Fachhochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Seminararbeit:
Document Object Model (DOM)
und
Simple API for XML (SAX)

Stefan M. Dellbrügge & Martin Kremser

14. Juni 2002

Sommersemester 2002
Verteilte und Parallele System I
Prof. Dr. Rudolf Berrendorf
Fachhochschule Bonn-Rhein-Sieg

Inhaltsverzeichnis

1 Einführung in XML

1.1 Ziele von XML

Die Extensible Markup Language (XML) wurde nach rund anderthalbjähriger Entwicklungstätigkeit am 10. Februar 1998 vom World Wide Web Consortium (W3C) in der Version 1.0 verabschiedet (W3C Recommendation `REC-xml-19980210`). XML ist eine Metasprache mit einer eindeutigen Trennung zwischen Daten und deren Darstellungformen. Der Entwicklung von XML vorausgegangen sind zwei weitere Markup-Sprachen des W3C, die einen spezielleren Fokus verfolgen:

Standardized General Markup Language (1986): Sollte die Trennung von Struktur und Darstellung eines Dokuments verwirklichen. Mit komplexen Struktur-Markern (*engl. Tag*) wird die inhaltliche Struktur von Dokumenten ausgezeichnet. Compiler überführen diese Struktur in fertig gestaltete Dokumente.

Hypertext Markup Language (1992): HTML ist eine Untermenge von SGML und sollte insbesondere der Physikergemeinde die Gestaltung ihrer Dokumente erleichtern. Mit der zunehmenden Verbreitung des World Wide Web (WWW) entwickelten Browserhersteller proprietäre Erweiterungen zu HTML und entlehnten die Sprache ihrem ursprünglichen Zweck zur einfachen Dokumentenstrukturierung.

Es zeichnete sich ab, daß für einen allgemein frei verfügbaren Datenaustausch noch keine sinnvolle Metasprache gefunden war. SGML war zu komplex für einfache Implementationen und dokumentenorientiert. Mit der Untermenge HTML konnten nur einfache Dokumente ausgezeichnet werden. Die Zweckentfremdung durch zusätzliche Tags der Browserhersteller leistete der allgemeinen Verfügbarkeit ebenfalls keine positiven Dienste.

Vor dem Hintergrund dieser Tendenzen wurde eine weitere Sprache entwickelt, deren Ziele als eine weitgehende Dokumentenunabhängigkeit und Inhaltsorientierung definiert wurden. XML geht über diese Ziele noch hinaus und fordert in der ersten Version des Standards z.B. ebenfalls die einfache Erstellung und Verarbeitung eines XML-Dokumentes für Online-Publikationen im Internet. Daraus läßt sich eindeutig der Fokus von XML herleiten, der nicht mehr auf die Dokumentenerstellung gerichtet ist, sondern einen Schritt weiter geht und ein allgemeines Datenaustauschformat zu definieren versucht.

Desweiteren sollte dieses Format universell sein. Damit stellt sich das Problem der Interpretation von XML-Inhalten. Denn ein universelles Datenaustauschformat für vielfältige Anwendungen setzt eine allgemeine Verfügbarkeit der notwendigen Struktur- und Semantikinformationen eines Dokuments voraus. Aus diesem Grunde ermöglicht XML die wahlfreie Erstellung von Tags für jeden Kontext (z.B. Banküberweisung oder Konstruktionszeichnung) mit Hilfe einer Document Type Definition (DTD) oder der neuen Typdefinition XML-Schema (siehe Kapitel 1.2). (Dokument-)Typdefinition und XML-Dokument zusammen bilden eine Einheit zur syntaktischen und semantischen Überprüfung durch einen XML-Parser. Das folgende dargestellte XML-Dokument zeigt exemplarisch die Möglichkeiten von XML als universelles Datenaustauschformat für das bekannte Programmierbeispiel "Hello World":

```
<?xml version="1.0"?>
<gruss>Hallo Welt!</gruss>
```

Funktionsweise

Das zentrale Datenobjekt der Extensible Markup Language ist das XML-Dokument. Ein XML-Dokument wird dabei in [W3C98] wie folgt beschrieben:

> [...] wenn es im Sinne dieser Spezifikation wohlgeformt ist. Ein wohlgeformtes XML-Dokument kann darüber hinaus gültig sein, sofern es bestimmten weiteren Einschränkungen genügt.

