

V-DOM: Dokumentenmodell für XML-basierte World-Wide-Web-Anwendungen

Martin Kempa

Institut für Informationssysteme, Medizinische Universität zu Lübeck,
Osterweide 8, 23562 Lübeck, Germany, kempa@ifis.mu-luebeck.de

Zusammenfassung

Eine der wichtigsten Eigenschaften von Anwendungen für das World-Wide-Web ist die Behandlung und Verarbeitung von Dokumenten im Programm. Mittels der standardisierten Extensible-Markup-Language können Auszeichnungssprachen zum Austausch hierarchisch strukturierter Texte und Daten definiert werden. Die baumartige Struktur der Dokumente wird ausgenutzt, um diese intern im Programm als baumartige Objekthierarchie darzustellen, wozu das Dokument-Objektmodell definiert wurde.

Dieses Papier präsentiert V-DOM, ein Objektmodell, das jeder Auszeichnungssprache eine Objektschnittstelle zuordnet, die über die standardisierten Anforderungen des Dokument-Objektmodells hinausgeht. Es wird die Möglichkeit zur Behandlung von Teildokumenten eingeführt und durch spezielle Methoden die statische Gültigkeit von Dokumenten und Teildokumenten bereits zur Zeit der Programmübersetzung sichergestellt.

1 Einleitung

Sämtliche Anwendungen, die die Vorteile des World-Wide-Web (WWW) nutzen, zeichnen sich durch eine intensive Verarbeitung von Dokumenten aus. Zur Zeit handelt es sich dabei hauptsächlich um Dokumente der Hypertext-Markup-Language (HTML) [RLHJ97], aber im zunehmenden Maße auch um Dokumente in XML, der Extensible-Markup-Language [W3 98b], die als Nachfolger für HTML angesehen wird. Analog zu HTML werden in XML die Strukturinformationen des Dokumentes durch Auszeichnungen in eine sequenzialisierte Repräsentation gebracht. Diese sind allerdings im Gegensatz zu HTML nicht fest vordefiniert, statt dessen können für verschiedene Einsatzgebiete unterschiedliche Auszeichnungselemente definiert werden. Aus diesem Grund ist XML nicht eine Auszeichnungssprache, sondern eine Metasprache zur Definition von Auszeichnungssprachen. Dementsprechend wurde auch schon mit XHTML [W3 99a] eine XML-konforme Version von HTML definiert.

Da für WWW-Anwendungen die Verarbeitung von Dokumenten nun so wichtig ist, liegt die Forderung für eine Implementierungsumgebung nahe, die eine gute Unterstützung zur Definition und Manipulation von Datenstrukturen für Dokumente vorsieht. Dies ist in den Skriptsprachen PERL [WS91] oder TCL [Ous94], die als gängige Implementierungssprachen für WWW-Anwendungen Verwendung finden, weder realisiert noch vorgesehen; hier erfolgt eine Dokumentenverarbeitung auf Zeichenkettenebene.

Einen ersten Schritt hin zu einem strukturierten Ansatz für die Programmierung von WWW-Anwendungen, stellt das Dokument-Objektmodell (DOM) [W3 98a] dar. Dabei handelt es sich um eine plattform- und sprachunabhängige Schnittstellenbeschreibung, die einen dynamischen Zugriff

auf den Inhalt, die Struktur und das Layout eines XML- oder HTML-Dokumentes aus einer Programmier- oder Skriptsprache heraus erlaubt. Dazu wird ein Dokument als Baumstruktur von Objekten repräsentiert, wobei der eigentliche Inhalt des Dokumentes in den Blättern steht und die Strukturinformationen die inneren Knoten des Objektbaumes bilden. Wird dieser Baum durch ein Programm verändert, hat dies entsprechende Konsequenzen auf das durch ihn repräsentierte Dokument.

Die Schnittstelle, die durch das DOM standardisiert wurde, ist für beliebige XML-Dokumente vorgesehen, und erfährt lediglich für HTML eine Spezialisierung. Damit erfolgt aber für den Großteil der in XML definierten Auszeichnungssprachen nur eine sehr allgemeine Unterstützung, die wesentliche Möglichkeiten zur Überprüfung der korrekten Verarbeitung von XML-Dokumenten bereits zur Zeit der Programmübersetzung nicht ausnutzt. Eine Verbesserung kann man nun dadurch erzielen, daß die Sprachbeschreibung einer speziellen XML-Auszeichnungssprache verwendet wird, um eine spezialisierte Objektschnittstelle zu erzeugen. Diese soll nun Methoden zur Verfügung stellen, die einerseits eine Überprüfung eines Teils der Gültigkeit der Dokumente bereits zur Übersetzungszeit der WWW-Anwendung erlauben, und andererseits die Verarbeitung von Teildokumenten ermöglichen, was durch eine Erweiterung des standardisierten DOM erreicht werden kann.

