Funktion von Saxon

Die Methodik zur Erweiterung von Saxon entspricht der von Qexo, mit dem Unterschied, dass hier nicht nur statisch definierte Funktionen sondern auch Methoden von Objekten ausgeführt werden können. Im Gegensatz zu Qexo werden externe Funktionen nicht innerhalb einer Anfrage definiert, sondern kompiliert im Zugriffspfad von Saxon abgelegt und mittels Namespace-Deklaration eingebunden (s.Abb. 5.4).

```
Namespace-Deklaration:

  xmlns:function="java:java.util.Function"

Verwendung einer Funktion:

  function:localname()
```

Abb. 5.4 : Erweiterbarkeit von Saxon

5.1.3 Standalone Aufruf Saxon und Qexo

Um Transformationsanfragen zu entwickeln und zu testen können die Prozessoren auch außerhalb des Propagationsmanagers aufgerufen werden:

```
Qexo:

  java -cp kawa-1.7.jar kawa.repl --xquery Anfrage.xql

Saxon:

  java net.sf.saxon.Query query Anfrage.xql
```

Abb. 5.5 : Standalone Aufruf

5.1.4 Konfiguration von Saxon

Saxon erlaubt eine umfangreiche Konfiguration und die Einbindung eines URI-Resolvers, wie in den Anforderungen enthalten (s.Abb. 5.6).

```
Standardkonfiguration:

  Configuration config = new Configuration();
  config.setHostLanguage( config.XQUERY);
  StaticQueryContext staticQueryContext = new StaticQueryContext(config);
  QueryProcessor qp = new QueryProcessor(staticQueryContext);
```

Abb. 5.6 : Konfiguration von Saxon

Beispiele aus einer Vielzahl möglicher Konfigurationsmethoden:

```
setURIResolver(javax.xml.transform.URIResolver resolver)
 Set the URIResolver to be used in this configuration.

setValidation(boolean validation)
 Determine whether the XML parser for source documents will
 be asked to perform DTD validation of source documents

setBaseURI(java.lang.String baseURI)
 Set the Base URI of the query

declareFunction(XQueryFunction function)
 Register a user-defined XQuery function

declareVariable(VariableDeclaration var)
 Declare a global variable.
```

5.2 Integration in SIES

Zur Integration in das SIES musste der Abhängigkeitsmanager und der Propagationsmanager angepasst werden. Der Abhängigkeitsmanager dient zur Definition von Datenflüssen und Transformationen. Der Propagationsmanager führt die definierten Transformationen beim Eintreffen relevanter Daten aus. Der Datenfluss des Propagationsmanagers ist in Abb. 5.7 dargestellt.

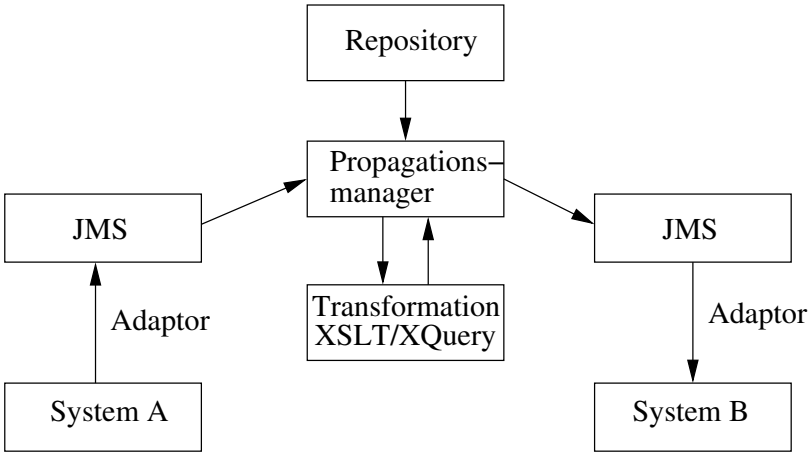


Abb. 5.7 : Datenfluss

5.2.1 Anpassung des Abhängigkeitsmanagers

Im Abhängigkeitsmanager war die Möglichkeit vorzusehen, XQuery-Anfragen zu bearbeiten und speichern zu können. Dazu musste die Sprache XRL+ angepasst werden, um eine Unterscheidung nach XSLT-Skripten und Anfragen ausdrücken zu können.

5.2.2 Erweiterung von XRL+

Bei einer Transformationsanweisung muss festgehalten werden, ob ein XSLT- oder XQuery-Prozessor zum Einsatz kommt. Hierzu wird die TRANSFORM-Anweisung um ein "type"-Attribut erweitert (s.Abb. 5.8), dessen Inhalt den zu verwendenden Prozessor spezifiziert. Der Transformationsprozessor führt schließlich das Attribut "script" übergebene Skript aus.

```
<xrl:TRANSFORM type="..." script="..." xml_in="..." xml_out="..."/>
```

Abb. 5.8 : Transformations-Anweisung

Die Attribute "xml_in" und "xml_out" stellen wie in der ursprünglichen Version die Verbindung zwischen ein- und ausgehenden Message-Queues her. Zur Validierung musste das XRL+-Schema angepasst werden (s.Abb. thechapter.9).

