- Einstellung für die Behandlung von begrenzendem Leerraum
- Variablendeklarationen, auch für externe Variablen
- Benutzerdefinierte Funktionen

Versionsangabe und Moduldeklaration gehören nicht zum Prolog, sondern müssen diesem vorangehen.

8.3 Das Verarbeitungskonzept von XQuery

Die XQuery-Spezifikation definiert nicht nur eine Sprache und ein Datenmodell, sondern auch ein Verarbeitungskonzept für die Sprache. Das Verarbeitungsmodell kann beim Verständnis des Verhaltens einer XQuery-Implementierung helfen; deshalb soll es hier kurz erklärt werden. Abbildung 8–1 gibt eine grafische Darstellung des Verarbeitungskonzepts. Alle grau unterlegten Komponenten gehören nicht zum eigentlichen Umfang von XQuery, sondern werden von XQuery vorausgesetzt.

Statischer Kontext

Die Auswertungsumgebung (d. h. die XQuery-Implementierung oder die Umgebung, in die diese eingebettet ist) kann einen statischen Kontext für die Auswertung vorgeben, also beispielsweise Variablenwerte, Verarbeitungsdirektiven und mehr. Wir werden weiter unten sehen, was alles zu einem solchen statischen Kontext gehört. Es handelt sich jedenfalls um Einstellungen, die von den zu verarbeitenden Daten unabhängig sind. Dieser statische Kontext wird in der statischen Analyse noch erweitert.

8.3.1 Statische Analyse

Die erste Phase der Auswertung wird »Statische Analyse« genannt, weil sie nur von der Anweisung und dem statischen Kontext, aber nicht von den verarbeiteten Daten abhängt. Ein XQuery-Hauptmodul wird zunächst analysiert (»parse«). Der XQuery-Prolog wird benutzt, um den statischen Kontext zu erweitern und zu verändern (Abschnitt 8.3.4). Die eigentliche Anfrage wird in einen Operatorbaum, d. h. eine interne Darstellung, überführt. Dabei wird geprüft, ob alle Typnamen, Namensraumpräfixe, Funktionsnamen und Variablennamen, die nicht in der Anfrage selbst definiert werden, im statischen Kontext vorliegen. Ist dies nicht der Fall, ist die Anfrage fehlerhaft und wird nicht weiter verarbeitet (Abschnitt 7.4.4).

