Seminar: XML–basierte Datenbanken

XML–Anfragesprachen

Frank Klein, 16.11.1999

Inhaltsverzeichnis:

1. Einführung

XML (Extensible Markup Language) ist eine Teilmenge von SGML. Wie SGML ist XML eine Metasprache, mit deren Hilfe Markup–Sprachen definiert werden können. Gültige XML–Dokumente besitzen eine DTD (Document Type Definition), während wohlgeformte Dokumente lediglich fest vorgegebenen Syntax–Regeln genügen müssen. DTDs dienen dazu, Element– und Attributnamen sowie ihren Zusammenhang zu definieren. Ein gültiges XML–Dokument darf nur die Elemente und Attribute verwenden, die in der entsprechenden DTD vorgegeben sind. Jedes XML–Dokument, gültig oder wohlgeformt, unterliegt speziellen Regeln. Elementnamen werden innerhalb spitzer Klammern geschrieben, (<ELEMENT>). Für jedes öffnende Element muß zwingend ein schließendes mit dem selben Namen und vorangestelltem „/“ existieren (</ELEMENT>). Diese Schreibweise wird TAG genannt. Elementinhalte stehen zwischen diesen TAGs. Elemente können untereinander geschachtelt werden. Es muß nur auf die Reihenfolge des Öffnens und Schließens geachtet werden. Innerhalb jedes XML–Dokuments existiert zwingend ein einziges all umfassendes Element. Zur Unterscheidung von XML–Versionen wird zu Beginn die Versionsnummer angegeben (<?xml version="1.0"?>. Durch Einhaltung dieser Regeln läßt sich eine Semi–Struktur von XML–Dokumenten erkennen. Elemente unterliegen vordefinierten Typen. Die wichtigsten sind PCDATA (parsed character Data), ID und IDREF. Innerhalb eines XML–Dokuments ist es möglich ein Element mit einem ID–Attribut zu definieren (<PERSON id="01"></PERSON>). Das ID–Attribut ist eindeutig in dem Dokument definiert. An jeder anderen Stelle im Dokument kann nun per IDREF auf dieses konkrete Element referenziert werden (<ADRESSE idref="01"></ADRESSE>).

Ein Beispiel für eine XML–Datei und die dazugehörige DTD ist in Abbildung 1 zu finden.

```
DTD:

<?xml version="1.0"?>

<!ELEMENT person (name, vorname, telefon*, fax*)>
        <!ATTLIST person id ID #REQUIRED>

<!ELEMENT name #PCDATA>
<!ELEMENT vorname #PCDATA>
<!ELEMENT telefon #PCDATA>
<!ELEMENT fax #PCDATA>


XML−Dokument:

<?xml version="1.0"?>

<person id="001">
        <name>Klein</name>
        <vorname>Frank</vorname>
        <telefon>4711</telefon>
        <fax></fax>
</person>
```

Abbildung 1: XML−Dokument

Aufgrund der Semi−Struktur von XML−Dokumenten können diese auch durch Graphen dargestellt werden. In XML−Dokumenten existieren Hierarchien, die in solchen Graphen erkennbar sind. In Abbildung 2 wird das Beispiel aus Abbildung 1 anhand eines Graphen gezeigt.

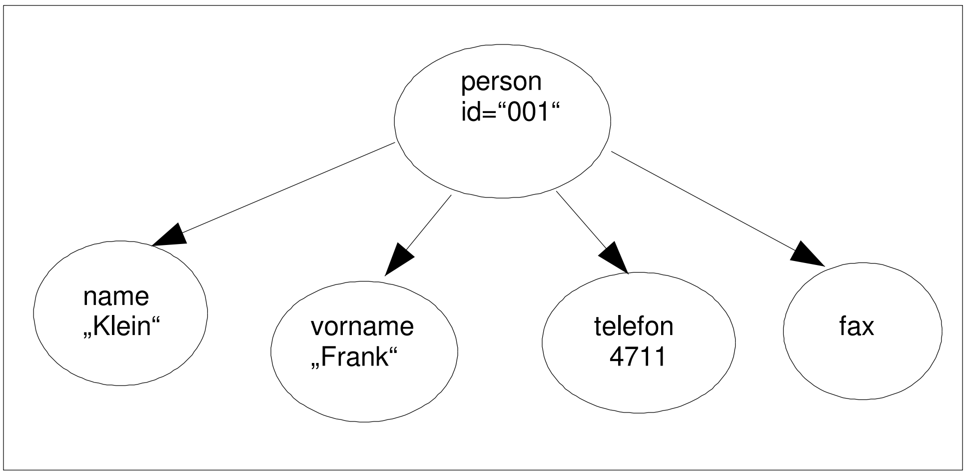


Abbildung 2: Graphische Darstellung eines XML−Dokumentes

Für die Ausgabe von XML–Dokumenten werden weitere Informationen zur Darstellung benötigt. Hierfür werden StyleSheets verwendet. XSL (extensible StyleSheet Language) ist die für XML vorgesehene Sprache. Auch die Größe von XML–Dateien ist ein Problem. XML–Dateien können unter Umständen sehr groß werden. Anfragesprachen sollen hier helfen, einen Überblick über den Inhalt der Dokumente beizubehalten oder nur gewünschte Inhalte herausfiltern zu können. Außerdem dienen sie dazu, neue Dokumente mit vom Benutzer gewünschten Inhalten zu generieren. Des weiteren dienen sie dazu, mehrere Dokumente nach Objekten zu durchsuchen (z.B. Suchmaschinen), sogenanntes Information retrieval.