Diese Definition enthält zwei wesentliche Begriffe, die im Kontext von XML und der Verarbeitung eines XML-Dokumentes häufig benötigt werden:

Wohlgeformt (*engl. well-formedness*) meint die Existenz von mindestens einem oder mehreren Elementen in einem XML-Dokument. Desweiteren muß ein Wurzelelement oder Dokumentelement existieren und alle folgenden Elemente müssen sich korrekt ineinander schachteln.

Gültigkeit (*engl. validity*) bezeichnet eine Erweiterung der Wohlgeformtheit und berücksichtigt die Definition der frei wählbaren XML-Elemente.

Aus diesen Definitionen wird deutlich, daß ein XML-Dokument aus einer beliebigen, wohlgeformten und gültigen Abfolge von Elementen (*engl. entities*) besteht. Das Beispiel 1.1 ist nach der obigen Definition ein wohlgeformtes, aber kein gültiges XML-Dokument, weil eine Typdefinition des Tags `<gruss>` nicht bekannt ist. Eine Interpretation der Inhalte dieses XML-Dokuments durch eine Applikation und eine Verarbeitung ist nicht möglich.

1.2 XML-Schema

Für die Verarbeitung von XML-Dokumenten ist eine Typdefinition der im Dokument verwendeten Tags notwendig. Diese Aufgabe soll im weiteren Lebenszyklus von XML-Applikationen die Typdefinitionssprache XML-Schema übernehmen, deren Entwicklung 1999 begonnen wurde. Die Sprache erweitert die bisher gebräuchlichen Document Type Definitions (DTD) auf rund 40 Datentypen zur Definition beliebiger Datenstrukturen in XML. Darüberhinaus ist eine detailierte Definition der Wertebereiche für einzelne Elemente möglich, die in der DTD nicht vorhanden ist. Mit XML-Schema erfolgt die Typdefinition in einer XML-Syntax, d.h. es wird keine weitere Sprache benötigt, um Typdefinitionen zu generieren. Insbesondere der letzte Punkt wird in der zukünftigen Verbreitung von XML-Schema als positiver Aspekt gewertet. Es wird erwartet, daß dadurch auch die Anwendung von XML weiter vorangetrieben wird.

Im folgenden wird für das Beispiel 1.1 ein einfaches XML-Schema definiert, so daß eine Gültigkeitsüberprüfung durch einen Parser möglich ist:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsd:schema targetNamespace="http://www.myDomain.de/">

<xsd:element name="gruss" type="xsd:string">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
    <xsd:pattern value=".*\!$"/>
    </xsd:restriction>
```