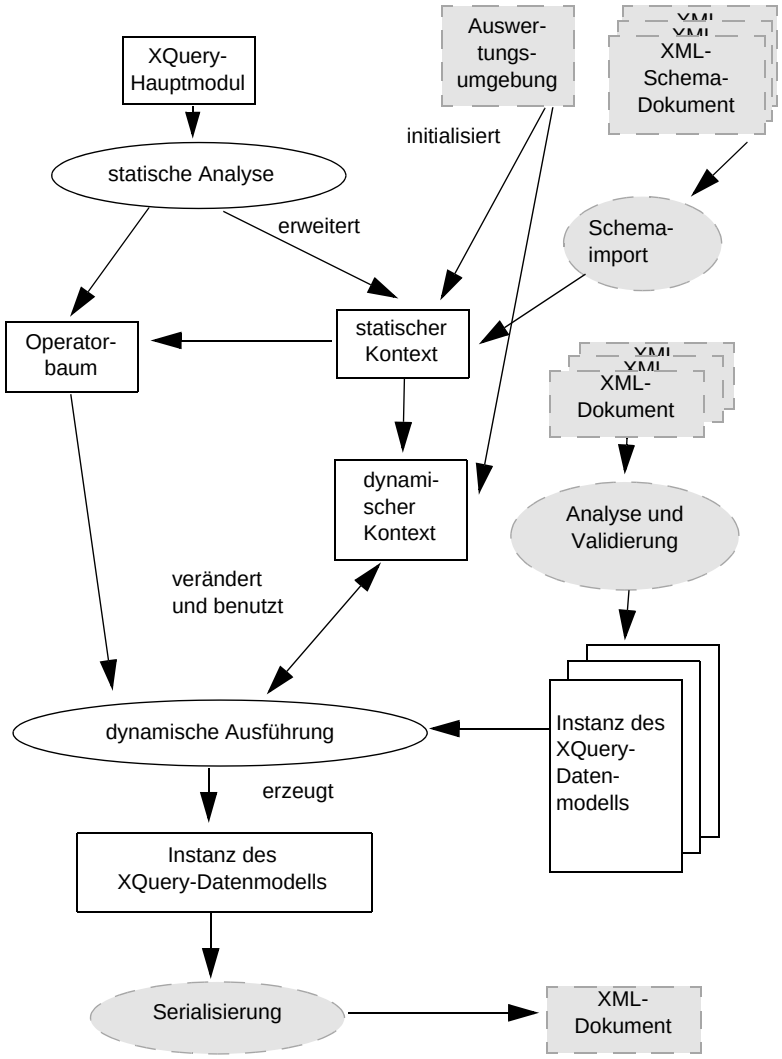

Abb. 8–1 *Das Verarbeitungskonzept von XQuery (nach [W3C-19])*

Statische Typprüfung

Im Rahmen der statischen Analyse kann eine XQuery-Implementierung bereits konstante Ausdrücke berechnen und einige Typzuweisungen und -prüfungen vornehmen (statische Typanalyse). So kann die folgende Anfrage schon in der statischen Analyse mit einem Fehler zurückgewiesen werden, weil die zu vergleichenden Variablen `$i` und `$j` einen inkompatiblen Typ haben:

```
import schema namespace x="http://example.org/x"
for $i in (1 to 3)
for $j as element(*, xs:string) in doc("Beispiel.xml")//x:Zahl
where $i gt $j
return
  ($i, "ist größer als", $j)
```

Ebenso würde ein Fehler erzeugt, wenn in der importierten Schemadefinition kein Element `Zahl` aus dem Namensraum `http://example.org/x` definiert ist oder wenn dieses nicht zu dem Typ `element(*, xs:string)` passt, weil es z. B. komplexen Inhalt hat. Man bemerkt hier, dass die statische Analyse konservativ ist, wenn es um die Beurteilung der Typsicherheit geht. Die Anfrage könnte ohne statische Typprüfung durchaus erfolgreich bearbeitet werden, ohne dass ein Typfehler auftritt, wenn nämlich in dem Dokument `Beispiel.xml` kein Element `x:Zahl` auftritt. In diesem Fall liefert die Anfrage eine leere Sequenz.

Konservatives Verhalten

Das Verhalten der statischen Typprüfung kann auf den ersten Blick erstaunlich sein. So wird folgende Anfrage einen statischen Typfehler liefern, wenn im Schema das Vorkommen der Elemente `Bonus` und `Gehalt` nicht auf höchstens 1 begrenzt ist:

```
for $a in fn:collection("Angestellte")
return $a/Gehalt + $a/Bonus
```

Gerade bei statischer Typprüfung erweist sich die `typeswitch`-Anweisung als hilfreich ([Katz04] und [MBC+04]). Die Anweisung

```
typeswitch ($Angestellter)
  case $a as element(*, Arzt_T) return $a/Fähigkeit
  case $k as element(*, Pfleger_T) return $k/Zertifikat
```

wird die statische Typprüfung problemlos bestehen, während die folgende Anweisung, die bezüglich ihres dynamischen Verhaltens identisch zu der obigen ist, an der statischen Typprüfung scheitern wird:

```
if ($Angestellter instance of element(*, Arzt_T)
  then  $Angestellter/Fähigkeit
  else if ($Angestellter instance of element(*, Pfleger_T)
       then $Angestellter/Zertifikat
```

Vorteil von treat as

Das liegt daran, dass die Ableitungsregeln, wie sie in [W3C-17] definiert sind, nicht beweisen können, dass auf das Element `Zertifikat` nur zugegriffen wird, wenn `$Angestellter` vom Typ `Pfleger_T` ist. In einer XQuery-Implementierung, die statische Typprüfung unterstützt (Abschnitt 9.1), ist daher nur die erste Anfrageformulierung möglich. Die `treat as`-Anweisung wurde nur wegen der statischen Typprüfung eingeführt. Sie sichert einen Typ zu, so dass sich die statische Typprüfung darauf verlassen kann (dafür wird gegebenenfalls in der dynami-

schen Ausführung ein Fehler erzeugt, wenn die Zusicherung nicht zutrifft). So kann die obige Anfrage noch gerettet werden:

```
if ($Angestellter instance of element(*, Arzt_T)
  then  ($Angestellter treat as Arzt_T)/Fähigkeit
  else if ($Angestellter instance of element(*, Pfleger_T)
       then ($Angestellter treat as Pfleger_T)/Zertifikat
```