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://informatik.uni-stuttgart.de/XRL"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:xrl="http://informatik.uni-stuttgart.de/XRL"
 elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:complexType name="routing_element">
      <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element ref="xrl:PROPAGATE"/>
         <xs:element ref="xrl:TRANSFORM"/>
         <xs:element ref="xrl:WAIT"/>
         <xs:element ref="xrl:SEQUENCE"/>
         <xs:element ref="xrl:PARALLEL"/>
         <xs:element ref="xrl:CONDITION"/>
         <xs:element ref="xrl:FILTER"/>
         <xs:element ref="xrl:WHILE_DO"/>
         <xs:element ref="xrl:TERMINATE"/>
      </xs:choice>
        </xs:complexType>
[...]

<xs:element name="TRANSFORM">
   <xs:complexType>
      <xs:attribute name="xml_in" type="xs:IDREF" use="required"/>
      <xs:attribute name="type" type="xs:string" use="required"/>
      <xs:attribute name="script" type="xs:string" use="required"/>
      <xs:attribute name="xml_out" type="xs:ID" use="required"/>
   </xs:complexType>
</xs:element>

[...]
```

Abb. 5.9 : Anpassung von XRL+

5.2.3 Datenhaltung

Die Datenhaltung des Repositories musste hinsichtlich der Speicherung und Unterscheidung von von Anfragen und Transformationsskripten angepasst werden. Die alte Struktur ist in Abb. 5.10, die neue Struktur in Abb. 5.11 gezeigt.

XrlXsltID	XrlID	XsltID
1	0	0
2	1	1
3	1	3
5	8	5
21	10	14

Abb. 5.10 : XRL zu XSLT-Zuordnungstabelle

XrlXsltID	XrlID	ScriptD	type
1	0	0	xslt
2	1	1	xslt
3	1	3	xslt
5	8	5	xquery
21	10	14	xquery

Abb. 5.11 : XRL zu XSLT und XQuery-Zuordnungstabelle

Zur Anfragespeicherung wurde eine parallele Tabelle zu den XSLT-Skripten angelegt (s.Abb. 5.12).

ScriptID	ScriptName	Script
0	Champagne2Chamapagne	<?xml ...>
1	emp_staff2emp	<?xml ...>
2	OrderManagement	<?xml ...>
3	po2tm	<?xml ...>

Abb. 5.12 : Skriptspeicherung

5.2.4 Anpassung des Propagationsmanagers

Es wurde parallel zum XSLT-Prozessor eine Klasse XQueryProcessor erstellt, die in der gleichen Form wie der XSLT-Prozessor benutzt werden kann.

Die Parameter der Transformationfunktion entsprechen denen des XSLT-Prozessors, wobei die zu verarbeitenden Datenströme übergeben werden können.

```
public void transform(String subjectId, InputStream xqueryStream,
Hashtable params, String targetId)
```

Abb. 5.12 : Skriptspeicherung

Die Parameter **subjectId** und **targetId** verbinden die XML-Ströme, der Datenstrom **xqueryStream** liefert die zugehörige Transformations-Anfrage. Variablendeklarationen können über den Parameter **params** übergeben werden.

Auflösen von URIs

Die Integration eines URI-Resolvers erfolgt durch den Konfigurationsmechanismus von Saxon.

Die Message-Queues

Die Kommunikation zwischen Propopagationsmanager und angekoppelten Systemen erfolgt über sog. Message-Queues. Message-Queues stellen einen zuverlässigen und ausfallsicheren Mechanismus dar, um Nachrichten auszutauschen. Dabei ist es Aufgabe des Empfängers, verfügbare Nachrichten abzuholen. Im SIES wurden die Java Message Services (JMS) verwendet, da sie sowohl die Möglichkeit bieten, Nachrichten persistent zu speichern, als auch verschiedene Kommunikationssemantiken wie Point-to-Point und Multicast unterstützen.

Die Anbindung an die Message-Queues wurde wie bisher bekannt auch für den Anfrageprozessor umgesetzt.

5.3 Realisierung von SIES-Transformationen in XQuery

Zur Demonstration der Funktion wurden einige XSLT-Skripte in XQuery nachgebildet.

5.3.1 Umrechnung

Die Umrechnung transformiert bekannte Abkürzungen in Vollformen.

```
define function name ($x) {
  if ($x = "U. Heinkel") then "Uwe Heinkel"
  else if ($x = "S. Opletal") then "Sascha Opletal"
  else $x
 }

<result>
{
for $a in doc("namen.xml")/namen/name
return <nameneu> {name($a/text())} </nameneu>

}
</result>
```

Abb. 5.1: Umrechnung von Abkürzungen

5.3.2 Konvertierung und Anreicherung

Dieses Beispiel zeigt eine Umsetzung des XSLT-Skripts aus Kapitel 3, Abschnitt 3.1.3.