1.1 Grundidee von Anfragesprachen

Anfragesprachen sollten verschiedenen Anforderungen genügen:

1. Sprache muß flexibel sein
   Eine Anfragesprache muß flexibel genug sein, um gültige und wohlgeformte XML–Dokumente verstehen zu können.

   - gültig: XML–Dokument besitzt eine DTD
   - wohl geformt: XML–Dokument ohne DTD aber mit Tag–Regeln

2. Anfragen über mehrere Dokumente (Webseite)
   Eine gute Anfragesprache kann Anfragen über mehrere Dokumente stellen. Dies sollte natürlich auch bei XML–GL, eine Anfragesprache die im folgenden kapitel näher erläutert wird, erfüllt sein.

3. Möglichkeit zur Unterstützung von DTDs, XML–basierte Metadaten etc.
   XML–GL sollte so flexibel sein, daß es auch weitere Sprachen (außer XML) verstehen kann. Dies ist insbesondere nützlich für weitere Versionen von XML oder verwandte Sprachen.

4. Erstellbarkeit neuer Dokumente
   Neue Dokumente sollen aus Anfragesprachen generiert werden können. Dies ist interessant, wenn die Ergebnisse weiter verwendet werden sollen.

5. Unterstützung regulärer Ausdrücke
   Reguläre Ausdrücke sind ein mächtiges Instrumentarium zur Darstellung von Ausdrücken. Eine Anfragesprache sollte dies unterstützen können, insbesondere wenn sie auch als Austauschsprache verwendet wird.

6. Numerische Funktionen
   Numerische Funktionen haben zwar keinen großen Stellenwert bei der Beurteilung einer Sprache, sind jedoch sehr nützlich um beispielsweise Nummerierung einführen zu können.

7. Geordnete und ungeordnete Anfragen

Oftmals ist es sinnvoll, daß die Ergebnisse in geordneter Reihenfolge ausgegeben werden. Daher sollte dies gleich auf der Anfrage−Ebene unterstützt werden.

8. Einfache Verständlichkeit
   In der heutigen Zeit ist es nicht mehr sinnvoll Sprachen zu entwickeln, die derart komplex sind, daß sie nur noch von Experten verstanden werden. Gerade im Bereich von XML soll auch Laien die Möglichkeit gegeben werden, Seiten zu erstellen und Anfragen darauf zu errichten.

1.2 XML−Anfrageprachen

Als Anfragesprachen für XML existieren derzeit verschiedene Vorschläge. Zwei dieser Vorschläge werden nachfolgende näher behandelt. Dies sind im speziellen:

- XML−GL
  XML Graphical Language

- XSL / XSLT
  XML Stylesheet Language

2. XML–GL

Wie in Abschnitt 1 gesehen, lassen sich XML–Dokumente graphisch darstellen. Es liegt also nahe, Anfragen ebenfalls graphisch darzustellen. Damit XML–Dokumente in eine nutzbare Darstellung gebracht werden können, wurde eine spezielle Sprache namens XML–GDM (XML–Graphical Data Model) eingeführt. Diese dient der Darstellung von XML–Dokumenten mit der Idee Anfragen stellen zu können. Der Vorteil graph–basierter Anfragen liegt darin, daß sie intuitiv verständlich sind. Es ist nicht notwendig, die Syntax und Semantik von Programmiersprachen zu erlernen.

Abbildung 3 zeigt eine DTD zur Abwicklung von Buchbstellungen.

```
<!ELEMENT bestellung (liefernzu, kontakt?, gegenstand+, datum)>
        <!ATTLIST bestellung nummer PCDATA #REQUIRED>
<!ELEMENT liefernzu (gesamtadresse | referenz)>
<!ELEMENT kontakt (referenz | PCDATA)>
<!ELEMENT gesamtadresse (Firma?, Stadt, Adresse+)>
<!ELEMENT referenz EMPTY>
        <!ATTLIST referenz customer IDREF>
<!ELEMENT person (vorname?, nachname, gesamtadresse)>
        <!ATTLIST person id ID>
<!ELEMENT Firma PCDATA>
<!ELEMENT adresse PCDATA>
<!ELEMENT stadt PCDATA>
<!ELEMENT datum (tag, monat, jahr)>
<!ELEMENT tag PCDATA>
<!ELEMENT monat PCDATA>
<!ELEMENT jahr PCDATA>
<!ELEMENT gegenstand (buch, menge)>
<!ELEMENT buch (isbn, titel, preis)>
<!ELEMENT autor (vorname, nachname)>
<!ELEMENT vorname PCDATA>
<!ELEMENT nachname PCDATA>
<!ELEMENT isbn PCDATA>
<!ELEMENT titel PCDATA>
```

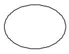
Abbildung 3: Beispiel–DTD

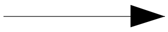
Alle nachfolgenden Beispiele beziehen sich auf dieses DTD.

2.1 XML–GDM

Für die Umwandlung von XML–Seiten in XML–GDM–Graphen müssen Konventionen vorgegeben werden, die hier aufgeführt werden:

- XML–Elemente werden durch Rechtecke dargestellt. Beschriftet werden sie mit dem Elementnamen. 

- Elementinhalte sind erkennbar an einem Kreis. Sie besitzen einen Namen und einen Typ. 

- Elemente und ihre Inhalte werden durch Beziehungspfeile dargestellt. Sie sind richtungsabhängig. 