2 Extensible-Markup-Language und Dokument-Objektmodell

Dieser Abschnitt erläutert die grundlegenden Konzepte und die Syntax von XML sowie die Idee des DOM. Eine umfassendere Einführung in XML ist in [Bra98a] zu finden, und technische Details werden in [Bra98b] gut erklärt.

2.1 Elemente und Attribute

Die Hauptbestandteile eines Dokumentes einer in XML definierten Auszeichnungssprache sind *Elemente*, die durch *Start-Tags* und *End-Tags* gekennzeichnet sind. Das Element vom Typ `practice` beginnt zum Beispiel mit dem Start-Tag `<practice>` und endet mit dem End-Tag `</practice>`. Um Eigenschaften für Elemente festzulegen, dienen die für jeden Elementtyp deklarierten *Attribute*. Deren Werte können für jedes Element durch das Start-Tag spezifiziert werden. Im Beispiel `<practice date="November">` wird der Wert des Attributs `date` im Element `practice` auf `November` gesetzt. Als Werte für Attribute sind Elemente und *Zeichen-Daten* zugelassen.

Die Auszeichnungssprache selbst wird mittels der *Dokumenttyp-Definition* (DTD) definiert, in der die Inhalte sämtlicher Elemente spezifiziert werden, die wiederum aus Elementen bestehen können. Der Inhalt der Elemente eines Elementtyps wird durch einen regulären Ausdruck über Elementtypen, dem *Inhaltsmodell* („*content model*“), beschrieben, in dem festgelegt wird, in welcher Reihenfolge andere Elemente oder Zeichen-Daten auftreten müssen. Für die Spezifikation eines Inhaltsmodells können Elemente als Folge (`a, b`) oder als Auswahlliste (`a|b`) gruppiert werden. Für diese *Gruppen* und Elemente kann zusätzlich die Anzahl des Auftretens durch die Operatoren (`?`, `+` und `*`) vorgegeben werden. Ebenfalls werden in der DTD die Attribute für jeden Elementtyp deklariert, die syntaktische Form ihrer Werte und die Vorgabe-Werte für unspezifizierte Attribute in konkreten Elementen festgelegt.

Ein kleines Beispiel für eine DTD zeigt die Abbildung 1 mit der von uns definierten Auszeichnungssprache PRACTICE, die eine Auszeichnungssprache für Übungs- oder Aufgabenblätter darstellt. Zeile 1 legt für jede *Dokumenteninstanz* dieser Auszeichnungssprache fest, daß sie durch ein Wurzel-Element vom Elementtyp `practice`, das in Zeile 3 definiert ist, gebildet wird, für den zusätzlich das Attribut `date` in Zeile 8 deklariert wurde. Es ist vom Typ `CDATA` und wurde ohne Standardwert deklariert. Stattdessen wird durch das Schlüsselwort `#REQUIRED` verlangt, daß das Attribut für jedes Element vom Typ `practice` spezifiziert wird. Jedes Element `practice` muß aus einem Titel vom Elementtyp `title`, der in Zeile 4 definiert ist, und mindestens einer Aufgabe

<!DOCTYPE practice [1
<!ENTITY % flow "(paragraph list)+"	2
<!ELEMENT practice (title, exercise+)	3
<!ELEMENT title (#PCDATA)	4
<!ELEMENT exercise (title, %flow;)	5
<!ELEMENT paragraph (#PCDATA)	6
<!ELEMENT list (exercise+)	7
<!ATTLIST practice date CDATA #REQUIRED	8
<!ATTLIST exercise points CDATA #IMPLIED	9
]>	10

Abbildung 1: Eine Auszeichnungssprache wird durch eine DTD definiert.

vom Typ `exercise` aus Zeile 5 bestehen, die wiederum aus Text vom Typ `paragraph` (Zeile 6) und Listen von Teilaufgaben, durch den Typ `list` (Zeile 7) markiert, bestehen kann. Für jede Aufgabe können Punkte mittels des Attributs `points` in Zeile 9 vergeben werden. Mit dem Schlüsselwort `#IMPLIED` werden optionale Attribute gekennzeichnet.

2.2 Entities und Entity-Referenzen

Eine Möglichkeit zur Modularisierung besteht in XML durch *Entities*. Diese werden in der DTD definiert und bestehen aus dem Namen des Entities und einem Ersetzungstext, der sich *intern* in der DTD oder *extern* in einer Datei befinden kann. Eine Art von Entities wird innerhalb der DTD verwendet und mit *Parameter-Entity* bezeichnet. Eine Referenz dieses Entities erkennt man durch den Referenznamen, der durch ein beginnendes `%` und ein endendes `;` eingeschlossen ist, wodurch der Ersetzungstext an diese Stelle eingesetzt wird. Parameter-Entities dienen zur Abkürzung innerhalb einer DTD und ermöglichen die Aufteilung dieser in verschiedene Dateien. Ein Beispiel für ein internes Parameter-Entity zeigt die Zeile 2 in Abbildung 1. Dort erfolgt die Deklaration des Entities `flow`, welches in der Zeile 5 referenziert wird.