```
<Materialstrommatrix>
{
let $max := doc("input.xml")/Fertigungsauftrag/Fertigungsschritte...
.../Arbeitsgang[position()=last()]/@Nr

for $nr in 0 to $max
let $teilenr := doc("input.xml")/Fertigungsauftrag/Teil/@Teilenr
let $menge   := doc("input.xml")/Fertigungsauftrag/Teil/@Menge
let $gang    := doc("input.xml")/Fertigungsauftrag/Fertigungsschritte...
.../Arbeitsgang[position()=$nr]

let $start := $gang/@Maschine
let $ziel  := doc("input.xml")/Fertigungsauftrag/Fertigungsschritte...
.../Arbeitsgang[position()=$nr + 1]/@Maschine

return  <Transport
               Start="{if ($nr = 0) then "Lager" else $start}"
                Ziel="{if ($nr = $max) then "Fertigteillager" else $ziel}"
                Menge="{$menge}" Teilenr="{$teilenr}" Nr="{$nr}">
       </Transport>

}
</Materialstrommatrix>
```

5.3.3 Aggregation

Die Aggregation vereinigt Anzahlen gleicher Teile mit gleichem Transportweg.

```
(: Transformation von AMP-Daten in XQuery :)

(: Funktion fuer distinct-values :)
declare function distinct-values($vals)
{
  for $x at $i in $vals
  where empty(for $y at $j in $vals
              where $j < $i and $x eq $y return $y)
  return $x
}

<materialstrom>
{
 for $trans in distinct-values(doc("data.xml")/auftr_abw/...
  ...prod_schritt/transport/@start_pe)

 let $gleiche :=  for $trans2 in doc("data.xml")/auftr_abw/
  ...prod_schritt/transport
                 where $trans = $trans2/@start_pe
                 return ($trans2/teil)
  return
   <transport_weg>
   { $trans }

   { for $teile in distinct-values(doc("data.xml")/auftr_abw/...
  ...prod_schritt/transport/@start_pe)
     where $teile = $trans
     return

       for $i in distinct-values($gleiche/@nr)
       return <teil nr="{ $i }" anzahl="{ sum(for $zahl in $gleiche
                  where $i = $zahl/@nr
                  return number($zahl/@anzahl))
                          }"/>
}
        </transport_weg>
}
</materialstrom>
```

5.3.4 Integration

Die Integration verbindet mehrere Ausgangsdatensätze zu einem neuen.

```
<teile>
{
for $a in doc("teilbez.xml")/teile/teil,
    $b in doc("teilpreis.xml")/teile/teil

let $dm := number($b/preis_dm/text())
where $a/@nr = $b/@nr
return <teil nr="{$a/@nr}" bezeichnung="{$a/bezeichnung/text()}">
       <preis>
       <dm> { $b/preis_dm/text() } </dm>
       <eur> {round(($dm * 100) div 1.95583) div 100 } </eur>
       </preis>
       </teil>
}
</teile>
```

5.3.5 Zyklisches Inkrementieren

In einer zyklischen Transformation wird in einem Zähler die Anzahl der Zyklen festgehalten. Der Zähler wird bei der Transformation als Attributwert ausgelesen, inkrementiert und an wieder als Attribut gespeichert.

```
for $a in doc("start.xml")//elem

return <elem cycle="{ number($a/@cycle)+1 }">
       {""}
       </elem>
```

6 Zusammenfassung und Ausblick

Es wurden die verfügbaren XML-Anfragesprachen untersucht und herausgearbeitet, ob und wie XML-Transformationen mit diesen durchführbar sind.

Dabei wurde festgestellt, dass XQuery, der neue Anfragesprachenstandard des W3C für XML-Dokumente, die meisten der gestellten Anforderungen erfüllt, da es die Eigenschaften vieler XML-Anfragesprachen in sich vereint. Aufgrund der Standardisierungs-Bemühungen des W3C stehen ausreichend Implementationen zur Verfügung, so dass eine geeignete ausgewählt und in den Propagationsmanager integriert werden konnte. Die Realisierbarkeit wurde durch Angabe äquivalenter XQuery Anfragen gezeigt.

Gegenwärtig stehen die Spezifikationen von XQuery 1.0 und XPath 2.0 kurz vor der Verabschiedung. Mit der Verabschiedung als Empfehlung (Recommendation Status) steht zu erwarten, dass die gegenwärtigen Mängel und Unvollständigkeiten der Implementationen schnell behoben werden.

Es gibt einige spezialisierte Implementationen, die Abwandlungen der Spezifikation beinhalten. So liegt der Fokus bei manchen Implementationen auf der Verarbeitung von Zeichenketten, wobei die Funktionalität stark über den XQuery-Standard hinausgeht. Manche anderen Implementationen benutzen eine andere Semantik einzelner Funktionen, wie etwa bei der Modifikation von Variablenbelegung, die nach XQuery-Standard nicht verändert werden können.

An einigen Stellen wird schon über Erweiterungen von XQuery nachgedacht. So wird vorgeschlagen, eine Sprache nach Art der Update-Languages zu integrieren [Bru 03]. Dies würde es ermöglichen, bei kleinen Änderungen der Ausgangsstruktur durch Modifikationsoperatoren eine einfachere Formulierung anzugeben, um die explizite Neukonstruktion eines Dokuments zu vermeiden.

Anhang A: Syntax von XQuery

Mit der Auswahl von XQuery als im SIES zu integrierende Sprache, wird hier ein kurzer Überblick über die wichtigsten syntaktischen Elemente gegeben. XQuery bettet zur Pfadbestimmung die Sprache XPath ein, auf die hier nicht eingegangen wird (siehe [W3C 99] [W3C 03/3]). Die Syntax ist in erweiterter Backus-Naur Form (EBNF) angegeben [14977].

Ausdrücke

XQuery stellt eine umfangreiche Sprache zur Verfügung, um XML-Dokumente abzufragen. Der Einstiegspunkt beim Aufbau einer Anfrage ist der Ausdruck.