- Kardinalitäten werden an die Beziehungspfeile geschrieben. 

- Xor–Abhängigkeit wird durch einen Halbkreis samt der Beschriftung „xor“ gekennzeichnet. 

Wichtig ist vor allem, daß die hierarchische Struktur erhalten bleibt. Anhand eines XML–GDM–Graphen läßt sich die DTD des XML–Dokumentes ablesen.

Eine detaillierte Gegenüberstellung der Zusammenhänge ist in Abbildung 4 aufgelistet.

XML DTD	***XML–GDM***
Nicht–terminierendes Element E	Objekt mit Namen E
Zwei nicht–terminierende Elemente E1, E2. mit E1 Unterelement E2	Beziehungspfeil zwischen Objekten E1, E2
Terminierendes Element E1, nicht–terminierendes Element E2, mit E1 Unterelement E2	Objekt E1 besitzt Eigenschaft E2, mit Typ PCDATA
Disjunktion (|), nicht–terminierendes Element E, Unterelemente E1, ... , En, xor	Beziehung (Pfeile) von Objekt E zu E1, ... , En, beschriftet mit xor
Gemischte Elemente, Element E1, mit Unterelement E2 oder PCDATA	Objekt E1 mit Disjunktion E2, PCDATA (s. oben)
Druckbare Atrribute und object–identifier	Eigenschaft eines Objektes E, durch schwarze Färbung gekennzeichnet
Objekt–valued Attribute	Beziehung zwischen Element E und Element ANY (XML–GDM Objekt)
Kardinalitäten (*,+,?)	*: (0:n) +: (1,n) ?: (0:1) objevt–valued IDREF: (1:1) oder (0:1) wenn IMPLIED IDREF: (1,n) oder (o:n) wenn IMPLIED sonst: (1:1)
Ordnung	Pfeilrichtung

Abbildung 4: Zusammenhang XML–DTD –– XML–GDM

In Abbildung 5 wird das DTD−Beispiel aus Figur 3 durch XML−GDM dargestellt.