Statische Typprüfung ist besonders dann hilfreich, wenn Daten aus einem bekannten und definierten Eingabeschema in ein bekanntes und definiertes Ausgabeschema umgeformt werden sollen. Allerdings muss man sehr diszipliniert mit Typen umgehen, um keine unangenehmen Überraschungen zu erleben. Schon folgende triviale Anfrage liefert einen statischen Typfehler:

```
let $x as xs:decimal := 1
let $y as xs:integer := $x*2
return $y
```

Die Variable `$x` hat zwar einen Wert, der zum Typ `xs:integer` passt, wird aber von der statischen Typprüfung nicht zur Zuweisung an `$y` zugelassen, weil ihre Typannotation `xs:decimal` ist.

Prepare/Execute

In Szenarien, in denen dieselbe XQuery-Anfrage mehrfach ausgeführt werden soll, ist zu erwarten, dass die XQuery-Implementierung ein Prepare/Execute-Modell anbietet, bei dem eine Anfrage nur einmal übersetzt (und damit der statischen Analyse unterzogen) wird, aber mehrfach ausgeführt wird (ggf. mit anderen Variablenwerten). Dann lohnt sich ein höherer Aufwand in der statischen Analyse besonders, denn dadurch kann oft der Aufwand während der dynamischen Ausführung reduziert werden.

8.3.2 Dynamische Ausführung

Wenn die statische Analyse keine Fehler festgestellt hat, folgt die dynamische Ausführung, also die eigentliche Auswertung der Anfrage, zu der nun auch die Eingabedaten verwendet werden. Die Eingabedaten, wie sie von den Funktionen `fn:doc` und `fn:collection` geliefert werden, müssen dabei als Instanzen des XQuery-Datenmodells vorliegen. Die damit unter Umständen verbundenen Analyse- und Validierungsschritte werden von XQuery vorausgesetzt, gehören aber nicht zum Verarbeitungsmodell von XQuery. Während der dynamischen Ausführung wird ein *dynamischer Kontext* (Abschnitt 8.3.5) aufgebaut, der zum Beispiel die aktuellen Variablenbindungen enthält. Jedem Ausdruck wird außerdem ein dynamischer Typ zugewiesen, der spezifischer sein kann als der während der statischen Analyse abgeleitete

Dynamischer Kontext

Typ. Während der dynamischen Ausführung kann es – selbst wenn eine statische Typprüfung durchgeführt wurde – zu typbedingten Fehlern kommen, wenn zum Beispiel ein Wert, der einer Typumwandlung unterzogen wird (`cast`), nicht mit dem Zieltyp kompatibel ist.

8.3.3 Serialisierung

Eine XQuery-Anfrage liefert immer eine Sequenz und somit eine Instanz des XML-Datenmodells. Es liegt bei der XQuery-Implementierung, wie diese Instanz des XML-Datenmodells weitergegeben wird. In vielen Fällen wird eine Serialisierung vorgenommen werden, d. h., die Instanz des XML-Datenmodells wird in eine Zeichenkette (z. B. ein XML-Dokument) überführt.

Eine eigene Spezifikation aus der XQuery-Familie [W3C-23] beschreibt, wie durch Serialisierung aus einer Instanz des XQuery-Datenmodells ein XML-Dokument oder ein XML Entity entsteht.

Dazu wird die zu serialisierende Sequenz zunächst transformiert (normalisiert), und zwar in folgenden Schritten:

- Aus einer leeren Sequenz wird eine leere Zeichenkette.
- Alle atomaren Werte der Sequenz werden durch Typumwandlung mittels `xs:string()` in Zeichenketten transformiert.
- Aufeinander folgende Zeichenketten werden miteinander verschmolzen, getrennt durch ein Leerzeichen.
- Jede Zeichenkette wird durch einen Textknoten ersetzt. Jetzt besteht die Sequenz nur noch aus Knoten.
- Jeder Dokumentknoten wird durch seine Kinder ersetzt.
- Ein neuer Dokumentknoten wird erzeugt und alle Einträge der Sequenz werden zu seinen Kindern.