```
Expr = ExprSingle {"," ExprSingle};
ExprSingle = (FLWORExpr | QuantifiedExpr | TypeswitchExpr
             | IfExpr | OrExpr);
PrimaryExpr =  ( Literal | VarRef | ParenthesizedExpr | ContextItemExpr
               | FunctionCall | Constructor );
```

Ausdrücke

Im Allgemeinen wird hierzu der FLWOR-Ausdruck verwendet. Mit dem FLWOR-Ausdruck kann man Elemente einer Datensammlung auswählen und bearbeiten. In Syntax und Semantik ähnelt der FLWOR-Ausdruck den select-Funktionen von SQL und OQL.

FLWOR-Ausdrücke

Zu Iterationsläufen mit gebundenen Variablen bietet XQuery sog. FLWOR Ausdrücke.

```
for $a in doc("document.xml")/elementA
where count($a/subelement) >= 10 and $a/@attributA="something"
order by avg($a/@attributB) descending
return $a
```

FLWOR-Beispiel

Mit der “where”-Anweisung werden die Daten anhand übergebener Kriterien eingeschränkt. Hierbei werden bool’sche Operationen wie “and” und “or” unterstützt. Vor der Ausgabe kann eine Sortierung vorgenommen werden.

```
FLWORExpr = {ForClause | LetClause}- [WhereClause] [OrderByClause]
            "return" ExprSingle ;
ForClause = "for" "$" VarName [TypeDeclaration] [PositionalVar]
            "in" ExprSingle {"," "$" VarName [TypeDeclaration]
            [PositionalVar] "in" ExprSingle};
LetClause = "let" "$" VarName [TypeDeclaration] ":=" ExprSingle
            {"," "$" VarName [TypeDeclaration] ":=" ExprSingle};
TypeDeclaration = "as" SequenceType ;
PositionalVar   = "at" "$" VarName ;
WhereClause     = "where" Expr ;
OrderByClause   = ("order" "by" | "stable" "order" "by") OrderSpecList ;
OrderSpecList   = OrderSpec {"," OrderSpec} ;
OrderSpec       = ExprSingle OrderModifier ;
OrderModifier   = ["ascending" | "descending"] ["empty" "greatest" |
                  "empty" "least"] ["collation" StringLiteral];
```

FLWOR-Ausdrücke

Literale

In XQuery gibt es numerische- und Stringliterale.

```
Literal         = (NumericLiteral | StringLiteral);
NumericLiteral  = (IntegerLiteral | DecimalLiteral | DoubleLiteral);
IntegerLiteral  = Digits ;
DecimalLiteral  = ("." Digits | Digits "." {0-9});
DoubleLiteral   = (("." Digits) | (Digits ("." [0-9]*)?))
                  ("e" | "E") ("+" | "-")? Digits ;
StringLiteral   = ('"' (PredefinedEntityRef | CharRef
                  | ('"' '"') | [^"&])* '"')
                  | ("'" (PredefinedEntityRef | CharRef
                  | ("'" "'") | [^'&])* "'") ;
PredefinedEntityRef = "&" ("lt" | "gt" | "amp" | "quot"
                      | "apos") ";" ;
CharRef             = "&#" (Digits | ("x" HexDigits)) ";" ;
```

Literale

Konstruktoren

Durch Konstruktoren wird die Rückgabestruktur eines FLWOR-Ausdrucks aufgebaut.