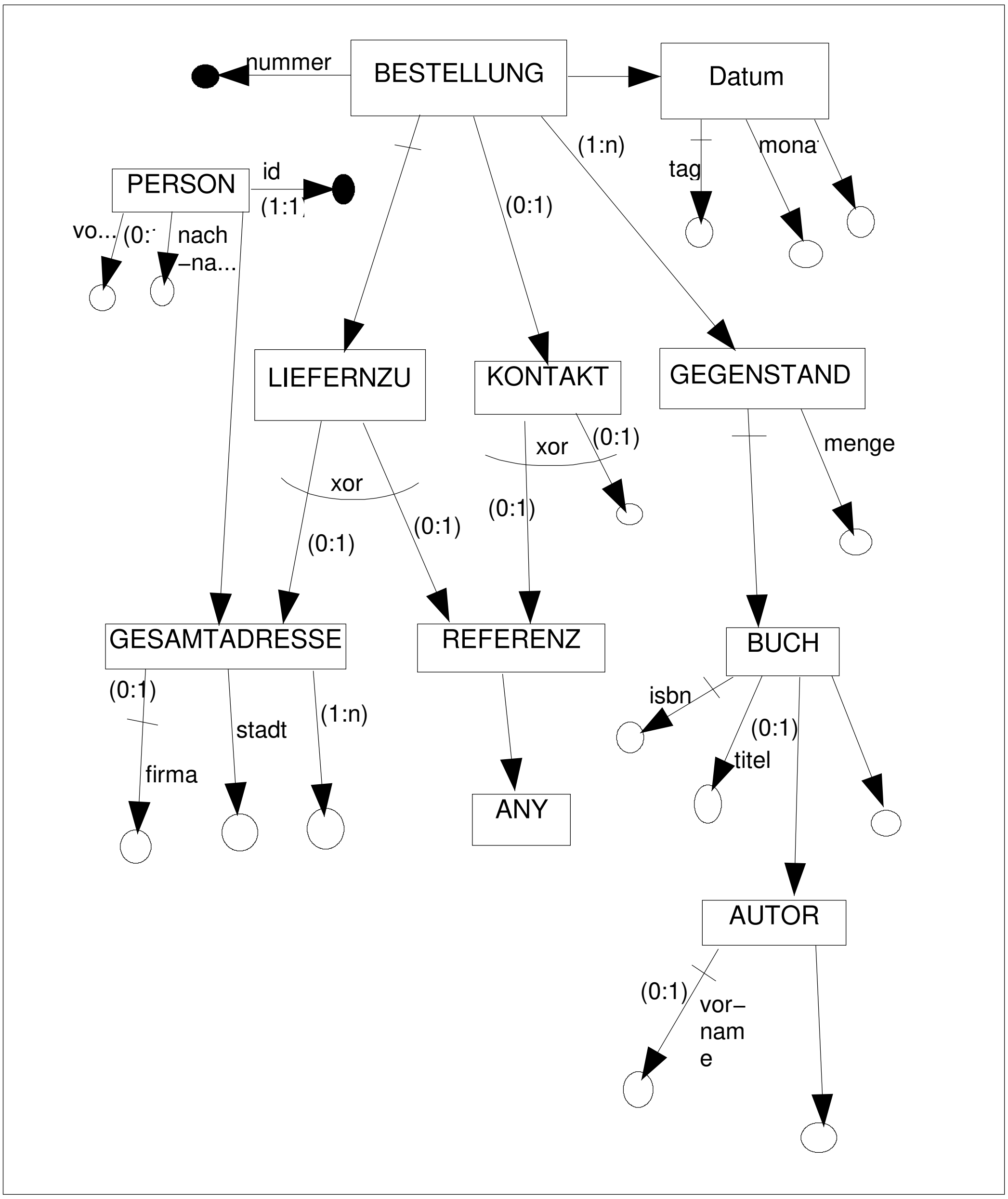


Abbildung 5: XML−GDM−Graph

2.2 XML–GL [CCDFPT99]

XML–GDM–Graphen können nun für Anfragen eingesetzt werden. Hierfür wird XML–GL als Anfrageprache benötigt. XML–GL beinhaltet alle Merkmale die in Kapitel 1.1 erwähnt wurden. Zum besseren Verständnis wird hier kurz ein Vergleich zwischen der bekannten Anfragesprache SQL und XML–GL gezogen.

- *„Extract Scope“* in XML–GL gibt an auf welche Dokumente die Anfrage gestellt werden soll. Es ist vergleichbar mit dem *„from“* von SQL.
- *„Match“* in XML–GL ist optional und gibt logische Verknüpfungen wider. Hier können Anfragen spezifiziert werden. In SQL wird dies durch *where* beschrieben.
- *„Clip Elemente“* in XML–GL sind die Ausgabeelemente. Hiermit wird angegeben, welche der gefundenen Elemente wirklich ausgegeben werden sollen. In SQL wird dies durch *select* ausgedrückt.
- *„Construct“* in XML–GL dient dazu, neue Elemente hinzuzufügen. Dies ist mit dem *create view*–Befehl aus SQL vergleichbar.

Eine XML–Anfrage wird durch ein Paar vom XML–GDM–Graphen dargestellt. Anfrage und Ergebnis werden durch eine horizontale Linie getrennt.
Auf der linken Seite steht die Anfrage bzw. das *Extract Scope* oder *Match*. Neben den Elementnamen besteht hier auch die Möglichkeit anzugeben, welche Dokumente durchsucht werden sollen. Erlaubt ist hier auch die Angabe von URLs sowie von Wildcards (*,?).
Auf der rechten Seite wird das gewünschte Ergebnis dargestellt. Aufgrund der Darstellung in XML–GDM wird hier auch die Ergebnis–DTD angegeben.

In Abbildung 6 wird eine einfache Anfrage beschrieben. Auf der extract–Seite wird nach dem Element BUCH gesucht. Dies soll in allen Dokumenten unter der angegebenen URL gesucht werden. Ausgegeben werden sollen alle gefunden Bücher einschließlich aller Unterelementen.

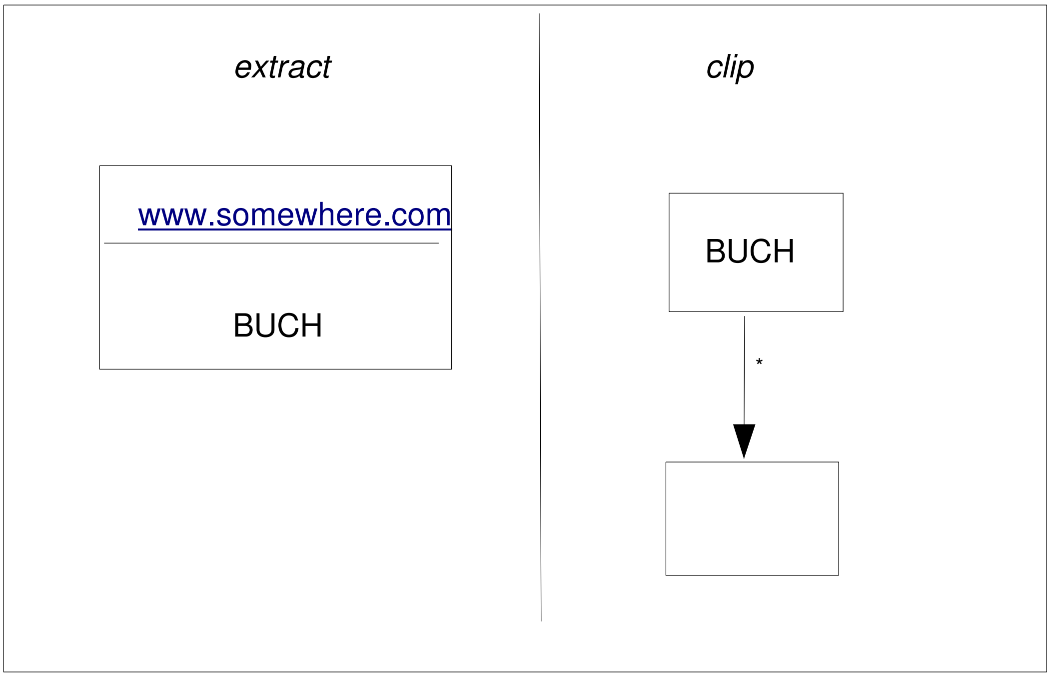


Abbildung 6: XML–GL Anfrage

Auf der Ergebnis–Seite ist anhand der Kardinalität (*) zu erkennen, daß sämtliche Unterelemente ausgegeben werden sollen. Oftmals ist es wünschenswert die Hierarchie–tiefe genauer zu spezifizieren. XML–GL bietet daher die Möglichkeit, die Such– und die Ausgabetiefe zu definieren. Durch Angabe eines festen Elementes (BUCH in Abbildung 6) wird auf die Hierarchieebene kein Einfluß genommen. In Abbildung 6 werden die XML–Dokumente komplett durchsucht, und alle BUCH–Elemente ausgegeben, unabhängig von der Hierarchie. Tiefenangaben können nur relativ zu einem vorgegebenen Element spezifiziert werden.