Eine andere Art von Entities wird mit *allgemeinem Entity* bezeichnet und dessen Referenzen treten ausschließlich im Dokument auf. Die Syntax unterscheidet sich etwas von Parameter-Entity-Referenzen, indem das Zeichen `&` statt `%` verwendet wird. Auf diese Art können Entities wie Makros für Teile des Inhaltes eines Dokumentes verwendet werden oder so auch zur Aufteilung eines Dokumentes in mehrere Dateien dienen. Obwohl generelle Entities nur in Dokumenten auftauchen können, müssen sie in der DTD definiert werden.

Die Abbildung 2 zeigt ein Dokument der oben definierten Auszeichnungssprache `PRACTICE`. In der Zeile 1 wird das generelle Entity `db`s definiert, für die in den Zeilen 4 und 8 Referenzen im Titel des Aufgabenblattes und der Aufgabe auftreten. In den Zeilen 3 bis 5 wird der Titel des Aufgabenblattes mit dem Element `title` ausgezeichnet. Die einzige Aufgabe des Dokumentes wird durch die Zeilen 6 bis 13 mittels des dafür vorgesehenen Elements `exercise` beschrieben und wird mit 2 Punkten bewertet, was das Attribut `points` festlegt. In den Zeilen 7 bis 9 wird die Aufgabe mit einem Titel versehen, während der eigentliche Inhalt der Aufgabe in den Zeilen 10 bis 12 zu finden ist.

2.3 Wohlgeformtheit und Gültigkeit

Für Dokumente einer in XML spezifizierten Auszeichnungssprache ist die Eigenschaft *wohlgeformt* („*well-formed*“) definiert. Damit wird die syntaktische Korrektheit eines Dokumentes bezeichnet. Jedes Entity, dessen Referenz im Dokument auftritt, muß in der DTD definiert sein. Weiterhin muß die logische Struktur korrekt sein und einer klammerartigen Ordnung folgen. Damit ist gemeint, daß

```

<!ENTITY & dbs "Datenbanksysteme" > 1
...
<practice date="November"> 2
  <title> 3
    1. Aufgabenblatt zu &dbs; 4
  </title> 5
  <exercise points="2"> 6
    <title> 7
      Grundlagen &dbs; 8
    </title> 9
    <paragraph> 10
      Was verstehen Sie unter Persistenz? 11
    </paragraph> 12
  </exercise> 13
</practice> 14

```

Abbildung 2: Ein Dokument der Auszeichnungssprache PRACTICE.

zu jedem Start-Tag ein korrespondierendes End-Tag existiert und beide sich in dem selben Ersetzungstext eines Entities befinden.

Eine strengere Eigenschaft als wohlgeformt wird durch *gültig* („valid“) definiert. Damit wird gefordert, daß sämtliche Deklarationen der DTD eingehalten werden. Beispielsweise muß der Inhalt eines Elements dem Inhaltsmodell des Elementtyps entsprechen sowie dessen Attributwerte mit den deklarierten Attributtypen übereinstimmen.

2.4 Dokument-Objektmodell (DOM)

Die Definition des DOM hat eine Standardisierung von Programmierschnittstellen für XML-Anwendungen zum Ziel. Es wird die logische Struktur des Dokumentes, die Art des Zugriffs auf dessen Daten und der Weg der Datenmanipulation definiert. Die logische Struktur eines Dokumentes im DOM wird mit *Strukturmodell* („structure model“) bezeichnet und entspricht einer baumartigen Repräsentation des Dokumentes. Ein wesentliches Merkmal ist die Darstellung durch Objekte, die durch einen Zustand und ein Verhalten bestimmt sind.

Die Repräsentation des Dokumentes aus Abbildung 2 im DOM ist in Abbildung 3 dargestellt. Das Wurzel-Element steht für das Element `practice` und verweist auf die beiden Kinderknoten,

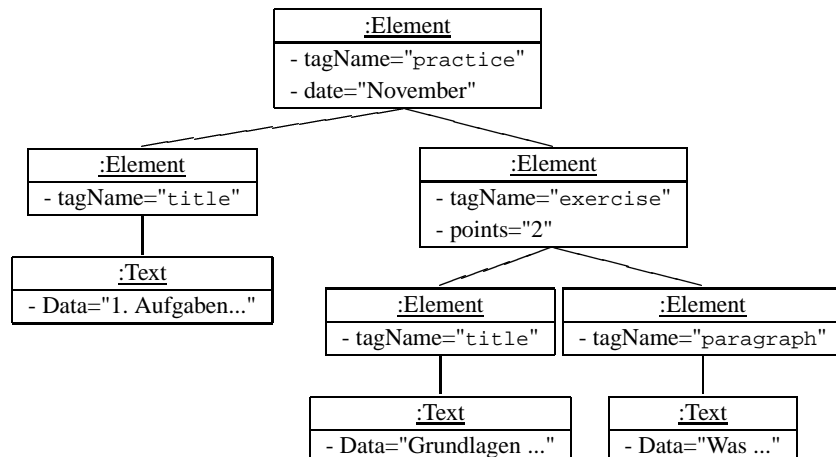


Abbildung 3: Das Dokument als Objekthierarchie des DOM.