```
Constructor = DirElemConstructor
              | ComputedConstructor
              | XmlComment
              | XmlPI
              | CdataSection;
DirElemConstructor = "<" QName AttributeList ("/>" |
                     (">" ElementContent* "</" QName [S] ">"));
ElementContent = ElementContentChar
                 | "{{"
                 | "}}"
                 | DirElemConstructor
                 | EnclosedExpr
                 | CdataSection
                 | CharRef
                 | PredefinedEntityRef
                 | XmlComment
                 | XmlPI;
ElementContentChar = Char - [{}<&];
AttributeList = {S [QName [S] "=" [S] AttributeValue]};
AttributeValue = ('"' {EscapeQuot | QuotAttrValueContent} '"')
                 | ("'" {EscapeApos | AposAttrValueContent} "'")
QuotAttrValueContent = QuotAttContentChar
                       | CharRef
                       | "{{"
                       | "}}"
                       | EnclosedExpr
                       | PredefinedEntityRef;
AposAttrValueContent = AposAttContentChar
                       | CharRef
                       | "{{"
                       | "}}"
                       | EnclosedExpr
                       | PredefinedEntityRef;
QuotAttContentChar = Char - ["{}<&];
AposAttContentChar = Char - ['{}<&];
EscapeQuot         = '"' '"';
EscapeApos         = "''";
EnclosedExpr       = "{" Expr "}";
```

Konstruktoren

Variablen und Bezeichner

Eine Variable besteht aus einem “$” gefolgt von einem qualifizierten Namen (QName). Ein qualifizierter Name beinhaltet den Namespace, in dem er verwendet wird.

```
VarRef     ="$" VarName;
VarName    = QName;
QName      = [Prefix ':'] LocalPart ;
Prefix     = NCName ;
LocalPart  = NCName ;
NCName     = (Letter | '_'){NCNameChar} ;
NCNameChar = (Letter | Digit | '.' | '-' | '_');
```

Variablendefinition

Vergleichsoperationen

XQuery stellt vier Arten von Vergleichsoperationen zur Verfügung:

```
ComparisonExpr = RangeExpr [ (ValueComp | GeneralComp |
                             NodeComp | OrderComp) RangeExpr ] ;
ValueComp      = "eq" | "ne" | "lt" | "le" | "gt" | "ge" ;
GeneralComp    = "=" | "!=" | "<" | "<=" | ">" | ">=" ;
NodeComp       = "is" | "isnot" ;
OrderComp      = "<<" | ">>" ;
RangeExpr      = AdditiveExpr [ "to" AdditiveExpr  ] ;
```

Vergleichsausdrücke

Die Operationen von “ValueComp” führen Vergleiche von einzelnen Werten durch. Der allgemeine Vergleich “GeneralComp” wird bei Sequenzen von Werten benutzt. Mit “NodeComp” kann ein Vergleich auf denselben Knoten durchgeführt werden. “OrderComp” verleicht die Anordnung von Knoten in einem Dokument.

Logische Ausdrücke und Quantoren

Ein logischer Ausdruck besteht aus einer OR- oder AND-Verknüpfung zweier Ausdrücke. Der logische Ausdruck wird zu den Bool’schen Werten “false” oder “true” ausgewertet. Weitere Ausdrücke können durch Existenz- und Allquantoren beschrieben werden.

```
OrExpr  = AndExpr { "or" AndExpr };
AndExpr = InstanceofExpr { "and" InstanceofExpr };
QuantifiedExpr = (("some" "$") | ("every" "$")) VarName TypeDeclaration?
                 "in" ExprSingle ("," "$" VarName TypeDeclaration? "in"
                 ExprSingle)* "satisfies" ExprSingle;
```

Logische Ausdrücke und Quantoren

Arithmetische Ausdrücke

Es werden die Grundrechenarten und Modulo-Rechnung unterstützt.

```
AdditiveExpr       = MultiplicativeExpr {("+" | "-") MultiplicativeExpr};
MultiplicativeExpr = UnaryExpr { ("*" | "div" | "idiv" | "mod") UnaryExpr};
UnaryExpr          = {"-" | "+"} UnionExpr ;
```

Arithmetische Ausdrücke

Klammern haben Vorrang vor unären Operatoren, unäre haben Vorrang vor binären.

Pfadausdrücke

Pfadausdrücke dienen zur Beschreibung der Position eines Knotens in einem Baum. Ein Schritt im Baum vom Vater zum Tochter-Knoten wird durch den Ausdruck "/" beschrieben (z.B. vater/tochter). Eine freie Sequenz eingeschobener Knoten wird durch den Ausdruck "//" beschrieben (z.b vater//tochter). Steht ein "/" am Beginn eines Ausdrucks, so bezieht sich der Ausdruck auf den zugehörigen Root-Knoten des gegenwärtig betrachteten Kontexts. Man spricht von einem absoluten Pfad. Der relative Pfad besteht aus einer Reihe von freien Sequenzen, die den Zielknoten bestimmen.