Abbildung 7 zeigt diese Möglichkeiten:

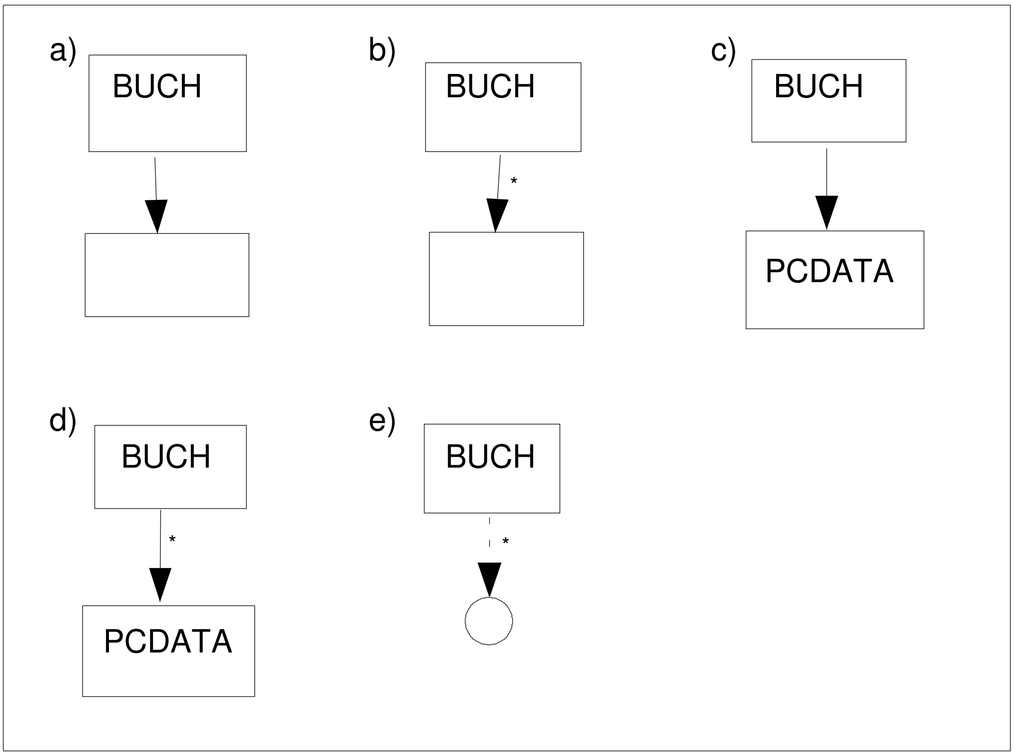


Abbildung 7: Tiefenangaben

a) In diesem Beispiel wird auf das direkt nachfolgende Element verwiesen. Durch Auslassen einer Kardinalität wird immer auf die nachfolgende Ebene gezeigt.

b) Durch Angabe einer Kardinalität (hier *) kann auf diverse Elementtiefen verwiesen werden. In diesem Beispiel werden alle Unterelemente durchsucht bzw. ausgegeben.

c) PCDATA ist ein vordefinierter XML–Datentyp (parsed character Data). Durch Angabe dieses Typs wird verlangt, daß das referenzierte Element nicht leer sein darf. In einem XML–GDM–Graphen bedeutet dies auch, daß das Element ein terminierendes Element ist (vgl. Abbildung 5). Dieses Beispiel verweist daher auf alle terminierenden Elemente der ersten Ebene (vgl. Beispiel a).

d) Im Vergleich zu Beispiel c) wird hier durch die Angabe einer Kardinalität auf alle terminierenden Elemente aller Ebenen verwiesen.

In den Beispielen a)–d) werden als Ergebnis auch alle Zwischenelemente ausgegeben.

e) Dieses Beispiel ist etwas komplexer. Ein fest vorgegebenes Element (dargestllt durch einen Kreis) wird irgendwo (Kardinalität *) unterhalb des BUCH–Elementes (dargestellt durch das BUCH– Rechteck) gesucht. Bei Ausgabe sollen jedoch alle Zwischenelemente ausgelassen werden. Dies wird durch die gestrichelte Linie dargestellt. Gestrichelte Objekte oder Linien bedeuten im Vergleich zu durchgezogenen eine Negation.

Wenn man die in Abbildung 6 gezeigte Anfrage, verschiedenee Ausgabedarstellungen vorgibt, erhält man unterschiedliche Ergbenisse (Abbildung 8).

```
a)      <BUCH>
            <TITEL>Einführung in XML–GL</TITEL>
            <AUTOR></AUTOR>
        </BUCH>


b)      <BUCH>
            <TITEL>Einführung in XML–GL</TITEL>
            <AUTOR>
                <VORNAME>Frank</VORNAME>
                <NAME>Klein</NAME>
                <ADRESSE></ADRESSE>
            </AUTOR>
        </BUCH>


c)      <BUCH>
            <TITEL>Einführung in XML–GL</TITEL>
        </BUCH>


d)      <BUCH>
            <TITEL>Einführung in XML–GL</TITEL>
            <AUTOR>
                <VORNAME>Frank</VORNAME>
                <NAME>Klein</NAME>
            </AUTOR>
        </BUCH>


e)      <BUCH>
            <NAME>Klein</NAME>
        </BUCH>
```