Wenn der entstehende Dokumentknoten genau einen Elementknoten als Kind hat und außerdem nur Kommentarknoten oder Knoten für Verarbeitungsanweisungen, dann wird daraus ein wohlgeformtes XML-Dokument. Anderenfalls wird ein Dokumentfragment erzeugt, das ein wohlgeformtes XML-Dokument ergibt, wenn es in folgendes XML-Dokument »eingepackt« wird (d. h. seine URI als `entity-URI` eingesetzt wird):

```
<!DOCTYPE doc [
    <!ENTITY e SYSTEM "entity-URI">
  ]>
<doc>&e;</doc>
```

Die Serialisierungsspezifikation nennt außerdem einige Parameter, die das Ergebnis der Serialisierung beeinflussen können, wie etwa:

- das Encoding des Ergebnisses
- den Media-Type des Ergebnisses

So kann beispielsweise ein entstehendes XML-Dokument als SVG-Grafik gekennzeichnet werden.

Es ist allerdings nicht zwingend, dass das Ergebnis einer XQuery-Auswertung zu XML wird. Zum einen ist dies nicht immer möglich, z. B. dann, wenn das Ergebnis der XQuery-Auswertung eine Sequenz von Attributknoten ist. Zum anderen wäre es aber auch eine Einschränkung der Möglichkeiten von XQuery. So kann mit einer XQuery-Anfrage ja auch ein HTML-Dokument oder ein anderes Ausgabeformat erzeugt werden. Die Spezifikation der Serialisierung [W3C-23] lässt daher neben der Serialisierungsmethode `xml`, die oben skizziert wurde, weitere Serialisierungsmethoden zu.

8.3.4 Statischer Kontext

Der statische Kontext, der die Auswertung einer XQuery-Anfrage steuert, besteht unter anderem aus folgenden Komponenten:

- *Bekannte Namensräume (»in-scope namespaces«)*
  Bei bekannten Namensräumen handelt es sich um Paare aus Präfix und Namensraum-URI. Diese werden benutzt, um qualifizierte Namen (QNames) innerhalb des XQuery-Moduls aufzulösen. Einige Namensräume sind vordefiniert (Abschnitt 8.2.1). Hinzu kommen noch weitere Namensräume durch die Namensraumdeklarationen im XQuery-Prolog sowie mit einem eingeschränkten Gültigkeitsbereich noch diejenigen Namensraumdeklarationen, die über Konstruktoren für Namensraumknoten (Abschnitt 3.5.4) oder in Namensraumattributen in direkten Elementkonstruktoren (Abschnitt 3.5.1) definiert werden.
- *Vorbelegung für den Elementnamensraum*
  Elementnamen und Typnamen, die kein Namensraumpräfix tragen, gehören zu diesem Namensraum. Dieser kann durch den XQuery-Prolog gesetzt werden (Abschnitt 8.2.1). Ist er nicht gesetzt, so gehören die entsprechenden Elemente und Typen zu keinem Namensraum. Innerhalb von Elementkonstruktoren kann dieser Namensraum durch ein `xmlns`-Attribut oder einen Namensraumknoten umdefiniert werden.