```
PathExpr         = ("/" [RelativePathExpr]| "//" RelativePathExpr
                   | RelativePathExpr);
RelativePathExpr = StepExpr {("/" | "//") StepExpr};

StepExpr =  AxisStep | FilterStep;
AxisStep = (ForwardStep | ReverseStep) Predicates;
FilterStep = PrimaryExpr Predicates;
ForwardStep = (ForwardAxis NodeTest) | AbbrevForwardStep;
ReverseStep = (ReverseAxis NodeTest) | AbbrevReverseStep;
ForwardAxis = ("child" "::")
              | ("descendant" "::")
              | ("attribute" "::")
              | ("self" "::")
              | ("descendant-or-self" "::")
              | ("following-sibling" "::")
              | ("following" "::");
ReverseAxis = ("parent" "::")
              | ("ancestor" "::")
              | ("preceding-sibling" "::")
              | ("preceding" "::")
              | ("ancestor-or-self" "::");
```

Pfadausdrücke

Der Achsenzugriff entspricht dem XQuery/XPath Datenmodell ([W3C 03/4]).

Funktionen

Zusätzlich zu den vordefinierten XQuery-Funktionen können eigene Funktionen in einer Anfrage definiert werden.

```
FunctionDecl = "declare" "function" QName "(" [ParamList] (")" | (")" "as"
               SequenceType)) (EnclosedExpr | "external");
ParamList    = Param {"," Param};
Param        = "$" VarName [TypeDeclaration];
TypeDeclaration = "as" SequenceType;
```

Funktionsdeklaration

Beispiel einer XQuery Anfrage mit Benutzerdefinierter Funktion

In diesem Beispiel wird eine Funktion "summary" definiert, die die Daten des Ausgangsdokuments "acme_corp.xml" umformatiert und dabei aus den Ausgangsdaten berechnete Werte mit einfließen lässt.

```
define function summary($emps as element(employee)*)
   as element(dept)*
{
   for $d in distinct-values($emps/deptno)
   let $e := $emps[deptno = $d]
   return
      <dept>
         <deptno>{$d}</deptno>
         <headcount> {count($e)} </headcount>
         <payroll> {sum($e/salary)} </payroll>
      </dept>
}

summary(doc("acme_corp.xml")//employee[location = "Denver"])
```

Funktiondefinition und Anwendung

Typumwandlung

XQuery stellt eine Reihe von Funktionen zur automatischen und expliziten Typumwandlung bereit.

```
InstanceofExpr = TreatExpr [ "instance" "of" SequenceType ];
TypeswitchExpr = "typeswitch" "(" Expr ")" CaseClause+ "default"
                 ["$" VarName] "return" ExprSingle;
CaseClause     = "case" ["$" VarName "as"] SequenceType "return" ExprSingle;
CastExpr       = ComparisonExpr [ "cast" "as" SingleType ];
SingleType     = AtomicType ["?"];
```

Typumwandlung

Kommentare