Abbildung 8: Ergebnisse durch Tiefenangabe

2.3 Konstruktion

Durch Anfragen auf Dokumenten ist es für den Nutzer nicht nachvollziehbar, wie das Ergebnis aussehen wird. Hierzu werden Möglichkeiten zur Konstruktion neuer XML–Dokumente verlangt. XML–GL unterstützt hierfür verschiedene Implementationen.

a) *constructed elements*
Die einfachste Möglichkeit, ein neues Dokument zu erstellen, ist die Aneinanderreihung der Ergebnisse, die von einem neuen Element erfaßt werden.

b) *list*
Zur Unterscheidung der einzelnen Ergebniselemente kann durch Verwendung einer Liste jedes Ergebnis in ein neues Element eingebettet werden.

c) *grouping list*
Mit Hilfe von grouping lists können Ergebnisse sortiert und gebündelt in neue Elemente eingefügt werden.

Die graphische Notation ist in Abbildung 9 zu sehen.

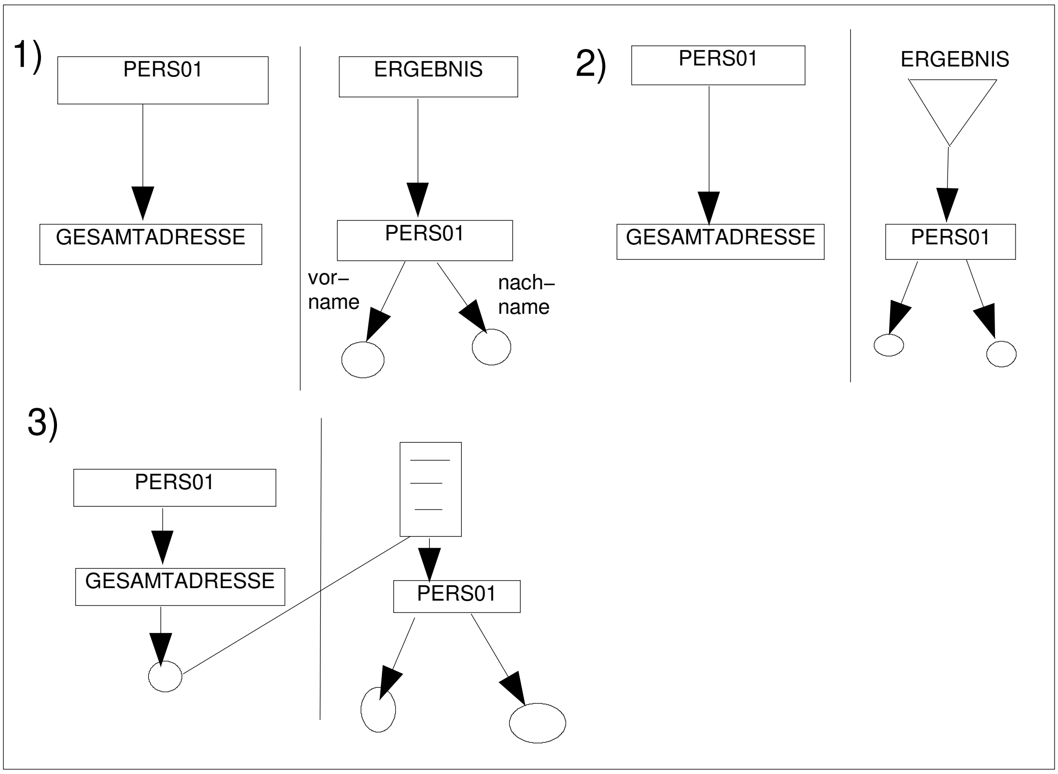


Abbildung 9: Konstruktion

Ein Auszug aus einem entsprechenden XML–Ergebnis wird in Abbildung 10 aufgeführt bezüglich der Anfragen aus Abbildung 9.

```
a)      <ERGEBNIS>
                <PERSON id=“C001“>
                        <VORNAME>Frank</VORNAME>
                        <NACHNAME>Klein</NACHNAME>
                </PERSON>
        </ERGEBNIS>
        <ERGBENIS>
                <PERSON id=“C002“>
                        <NACHNAME>Weikum</NACHNAME>
                </PERSON>
        </ERGEBNIS>
        <ERGEBNIS>
                <PERSON id=“C003“>
                        <VORNAME>Mark</VORNAME>
                        <NACHNAME>Schweppe</NACHNAME>
                </PERSON>
        </ERGEBNIS>


b)      <ERGBENIS>
                <PERSON id=“C001>
                        <VORNAME>Frank</VORNAME>
                        <NACHNAME>Klein</NACHNAME>
                </PERSON>
                <PERSON id=“C002“>
                        <NACHNAME>Weikum</NACHNAME>
                </PERSON>
                <PERSON id=“C003“>
                        <VORNAME>Mark</VORNAME>
                        <NACHNAME>Schweppe</NACHNAME>
                </PERSON>
        </ERGEBNIS>


c)      <ERGEBNIS>
                <PERSON id=“C001“>
                        <VORNAME>Frank</VORNAME>
                        <NACHNAME>Klein</NACHNAME>
                </PERSON>
                <PERSON id=“C002“>
                        <NACHNAME>Weikum</NACHNAME>
                </PERSON>
        </ERGEBNIS>
        <ERGEBNIS>
                <PERSON id=“C003“>
                        <VORNAME>Mark</VORNAME>
                        <NACHNAME>Schweppe</NACHNAME>
                </PERSON>
        </ERGEBNIS>
```

Abbildung 10: Ergebnisse durch Konstruktion

3. XSL [QLPP99]

Zur Darstellung von XML–Dokumenten werden Zusatzinformationen benötigt. Hierzu wird XSL (extensible StyleSheet Language) verwendet. Über XSL wird definiert, wie die Ausgabe eines XML–Dokumentes dargestellt wird. XSL bietet die Möglichkeit, genau wie Anfragesprachen, neue Dokumente aus bereits bestehenden zu erstellen. Daher bietet sich an, nach weiteren Gemeinsamkeiten zwischen XSL und einer Anfragesprache zu suchen.