die für die Elemente `title` und `exercise` im Dokument stehen, die den Inhalt von `practice`

bilden. Analog wird für sämtliche, verschachtelten Elemente im Dokument eine korrespondierende, baumartige Darstellung von Objekten erzeugt.

3 XML-Anwendungen

Das Einsatzgebiet für XML und damit verbunden von Anwendungen, die XML verarbeiten werden, erscheint vielfältig. Zunächst einmal ist eine Verwendung von XML in der klassischen Dokumentenverarbeitung vorgesehen. Diese umfaßt nach [Neu99] Anwendungen zur Transformation, Konvertierung, Formatierung und von Abfragewerkzeugen. Bei der Anwendung *Transformierer* wird aus einem XML-Dokument ein neues, verändertes XML-Dokument erzeugt, das sich in der Struktur vom ursprünglichen Dokument dahingehend unterscheidet, daß Elemente gelöscht, verschoben oder eingefügt wurden. Ein *Konvertierer* leistet dagegen die Umwandlung eines Eingabedokumentes in ein XML-fremdes Format, wie z. B. Postscript. Schaltet man die beiden Anwendungen hintereinander, läßt also einer Transformation eine Konvertierung folgen, kann man von einem *Formatierer* sprechen, falls aus einem XML-Dokument eine Repräsentation erstellt wird, die sich zum Lesen, Drucken oder Veröffentlichen eignet. Die Anfrage ist ebenfalls eine wichtige Funktionalität für fast alle Anwendungen, da mit dieser die Suche und Extraktion von Dokumententeilen, die einem bestimmten Muster entsprechen, vorgenommen werden können.

Neben den klassischen Anwendungen aus der Dokumentenverarbeitung spielt XML in zunehmendem Maße für den Datenaustausch eine Rolle. WWW-Anwendungen des Internets sind dafür das populärste Beispiel, indem eine Verdrängung von HTML zugunsten XML als Datenaustauschformat stattfindet. Zusätzlich wird XML in vielen Bereichen als Datenaustauschformat eingesetzt, in denen HTML als Austauschformat keine Rolle spielt, wo es beliebige andere Formate ersetzt. Anwendung für das WWW gibt es eine Vielzahl mit einem weiten Spektrum von Einsatzgebieten, aber besonders der Bereich des *Electronic-Commerce* macht sich für den Einsatz von XML stark. Neben diesem Bereich scheint sich XML, durch seine Möglichkeit jede Art von Daten zu repräsentieren, als Datenformat in viele Gebiete der Informatik auszubreiten. So gibt es inzwischen mehrere Ansätze wie in [BI98] und [SY98] für UML-Werkzeuge, die XML als Beschreibungssprache für Software-Modelle verwenden.

3.1 XML-Parser

Eine Anwendung, die XML-Dokumente verarbeitet, hat keine Information über die syntaktische oder technische Repräsentation eines Dokumentes, sondern nur eine logische Sicht auf die Dokumentenstruktur. Ein Zugriff auf ein XML-Dokument geschieht mittels eines XML-Parsers, womit ein Algorithmus gemeint ist, der ein XML-Dokument parsiert und mittels einer Programmschnittstelle der Applikation zur Verfügung stellt. In der Entwicklung von XML-Parsern haben sich zwei verschiedene Arten dieser Programmschnittstellen herausgebildet, die baumbasierte und die ereignisbasierte Programmschnittstelle.

Bei der baumbasierten Programmschnittstelle wird ein XML-Dokument vollständig parsiert und der Anwendung in der baumähnlichen Datenstruktur des DOM übergeben. Neben dem Vorteil dieser Programmschnittstelle, der in den vielen vordefinierten Methoden liegt, die die Bearbeitung von Dokumenten erheblich vereinfachen, kann man darin auch einen Nachteil dieses Ansatzes sehen, da eine vollständige Kopie des Dokumentes im Hauptspeicher gehalten werden muß, was für eine Reihe von Anwendungen nicht notwendig ist. Es sind sogar XML-Dokumente denkbar, die von der Größe her nicht in den Hauptspeicher passen würden.