```
ExprComment = "(:" (ExprCommentContent | ExprComment)* ":)";
ExprCommentContent = Char;
```

Kommentar

Anhang B: XRL+ Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://informatik.uni-stuttgart.de/XRL"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xrl="http://informatik.uni-stuttgart.de/XRL"
elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:complexType name="routing_element">
    <xs:choice minOccurs="0" maxOccurs="unbounded">
       <xs:element ref="xrl:PROPAGATE"/>
          <xs:element ref="xrl:TRANSFORM"/>
          <xs:element ref="xrl:WAIT"/>
          <xs:element ref="xrl:SEQUENCE"/>
          <xs:element ref="xrl:PARALLEL"/>
          <xs:element ref="xrl:CONDITION"/>
          <xs:element ref="xrl:FILTER"/>
          <xs:element ref="xrl:WHILE_DO"/>
          <xs:element ref="xrl:TERMINATE"/>
    </xs:choice>
 </xs:complexType>
 <xs:element name="ROUTE">
    <xs:complexType>
       <xs:complexContent>
          <xs:extension base="xrl:routing_element">
             <xs:attribute name="id" type="xs:string" use="required"/>
             <xs:attribute name="created_by" type="xs:string"/>
             <xs:attribute name="creation_date" type="xs:string"/>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 </xs:element>
 <xs:element name="CONDITION">
    <xs:complexType>
       <xs:sequence>
          <xs:element name="TRUE" type="xrl:routing_element"/>
          <xs:element name="FALSE" type="xrl:routing_element"/>
       </xs:sequence>
       <xs:attribute name="xml" type="xs:IDREF" use="required"/>
       <xs:attribute name="expression" type="xs:string" use="required"/>
    </xs:complexType>
 </xs:element>
 <xs:element name="FILTER">
    <xs:complexType>
       <xs:attribute name="xml_in" type="xs:IDREF" use="required"/>
       <xs:attribute name="expression" type="xs:string" use="required"/>
       <xs:attribute name="xml_out" type="xs:ID" use="required"/>
    </xs:complexType>
 </xs:element>
```

```
<xs:element name="PARALLEL">
   <xs:complexType>
      <xs:complexContent>
         <xs:extension base="xrl:routing_element">
            <xs:attribute name="sync" type="xs:NMTOKEN" use="required"/>
         </xs:extension>
      </xs:complexContent>
   </xs:complexType>
</xs:element>
<xs:element name="PROPAGATE">
   <xs:complexType>
      <xs:attribute name="xml" type="xs:IDREF" use="required"/>
      <xs:attribute name="system" type="xs:string" use="required"/>
      <xs:attribute name="schema" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
<xs:element name="SEQUENCE" type="xrl:routing_element"/>
<xs:element name="TERMINATE"/>
<xs:element name="TRANSFORM">
   <xs:complexType>
      <xs:attribute name="xml_in" type="xs:IDREF" use="required"/>
      <xs:attribute name="type" type="xs:string" use="required"/>
      <xs:attribute name="script" type="xs:string" use="required"/>
      <xs:attribute name="xml_out" type="xs:ID" use="required"/>
   </xs:complexType>
</xs:element>
<xs:element name="WAIT">
<xs:complexType>
   <xs:choice maxOccurs="unbounded">
      <xs:choice>
         <xs:element name="MESSAGE_EVENT">
            <xs:complexType>
               <xs:attribute name="system" type="xs:string" use="required"/>
               <xs:attribute name="schema" type="xs:string" use="required"/>
               <xs:attribute name="xml_out" type="xs:ID" use="required"/>
            </xs:complexType>
         </xs:element>
         <xs:element name="TIMER_EVENT">
            <xs:complexType>
               <xs:attribute name="time" type="xs:string" use="required"/>
               <xs:attribute name="type" default="relative">
                  <xs:simpleType>
                     <xs:restriction base="xs:NMTOKEN">
                        <xs:enumeration value="absolute"/>
                        <xs:enumeration value="relative"/>
                        <xs:enumeration value="start_relative"/>
                     </xs:restriction>
                  </xs:simpleType>
               </xs:attribute>
            </xs:complexType>
         </xs:element>
      </xs:choice>
   </xs:choice>
```

```
    <xs:attribute name="sync" type="xs:NMTOKEN" use="required"/>
    </xs:complexType>
 </xs:element>
 <xs:element name="WHILE_DO">
    <xs:complexType>
       <xs:complexContent>
          <xs:extension base="xrl:routing_element">
             <xs:attribute name="xml" type="xs:IDREF" use="required"/>
             <xs:attribute name="expression" type="xs:string" use="required"/>
          </xs:extension>
       </xs:complexContent>
    </xs:complexType>
 </xs:element>
</xs:schema>
```

Literaturverzeichnis

[14977] *Norm ISO/IEC 14977:1996(E)*
Spezifikation der EBNF-Notation

[Aal 00] W. van der Aalst and A. Kumar
XML Based Schema Definition for Support of Inter-organizational Workflow
University of Colorado and University of Eindhoven report, 2000

[Abi 97] S. Abiteboul, D. Quass u.a.
The Lorel query language for semistructured data
In Journal of Digital Libraries, Vol. 1:1, 1997

[Abi 00] S. Abiteboul, P. Bunemann, D. Suciu
Data on the Web: From Relations to Semistructured Data and XML
Morgan Kaufmann, USA, 2000

[Ber 01] A. Berlea, H. Seidl
fxt - A Transformation Language for XML Documents
Journal of Computing and Information Technology
Special Issue on Domain-Specific Languages, 2001

[Ben 03] V. Benzaken, G. Castagna, A. Frisch
CDuce: An XML-Centric General Purpose Language
In ACM ICFP03, Uppsala, Sweden, 2003

[Bon 00] Angela Bonifati, Stefano Ceri
Comparative Analysis of Five XML Query Languages
ACM Sigmod Record, Vol.29, S. 68-79, 2000

[Bon 01] Angela Bonifati, Dongwon Lee
Technical Survey of XML Schema and Query Languages
Technical report, UCLA Computer Science Dept., 2001

[Bru 03] E. Bruno, J. Le Maitre, E. Murisasco
Extending XQuery with transformation operators
Proc. of the 2003 ACM symposium on Document engineering, Grenoble, France, S. 1-8, 2003

[Cer 99] S. Ceri, S. Comai, E. Damiani, u.a.
XML-GL: a graphical language for querying and restructuring XML documents
Proceedings of the Int. World Wide Web Conference, Canada, 1999

[Cha 00] D. Chamberlin, J. Robie, D. Florescu
Quilt: An XML Query Language for Heterogeneous Data Sources
International Workshop on the Web and Databases, Dallas, Texas, 2000

[Clu 00] S. Cluet and J. Simeon
YATL: a functional and declarative language for XML
Draft manuscript, März 2000