XSL basiert auf *templates*, die für die Ausgabedarstellung zuständig ist. Zur Unterscheidung einzelner Elemente werden *patterns* eingesetzt. Templates und Patterns werden im nachfolgenden genauer beschrieben.

3.1 Pattern

Wie in Kapitel 1 erwähnt, sind XML–Dokumente semi–strukturiert. Eine andere Art der Interpretation solcher Dokumente ist die Möglichkeit, sie in Form eines Filesystems darzustellen. Ein XML–Dokument besitzt ein einziges umfassendes Element. Dies kann, wie bei einem Filesystem, als root–Element (/) interpretiert werden. Alle direkten Unterelemente werden an dieses root–Element angehangen (/Kategorie). Es kann auch vorkommen, daß Zwischenelemente nicht bekannt sind, oder nicht angegeben werden wollen. Aus diesem Grund sind Wildcards (//) erlaubt. Zur weiteren Spezifizierung des Elementes sind Attribute zugelassen.

In Abbildungen 11a und 11b wird eine XML–Datei angegeben, auf die um Laufe des Kapitels verwiesen wird.

DTD:

```
<?xml version="1.0"?>

<!ELEMENT Kategorie (Produkt, sku)>
<!ELEMENT Produkt (sku)>
        <!ATTLIST Produkt size PCDATA #REQUIRED>
<!ELEMENT sku #PCDATA>
```

XML–Dokument:

```
<?xml version="1.0"?>

<Kategorie>
        <Produkt size="10">
                <sku>Irgendein Elementinhalt</sku>
        </Produkt>
</Kategorie>
```

Abbildung 11a: XML–Dokument

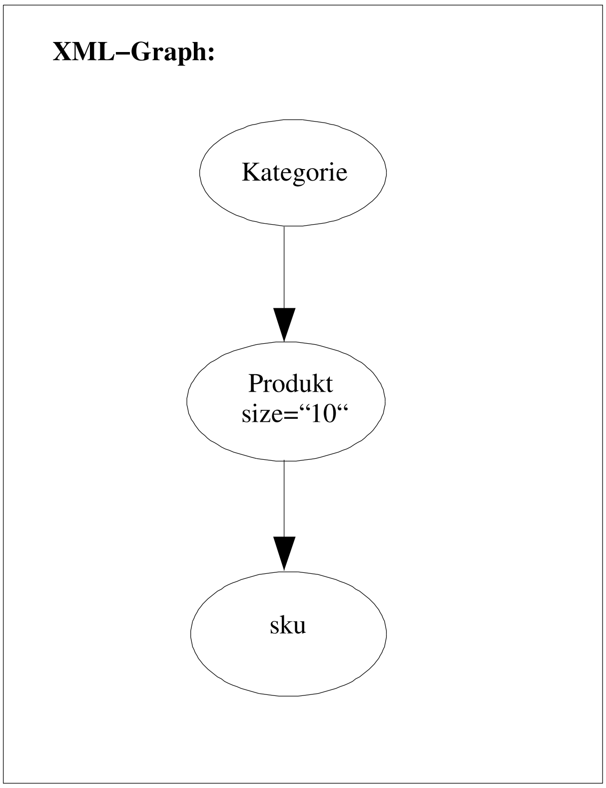


Abbildung 11b: XML–Graph zu Abbildung 11a

Abbildung 12 verdeutlicht den Zusammenhang von Patterns anhand des Beispiels aus Abbildung 11.

Pattern	***Beschreibung***
/	Root–Element
Kategorie/Produkt	Produkt–Elemente, direkte Nachfolger von Kategorie
Kategorie/Produkt/sku	Sku–Elemente, direkte Nachfolger von Produkt, direkte Nachfolger von Kategorie
Kategorie//sku	Sku–Elemente, irgendwo innerhalb Kategorie
//sku	Sku–Elemente, irgendwo
Produkt[Attribut(size)="10"]	Produkt–Elemente, Größe = 10
Produkt[Attribut(size)="10"]/sku	Sku–Elemente, direkte Nachfolger von Produkt mit Größe = 10

Abbildung 12: Patterns

Durch Angabe von Patterns wird auf Elemente referenziert. Dies ist vergleichbar mit einer *SELECT* –Anweisung in SQL. Werden Attribut–Kriterien angegeben, kann dies mit der *WHERE*–Anweisung verglichen werden. Hier ist eine Gemeinsamkeit zwischen XSL und SQL zu erkennen.

3.2 Templates

Templates werden zur Konstruktion der Ausgabe verwendet. Definiert werden sie durch *xsl:template–name*. Patterns werden verwendet, um template–Aufrufe auf gezielte Elemente anwenden zu können. Dies geschieht durch das match–Attribut (<xsl:template match="Kategorie/Produkt">).

Aus Abbildung 13 werden einige Beispiel–templates ersichtlich.