Diese Problematik wird durch die ereignisbasierten Parser aufgegriffen [Meg98]. Der Prozeß des Parsierens wird in Ereignisse sequenzialisiert, die beispielsweise durch ein Start-Tag, ein End-Tag oder eine Entity-Referenz ausgelöst werden können. Dem Parser wird für jede Art dieser Ereignisse ein spezieller Algorithmus, ein *Handler*, mitgegeben, der durch diese angestoßen wird und die Daten des Dokumentes verarbeiten kann. Da durch eine ereignisbasierte Schnittstelle ebenfalls eine

baumartige Datenstruktur aufgebaut werden kann, steht damit eine allgemeinere Programmschnittstelle zur Verfügung.

Herkömmlichen XML-Parsern ist die allgemeine Vorgehensweise gemein, daß sie in einem ersten Durchlauf das Dokument auf die Eigenschaft wohlgeformt überprüfen und in einem zweiten die Gültigkeit erkennen, falls dies überhaupt erfolgt¹. Diese Mächtigkeit beliebige XML-Dokumente zu verarbeiten, ist nur für wenige Anwendungen, wie z. B. für allgemeine XML-Browser, wirklich notwendig und sinnvoll. Wir denken, daß ein großer Teil von WWW-Anwendungen nur wenige aber feste Auszeichnungssprachen benutzt, für die die Verwendung eines allgemeinen XML-Parser zu unflexibel ist, und sich ein spezialisierter Parser geradezu anbietet. Verwendet man für eine spezielle XML-Sprache nicht einen allgemeinen XML-Parser, sondern einen auf diese Auszeichnungssprache entsprechend abgestimmten Parser, kann man die Eigenschaften wohlgeformt und statische Gültigkeit in einem Durchlauf sicherstellen. Ein Beispiel für eine solche Anwendung ist der klassische Browser für HTML-Dokumente.

4 Die Auszeichnungssprache impliziert eine Objektschnittstelle

Mit *Validating*-DOM (V-DOM) präsentieren wir eine Erweiterung des DOM, um wesentliche Einschränkungen dessen zu beheben. Die Idee ist, für jede durch eine DTD definierte Auszeichnungssprache eine typsichere Objektschnittstelle zu generieren, die im Gegensatz zu [Alt99] eine Erweiterung des DOM darstellt. Die strikte Typisierung soll dabei gerade die *statische Gültigkeit*² der in den WWW-Anwendungen zu verarbeitenden Dokumenten oder Teildokumenten zur Übersetzungszeit sicherstellen. Damit können Fehler in den Applikationen im Vergleich zum DOM, wo diese Fehler in Dokumenten erst zur Ablaufzeit erkannt werden können, wesentlich früher ermittelt werden. Die Einschränkung zur allgemeinen Gültigkeit besteht hauptsächlich in der Überprüfung der Anzahl des Auftretens von Elementen (+, *), bei der fehlerhafte Löschoptionen im Dokument erst zur Laufzeit erkannt werden können.

Das DOM unterstützt als Objektschnittstelle sämtlich XML-Dokumente, was eine sehr allgemeine Definition erfordert. Jedes Dokument besteht aus verschachtelten Elementen, die wiederum Attribute beinhalten können, wie in Abbildung 3 zu sehen ist. Diese Möglichkeit, jedes Dokument auf gleiche Weise anzusprechen, entwickelt sich allerdings zum Nachteil, wenn man die Verarbeitung von nicht gültigen, also fehlerhaften, Dokumenten unterbinden möchte. Es kann nämlich ohne weiteres ein Dokument erstellt werden, was nicht den Anforderungen der zugrundeliegenden DTD entspricht. Diese Dokumente, die der Gültigkeit widersprechen, können zwar zur Laufzeit aufgespürt werden, erfordern dort aber stets eine aufwendige Analyse, obwohl ein großer Teil dieser Merkmale durchaus statisch überprüfbar ist.

4.1 Die Objektschnittstelle

Die Idee ist nun die Schnittstelle so zu erweitern, daß diese statischen Merkmale der Gültigkeit bereits zur Zeit der Übersetzung sichergestellt werden. Dazu wird für jede Auszeichnungssprache die ihr zugrundeliegende DTD herangezogen und für diese eine erweiterte Schnittstelle erzeugt, die durch spezielle Methoden sowohl die Inhaltsmodelle der Elementtypen sicherstellen kann, als auch nur korrekte Attribute und Attributwerte zuläßt. Das sich solch eine Erweiterung gut ins DOM integriert, zeigt sich daran, daß bereits für HTML eine solche Spezialisierung im DOM existiert, die allerdings nur spezielle Methoden für Attribute kennt. Auch dort wird wie in unserem Ansatz aus jedem Elementtyp, der in der DTD definiert wurde, eine spezielle Schnittstelle spezifiziert, die einer Spezialisierung der DOM-Schnittstelle `Element` entspricht.

¹Speziell bei ereignisbasierten Programmierschnittstellen wird die Gültigkeit meist nicht überprüft.