- *Vorbelegung für den Funktionsnamensraum*
  Funktionsnamen, die kein Namensraumpräfix haben, gehören zu diesem Namensraum, der mit dem Namensraum der XQuery-Funktionen vorbesetzt ist, aber im XQuery-Prolog umdefiniert werden kann (Abschnitt 8.2.1).
- *Schemadefinitionen (»in-scope schema definitions«)*
  Schemadefinitionen beinhalten alle Typdefinitionen, Element- und Attributdefinitionen, die zum Auswertungszeitpunkt bekannt sind. Das sind zum einen die Typen aus XML Schema und die Typen aus XQuery (Abschnitt 3.4), zum anderen die Typen, Elemente und Attribute, die in importierten Schemata (Abschnitt 8.2.2) definiert wurden.
- *Variablendefinitionen (»in-scope variables«)*
  Dies sind die Variablen, die in einem XQuery-Ausdruck verwendet werden dürfen, mit ihrem Typ. Dazu gehören die im XQuery-Prolog definierten Variablen (Abschnitt 8.2.3) und die in importierten Modulen definierten Variablen, aber auch (mit eingeschränktem Gültigkeitsbereich) die Variablen, die in XQuery-Anweisungen wie z. B. FLWOR-Ausdrücken gebunden oder in Parameterleisten von Funktionen definiert werden.
- *Funktionen (»in-scope functions«)*
  Dies sind alle Funktionen, die in einem XQuery-Ausdruck aufgerufen werden dürfen. Zunächst gehören dazu alle XQuery-Funktionen (Kapitel 7) und die Konstruktoren für alle bekannten Typen (Abschnitt 3.2.1). Hinzu kommen alle Funktionen, die in importierten Modulen definiert wurden, und alle Funktionen, die im XQuery-Prolog definiert werden.
- *Vorbelegung für die Sortierordnung*
  Dies ist üblicherweise die Unicode-Codepoint-Sortierordnung, dies kann aber durch den XQuery-Prolog verändert werden (Abschnitt 8.2.6).
- *Validierungsmodus*
  Dieser Modus steuert unter anderem, wie neu konstruierte Elemente validiert werden. Initial hat er den Wert `lax`, kann aber im XQuery-Prolog verändert werden (Abschnitt 8.2.4). Mit eingeschränktem Wirkungsbereich kann er auch durch eine `validate`-Anweisung (Abschnitt 3.8.2) überschrieben werden.
- *Behandlung von begrenzendem Leerraum*
  Begrenzender Leerraum wird normalerweise unterdrückt (`strip`). Dies kann jedoch durch die Angabe von `declare xmlspace preserve` (Abschnitt 8.2.5) im XQuery-Prolog geändert werden.

Die XQuery-Spezifikation lässt es zu, dass eine XQuery-Implementierung die initialen Werte des statischen Kontexts erweitert (also zum Beispiel weitere vordefinierte Namensraumpräfixe hinzufügt) oder ändert.

8.3.5 Dynamischer Kontext

Analog zum statischen Kontext gibt es auch einen dynamischen Kontext, der aber erst während der dynamischen Ausführung existiert und durch die vorgefundenen Daten beeinflusst wird. Zum dynamischen Kontext gehören zum Beispiel folgende Komponenten:

- *Der Fokus (»focus«)*
  Der so genannte Fokus besteht aus mehreren Komponenten. Dazu gehört der aktuell betrachtete Eintrag (»context item«), das ist der Eintrag der gerade betrachteten Sequenz, der von dem Ausdruck ».« geliefert wird. Hinzu kommt die Eintragsposition (»context position«), also das Ergebnis des Funktionsaufrufs `fn:position()`. Die letzte Komponente ist die Anzahl der Einträge in der gerade betrachteten Sequenz (»context size«), also das Ergebnis des Funktionsaufrufs `fn:last()`.
- *Die dynamischen Variablen*
  Bei den dynamischen Variablen handelt es sich um eine Menge von Paaren aus Variablenname und aktuellem Wert.
- *Datum und Uhrzeit*
  Dies ist ein von der XQuery-Implementierung gewählter Zeitpunkt, der aber innerhalb der Zeitspanne liegen muss, in der die aktuelle Auswertung geschieht. Auf diese Komponente des dynamischen Kontexts kann man mit den Funktionen `fn:current-date()`, `fn:current-time()` und `fn:current-dateTime()` zugreifen (Abschnitt 7.3.1). Mehrfache Aufrufe dieser Funktionen innerhalb einer XQuery-Anfrage liefern jeweils denselben Wert.
- *Implizite Zeitzone*
  Dies ist die Zeitzone, die benutzt wird, wenn in einem Vergleich oder einer anderen Operation ein Wert vom Typ `xs:date`, `xs:time` oder `xs:dateTime` auftritt, der keine Zeitzone hat.

8.4 Zusammenfassung

XQuery besitzt ein Modulkonzept, das es erlaubt, XQuery-Bausteine wie zum Beispiel Funktionen wiederverwendbar zu realisieren. Module können – wie im ersten Abschnitt ausführlich erläutert – von