Template	***Beschreibung***
<xsl:template match="sku"> SKU: <xsl:value–of/> </xsl:template>	Für jede Instanz sku, konstruiere test „SKU:", gefolgt durch Inhalt des Elements.
<xsl:template match="//product"> NAME: <xsl:value–of select="attribute(name)"/> <xsl:apply–templates select="sku"/> </xsl:template>	Konstruiere Namen aller Produkte und zugehöriger skus, Name erstellt durch name–attribute
<xsl:template match="category"> <SKUS CATEGORY="{attribute(name)}"> <xsl:apply–templates select="product/sku"/> </xsl:template>	Konstruiere ein sku–Element für jede Kategorie. Name erstellt durch name–attribute. Liste alle skus innerhalb der Child–Elemente

Abbildung 13: Templates

Templates können auch zur Sortierung der Ausgabe eingesetzt werden, vergleichbar mit einem *ORDER BY* aus SQL. Mit Hilfe von templates ist es möglich neue Dokumente zu generieren. Dies ist vergleichbar mit dem Ergebnis von Anfrageprachen.

3.3 Vergleich XSL – Anfragesprache

Im vorangegangen Abschnitt wurde gezeigt, daß XSL als StyleSheet Language durchaus Gemeinsamkeiten mit Anfragesprachen aufweist. XSL besitzt jedoch einige Einschränkungen gegenüber Anfragesprachen. So kann XSL beispielsweise nur auf einzelne Dokumente angewandt werden. Solche Unterschiede sind auf die Verwendungszwecke der einzelnen Sprachklassen zurückzuführen.

Anfragesprachen zielen auf komplexe Anfragen ab. Diese sollen nach Möglichkeit auch wiederverwendbar sein. Informationen sind oftmals über mehrere Dokumente verbreitet. Daher müssen auch Anfragen über eine Menge von Dokumenten möglich sein.

Bei StyleSheet–Sprachen wird gewünscht, daß rekursive und heterogene Daten unterstützt werden. Numerische Funktionen, wie Numerierung, dienen dem Nutzer dazu, eine strukturierte Ausgabe zu generieren. Im Gegensatz zu Anfragesprachen wird für StyleSheets eine Unterstützung mehrerer Ausgabeformate gewünscht (HTML, PDF, etc.).

4. XSLT [JC99]

XSL bietet die Möglichkeit, XML–Dokumente gezielt ausgeben zu können. XSLT (XSL Transformation Language) hingegen wird genutzt, um XML–Dokumente gezielt in andere XML–Dokumente umzuwandeln. XSLT basiert, genau wie XSL, auf templates und patterns. Erweitert wurde es unter anderem um bedingte Anweisungen (if, when, etc.).

In Abbildung 14 werden einige Beispiele gezeigt, wie sie auf eigene Dokumente angewandt werden könnten. Deutlich ist hier die template– und pattern–Struktur aus Kapitel 3 ersichtlich.

a) „Suche nach Vornamen in Person!“

```
<xsl:template match="person">
        <P>
                <xsl:value–of select=“@vorname“/>
                <xsl:text> </xsl:text>
                <xsl:apply–templates>
        </P>
<xsl:template>
```

b) „Finde alle Bestellungen und ihre Unterelemente, und füge sie einem block–Element hinzu!“

```
<block>
        <xsl:apply–templates select=“//bestellung“/>
</block>
```

c) bedingte Anweisung

```
<xsl:choose>
        <xsl:when test=“ancestor(H1)“>
                <xsl:apply–templates/>
        </xsl:when>
</xsl:choose>
```

Abbildung 14: XSLT–Beispiele

5. Zusammenfassung

Für XML als ein semi–strukturiertes Dokument, bietet sich eine Graph–basierte Anfragesprache wie XML–GL geradezu an. Anfragen können mit Hilfe dieser Methoden intuitiv dargestellt werden. Nachteilig ist meiner Meinung nach, daß XML–GDM–Graphen bei komplexen Anfragen unübersichtlich werden können. Dann müßten die Graphen farbig gekennzeichnet werden, bzw. gar auf eine dritte Darstellungsbene erweitert werden. Für die meisten Anfragen stellt diese Anfragesprache allerdings eine gute Alternative zu den übrigen dar. Gerade für unerfahrene Benutzer ist sie sehr gut geeignet.

XSL dagegen als Anfragesprache zu verwenden war meiner Meinung nach nicht der ursprüngliche Entwicklunsgedanke dieser Sprache. Für einfache Anfragen über ein einziges Dokument ist sie jedoch ebenfalls geeignet. XSL wird bei einer Ausgabe auf jeden Fall benötigt, da ansonsten kein annehmbares Ergebnis erzielt wird. Für einfache Anfragen wird daher kein großer Aufwand benötigt. Unter anderem weil XSL nur auf einzelne Dokumente begrenzt ist, ist es nur begrenzt als Anfragesprache einsetzbar.

6. Literaturverzeichnis

[QLPP99] The Query Language Position Paper of the XSL Working Group.
http://www.w3.org/TandS/QL/QL98/pp/xsl–wg–position.html,
Zugriff: 20.10.1999.

[CCDFPT99] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi,
L Tanca:
XML–GL: A graphical language for querying and reshaping
XML.
WWW8 / Computer Networks 31(11–16): 1171–1187, 1999.

[JC99] J. Clark: XSL Transformations (XSLT).
http://www.w3.org/TR/xslt,
Zugriff: 20.10.1999.