²Statische Gültigkeit bezeichnet hier die Merkmale der Gültigkeit, die zur Übersetzungszeit sichergestellt werden können.

In unserem Beispiel wird also für den Elementtyp `practice` aus Abbildung 1 Zeile 3 eine Schnittstelle `PracticeElement` generiert, die Abbildung 4 zeigt.³ Für das Attribut `date` erfolgt

```

interface PracticeElement : Element {           1
  // attributes                                 2
  attribute DOMString          date;           3

  // elements                                   4
  attribute TitleElement      title;           5

  readonly attribute unsigned long length;     6
  void          add(in ExerciseElement exercise); 7
  void          remove(in unsigned long index);   8
  ExerciseElement item(in unsigned long index);  9
}; // PracticeElement                          10

```

Abbildung 4: Die erzeugte Schnittstelle `PracticeElement`.

eine triviale Abbildung in Zeile 3. Für das Inhaltsmodell werden das Attribut und die Methoden in den Zeilen 5 bis 9 deklariert. In Zeile 5 ist für das laut Inhaltsmodell obligatorische Element `title` ein IDL-Attribut gleichen Namens vom Schnittstellentyp `TitleElement` vorgesehen, der analog aus der DTD für den Elementtyp `title` generiert wurde. Für die darauf folgenden Elemente vom Elementtyp `exercise` werden in den Zeilen 6 bis 9 das Attribut `length` und die Methoden `add`, `remove` und `item` bekanntgemacht.

Betrachten wir nun unser Dokument aus Abbildung 2 in der erzeugten Schnittstelle des V-DOM, so erhalten wir eine Instanz, die durch die Abbildung 5 dargestellt ist. Durch die Spezialisierung der

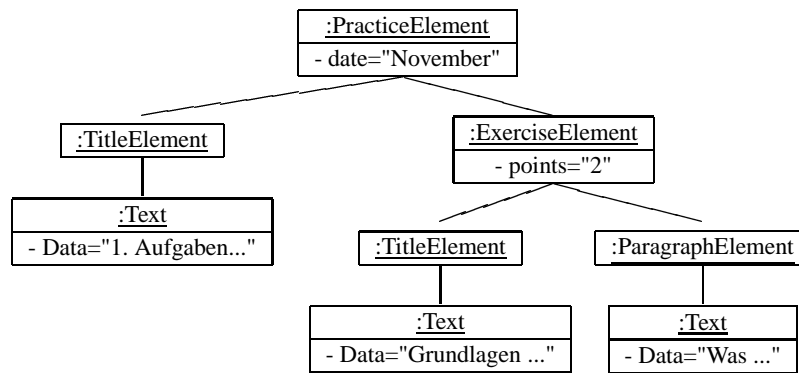


Abbildung 5: Das Dokument als Objekthierarchie im V-DOM.

Objektschnittstellen ist nun im Gegensatz zum DOM die statische Gültigkeit für jede Objekthierarchie sichergestellt.

4.2 Die Dokumenten-Schnittstelle

Das Konzept zum Einlesen von Dokumenten basiert darauf, daß jede Objektschnittstelle, die mit einem Elementtyp der Auszeichnungssprache korrespondiert, einen eigenen Algorithmus zum Einlesen genau dieser Elemente als Methode realisiert. Wir erhalten damit neben der Möglichkeit ganze Dokumente in die Objekthierarchie zu überführen, was gerade bei großen Dokumenten einen gewissen Aufwand bedeutet, die Fähigkeit Teildokumente einzulesen, was eine wesentliche Erweiterung und Vereinfachung des DOM darstellt.

³Analog zum DOM notieren wir die Schnittstelle in IDL, um für unsere Konvertierung die Unabhängigkeit von einer Programmiersprache zu betonen.

Bereits für die Erzeugung der Objektschnittstelle muß die DTD der Auszeichnungssprache verarbeitet werden, wodurch es nahe liegt, diese Informationen ebenfalls auch zur Generierung eines spezialisierten Parsers, der lediglich die Dokumente dieser Auszeichnungssprache verarbeitet, zu nutzen. Durch den einfachen Aufbau der Dokumente in XML kann nach [Neu99] eine Realisierung im Stile von Recursive-Descent-Parsern erfolgen. Dabei wird für jede syntaktische Komponente, hier vor allem die Elementtypen, eine Funktion definiert, die genau diese Komponente parsiert. Da Komponenten wiederum aus anderen Komponenten bestehen können, werden diese Funktionen ineinander verschachtelt aufgerufen. Es ist offensichtlich, wie gut sich diese Arbeitsweise in unser obiges Konzept einfügt.