[Con 03] Carmen Constantinescu, Uwe Heinkel, u.a.
A System for Data Change Propagation in Heterogeneous Information Systems
Enterprise Information Systems IV, Kluwer Academic Publishers, Dordrecht, Niederlande, 2003

[DeH 03] D. DeHaan, D. Toman u.a.
A Comprehensive XQuery to SQL Translation Using Dynamic Interval Coding
In Proceedings of the ACM SIGMOD. ACM Press, 2003

[Eck 00] Robert Eckstein
XML - kurz und gut
O'Reilly Verlag, Köln, 2000

[Fer 99] M. Fernandez and J. Simeon et.al.
XML query languages: Experiences and exemplars
Bell Labs, 1999

[Flo 99] D. Florescu, A. Deutsch, A. Levy, u.a.
A Query Language for XML
Proceedings of Eighth International World Wide Web Conference, 1999

[Flo 99/2] D. Florescu, A. Deutsch, A. Levy, u.a.
Querying XML Data
IEEE Data Engineering Bulletin, Vol. 22, Nr. 3, S. 10-18, 1999

[Fuh 01] N. Fuhr and K. Großjohann
XIRQL: A Query Language for Information Retrieval in XML Documents
ACM-SIGIR, New Orleans, 2001

[Har 03] H. Hosoya, B. C. Pierce
XDuce: A Typed XML Processing Language
ACM Transactions on Internet Technology, Vol. 3, Nr. 2, S. 117-148, 2003

[Kay 00] Michael Kay
XSLT Programmer's Reference
Wrox Press, USA, 2000

[Kay 03] Michael Kay
XQuery, XPath and XSLT
In XQuery from the Experts, Addison-Wesley, 2003

[Ker 01] Oliver Kersten
Konzeption eines Propagationsmanagers
Diplomarbeit Nr. 1937, Universität Stuttgart, 2001

[Lud00] B. Ludäscher, Y. Papakonstantinou u.a.
Navigation-Driven Evaluation of Virtual Mediated Views
Proc. of EDBT 2000, Konstanz, 2000

[Met 99] J. Robie
XQL (XML Query Language)
W3C Proposal, August 1999
http://www.w3.org/TR/2004/REC-xml-20040204/

[Mit 03] B. Mitschang, E. Westkämper, u.a.
Divide et Impera: A Flexible Integration of Layout Planning and Logistics Simulation through Data Change Propagation
36th CIRP-International Seminar on Manufacturing Systems, Saarbruecken, 2003

[Ran 02] Ralf Rantzau, Carmen Constantinescu, u.a.
Champagne: Data Change Propagation for Heterogeneous Information Systems
28th VLDB Conference, Hong Kong, China, 2002

[Rob 99] J. Robie, E. Derksen, u.a.
XQL (XML Query Language)
University of North Carolina, August, 1999
http://metalab.unc.edu/xql/xql-proposal.xml

[Rob 00] J. Robie, D. Chamberlin, D. Florescu
Quilt: An XML Query Language
XML Europe, Paris, June 2000

[W3C 99] World Wide Web Consortium, W3C
XML Path Language (XPath) Version 1.0
W3C Recommendation, November 1999
http://www.w3.org/TR/xpath/

[W3C 99/2] World Wide Web Consortium, W3C
XSL Transformations (XSLT) Version 1.0
W3C Recommendation, November 1999
http://www.w3.org/TR/xslt

[W3C 01] World Wide Web Consortium W3C
The XML Query Algebra
W3C Working Draft, Februar, 2001
http://www.w3.org/TR/2001/WD-query-algebra-20010215/

[W3C 01/2] World Wide Web Consortium W3C
XML Schema Part 2: Datatypes
W3C Recommendation, Mai, 2001
http://www.w3.org/TR/xmlschema-2/

[W3C 03] World Wide Web Consortium, W3C
XQuery 1.0: An XML Query Language
W3C Working Draft, November 2003
http://www.w3.org/TR/xquery/

[W3C 03/2] World Wide Web Consortium, W3C
XQuery 1.0 and XPath 2.0 Functions and Operators
W3C Working Draft, November 2003
http://www.w3.org/TR/xpath-functions/

[W3C 03/3] World Wide Web Consortium W3C
XML Path Language (XPath) 2.0
W3C Working Draft, November 2003
http://www.w3.org/TR/xpath20/

[W3C 03/4] World Wide Web Consortium W3C
XQuery 1.0 and XPath 2.0 Data Model
W3C Working Draft, November 2003
http://www.w3.org/TR/xpath-datamodel/

[W3C 04] World Wide Web Consortium, W3C
Extensible Markup Language (XML) 1.0 (Third Edition)
W3C Recommendation, Februar 2004

[Wal 00] D. Walker, Dominique Petitpierre, S. Armstrong
XMLTrans: a Java-based XML Transformation Language for Structured Data
Proc. of Coling 2000, Saarbrücken, 2000