Man kann diese Idee sogar weiter bis hin zur Implementierung von allgemeinen XML-Parsern verfolgen. Im Gegensatz zu herkömmlichen Parsern, die erst das gesamte Dokument einlesen und dann in einem zweiten Schritt die Korrektheit zur angegebenen DTD überprüfen, kann nun anhand der DTD für jede dadurch definierte Auszeichnungssprache ein spezieller Parser generiert werden. Der Vorteil bei diesem Vorgehen liegt offensichtlich darin, daß für mehrere Dokumente einer Auszeichnungssprache die zugrundeliegende DTD nur einmal eingelesen und ausgewertet werden muß, was gerade bei umfangreichen Sprachdefinitionen Rechenzeit einspart.

4.3 Implementierung eines V-DOM-Generators

Als Prototyp steht inzwischen eine Implementierung zur Verfügung, die unsere Idee realisiert. Wie in

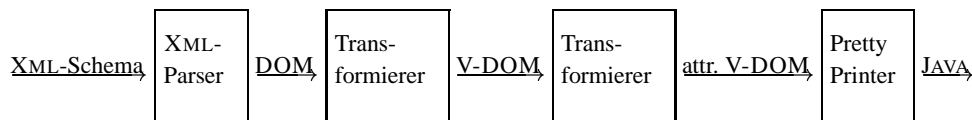


Abbildung 6: Phasen der V-DOM-Generator-Implementierung.

Abbildung 6 zu sehen ist, wird mittels eines XML-Parsers in einer ersten Phase die als XML-Schema-Dokument ([W3 99b]) vorliegende DTD einer XML-basierten Sprache eingelesen. Wir haben uns für XML-Schema entschieden, um einmal die schon vorhandenen allgemeinen XML-Parser nutzen zu können, und zweitens auch das neue Objektmodell selbst in der Implementierung zu verwenden. Es entsteht im Vergleich zur DTD keine Einschränkung sondern eher eine Bereicherung in der Ausdruckskraft, obwohl unser Prototyp bisher nur die Möglichkeiten der DTD unterstützt. Auch ließe sich bedingt durch die stärkeren Typisierungsmöglichkeiten in XML-Schema der V-DOM-Ansatz gut entsprechend übertragen.

Der XML-Parser erzeugt die interne Repräsentation des Dokuments im DOM, was zur Zeit leider noch als Zwischenschritt notwendig ist, um den bereits existierenden Parser zu nutzen. Für eine verbesserte Version unseres Prototypen ist eine erste Phase geplant, die über eine ereignisbasierte Schnittstelle direkt die Repräsentation im V-DOM erzeugt. Abbildung 7 zeigt unsere Beispielsprache PRACTICE in der Syntax einer XML-Schema-Definition. Elementtypen werden durch das Element `element` deklariert und deren Inhaltsmodell durch das Element `type` definiert. Die Anzahl des Auftretens von Elementen wird hier durch die Attribute `minOccurs` und `maxOccurs` wie in den Zeilen 2, 9, 10 und 26 eingeschränkt. Statt der Entities kann man hier auf benannte Gruppen mittels des Elements `group` wie in den Zeilen 2 bis 5 zurückgreifen. Eine Alternative ist in Zeile 2 durch das Setzen des Attributs `order` auf den Wert `choice` zu sehen, ansonsten gilt für das Inhaltsmodell die Folge. Um Attribute bekanntzumachen, wird das Element `attribute` wie in den Zeilen 10 und 18 verwendet.

In einem zweiten Schritt wird diese Repräsentation in die V-DOM-Implementierung für XML-Schema transformiert. Unsere Implementierung selbst ist damit ein Beispiel für die Programmierung mit V-DOM. In einem 3. Schritt wird mit Hilfe einer Attributierung der Objekthierarchie die eigentliche Transformation nach JAVA vorbereitet, die dann in der 4. Phase durch einen „Pretty-Printer“

<schema targetNamespace="practice">	1
<group name="flow" order="choice" minOccurs="1" maxOccurs="*">	2
<element ref="paragraph"/>	3
<element ref="list"/>	4
</group>	5
<element name="practice">	6
<type>	7
<element ref="title" />	8
<element ref="exercise" minOccurs="1" maxOccurs="*" />	9
<attribute name="date" type="string" minOccurs="1" />	10
</type>	11
</element>	12
<element name="title" type="PCDATA"/>	13
<element name="exercise">	14
<type>	15
<element ref="title"/>	16
<group ref="flow" />	17
<attribute name="points" type="string" />	18
</type>	19
</element>	20
<element name="paragraph">	21
<type content="mixed" />	22
</element>	23
<element name="list">	24
<type>	25
<element ref="exercise" minOccurs="1" maxOccurs="*" />	26
</type>	27
</element>	28
</schema>	29

Abbildung 7: PRACTICE als XML-Schema-Definition.

ausgegeben wird. Wir haben uns hier für die Ausgabe in JAVA anstelle von IDL entschieden, um die Ausgabe problemlos in Anwendungen zu nutzen.

5 Ausblick

Dieses Papier, eine ausführliche Fassung von [Kem00], präsentiert das Objektmodell V-DOM, das zu jeder XML-Auszeichnungssprache eine voll funktionsfähige Objektschnittstelle definiert, und die prototypische Implementierung eines Generators dieser Schnittstellen vorstellt. Die Objektschnittstelle zeichnet sich dadurch aus, daß nur statisch gültige Dokumente verarbeitet werden können, was durch eine strikte Typisierung erreicht wird, so daß Fehler bereits zur Übersetzungszeit der WWW-Anwendung erkannt werden. Um auch die Verarbeitung von Teildokumenten zu erlauben, wie es häufig bei Anwendungen mit Datenbankzugriff notwendig ist, verfügt jede Objektschnittstelle über einen eigenen Parser, der Teildokumente in die Objekthierarchie überführt. Eine Konvertierung einer Objekthierarchie des DOM in das V-DOM eröffnet die Möglichkeit, bereits existierende DOM-basierte Anwendungen zu nutzen.

Die Vorteile dieser Vorgehensweise wurde inzwischen erkannt und es gibt von mehreren Herstellern (Microsoft, Oracle, ObjectSpace und Shinka) inzwischen erste Implementierungen. Nach unserer Ansicht unterstützen diese weder das bereits standardisierte DOM noch eine Dokumenten-

Schnittstelle, um Dokumente in die Objekthierarchie zu überführen. Leider wurde im Gegensatz zum DOM bisher eine gemeinsame Standardisierung durch das World-Wide-Web-Konsortium (W3C) versäumt, weshalb diese möglichst bald nachgeholt werden sollte.

Wir denken, daß mit V-DOM die Basis eines Programmiermodells für WWW-Anwendungen existiert, welches deren korrekte Implementierungen wesentlich erleichtert. Gerade im Hinblick auf kleine mobile Anwender halten wir eine Verbesserung der Korrektheit in Dokumenten für eine wichtige Zielsetzung.

Danksagung: *Der Autor dankt V. Linnemann für wertvolle Hinweise zur Verbesserung der Arbeit.*

Literatur

- [Alt99] Chuck Altman. *Using Dynamic XML, Developer's Guide*. ObjectSpace, Inc., Dallas, Texas, third edition, 1997-1999. Software version 1.1.
- [BI98] Steve Brodsky and Sridhar Iyengar. XML Metadata Interchange (XMI), Proposal to OA & DTF RFP-3, Stream based Model Interchange Format (SMIF). <http://www.omg.org/cgi-bin/doc?ad/98-10-05>, 20. October 1998.
- [Bra98a] Neil Bradley. *The XML Companion*. Addison Wesley Longman Limited, Harlow, Essex, 1998.
- [Bra98b] Tim Bray. The Annotated XML Specification. <http://www.xml.com/axml/axml.html>, 1998.
- [Kem00] Martin Kempa. Dokumentenmodell für XML-basierte World-Wide-Web-Anwendungen: VDOM. In *12. Workshop Grundlagen Datenbanken, Ploen, 2000*. in German.
- [Meg98] David Megginson et. al. SAX: The Simple API for XML. <http://www.megginson.com/SAX/index.html>, May 1998.
- [Neu99] Andreas Neumann. fxp - Processing Structured Documents in SML. In P. Trinder and G. Michaelson, editors, *1st Scottish Functional Programming Workshop, Draft Proceedings*. Herriot-Watt University, Edinburgh, Scotland, 1999.
- [Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Professional Computing. Addison-Wesley, 1994.
- [RLHJ97] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.0 Specification. Recommendation, <http://www.w3.org/TR/REC-html40-971218/>, 18. December 1997. W3 Consortium.
- [SY98] Junichi Suzuki and Yoshikazu Yamamoto. Making UML model exchangeable over the Internet with XML: UXF approach. In «UML»'98 – *Beyond the Notation*, Mulhouse, France, 1998.
- [W3 98a] W3 Consortium. Document Object Model (DOM) Level 1 Specification, Version 1.0. Recommendation, <http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/>, 1. October 1998.
- [W3 98b] W3 Consortium. Extensible Markup Language (XML) 1.0. Recommendation, <http://www.w3.org/TR/1998/REC-xml-19980210/>, 10. February 1998.
- [W3 99a] W3 Consortium. XHTML 1.0: The Extensible HyperText Markup Language, A Reformulation of HTML 4.0 in XML 1.0. Working Draft, <http://www.w3.org/TR/1999/xhtml1-19990505/>, 5. May 1999.
- [W3 99b] W3 Consortium. XML Schema Part 1: Structures. Working Draft, <http://www.w3.org/TR/1999/WD-xmlschema-1-19991217/>, 17. December 1999.
- [WS91] Larry Wall and Randall Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., 103 Morris Street, Suite A, Sebastopol, CA 95472, 1991.