```
    </xsd:simpleType>
  </xsd:element>
```

Mit dieser einfachen Typdefinition wird ein neues Element `<gruss>` definiert, daß aus einer beliebigen Zeichenfolge besteht. Das Ende dieser über das Element `xsd:restriction` eingeschränkten Zeichefolge soll ein Ausrufezeichen sein. Alle verfügbaren Datentypen von XML-Schema können über dieses Element genauer spezifiziert und eingeschränkt werden. In einer Typdefinition für eine Zeichenkette kann z.B. die Länge über die Einschränkung `<xsd:maxlength value="30"/>` auf maximal 30 Zeichen festgesetzt werden. Diese Beispiele verdeutlichen, auf welche Weise XML-Schema es gegenüber einer DTD ermöglicht, Datenstrukturen zu definieren und deren gültige Wertebereiche einzuschränken.

Weiteres Merkmal von XML-Schema ist eine Referenztechnik innerhalb der Datentypdefinition. Mit einer Definition `xsd:element ref=`*element* kann in einer komplexen Datenstruktur auf bereits vorher definierte einfache Strukturen zurückgegriffen werden. Damit werden Datentypen sehr einfach wiederverwendbar und können flexibel in verschiedenen Typdefinitionen eingesetzt werden.

Die beschriebene XML-Schema-Definition dient einem validierenden Parser als Vorlage für eine Gültigkeitsüberprüfung eines gegebenen XML-Dokuments. Ohne diese Vorlage kann nur die Wohlgeformheit eines Dokuments geprüft werden. Die Datentypdefinition erweitert diese Prüfung auf die Inhalte und kann deren Typkorrektheit nachweisen. Mit XML-Schema wird diese Gültigkeitsprüfung um einen Schritt erweitert und ermöglicht nicht nur den Nachweis des korrekten Datentyps sondern auch des korrekten Wertebereichs der XML-Inhalte.

1.3 XML für den Datenaustausch

XML liefert Mechanismen für den Datenaustausch und die Darstellung von Inhalten für die unterschiedlichsten Ziele. Realisiert wird dies über die Trennung von Struktur, Inhalt, und Erscheinungsbild. Inzwischen gibt es eine Vielzahl an XML-basierten Protokollen und Formaten. Die Bandbreite reicht von Standards für die Zusammenarbeit an Web-Dokumenten oder den Datenabgleich über das Internet bis hin zur Kommunikation von Softwareobjekten über das Netz. Einer der wichtigste Anwendungsbereiche ist bisher das Content Management.
XML hat sich als Protokoll-Sprache für Komponenten-Software entwickelt. Es bildet die Grundlage für Technologien wie CORBA, COM oder JavaBeans.

2 Application Programming Interfaces für XML

XML-APIs werden zu dem Zweck eingesetzt, eine vereinfachte, standardisierte Bearbeitung von XML-Dokumenten zu ermöglichen. Früher gehörte zu jedem Parser eine eigene API. Heute haben sich Standards etabliert, die sprachenunabhängige APIs zur Bearbeitung von XML bereitstellen.

Sowohl als OpenSource, als auch als komerzielle Software ist eine große Anzahl von XML-Parsern zu erwerben. Sie sind für nahezu alle bekannten Programmier- und Skriptsprachen verfügbar. Dies liegt an der Bedeutung, die einem Parser beim Arbeiten mit XML zukommt. Für Java existiert die weitaus größte Anzahl. Das ist zum einen mit der Web-Ausrichtung von Java sowie der vollständig implementierten Unicode-Unterstützung zu begründen.

Grundsätzlich kann man zwischen zwei Arten von Parsern unterscheiden:

- Nichtvalidierende Parser:
  Sie überprüfen nur, ob das Dokument wohlgeformt ist.
- Validierende Parser:
  Überprüfen zusätzlich, ob das Dokument sich an die vorgegebene DTD oder das XML-Schema hält.

Die meisten Parser-Pakete bieten die Möglichkeit, das XML-Dokument in eine Baumstruktur zu überführen. Diese Baumstruktur ist dann gemäß dem Document Objetc Model (DOM) des W3C implementiert. Einige Parser integrieren auch eine Rendering Engine. Hier handelt es sich um ein XSLT-Modul, das aus dem XML-Dokument mittels eines Stylesheets HTML oder ein anderes Textformat erzeugen kann.

Bei vielen XML-Anwendungen ist es nicht notwendig, das XML-Dokument selber anzuzeigen, sondern es geht nur darum die Daten des Dokumentes in einer Datenbank zu speichern und weiterzuverarbeiten.

Frei verfügbar und am häufigsten implementiert sind die APIs

- Simple API for XML (SAX),
- Document Objet Model (DOM),

Parser	Vorteil	Nachteil
SAX	Schnell	Call-Back-Verfahren
DOM	Objekt-orientiert	langsam
JAXP	DOM- und SAX-Parser	langsam
JDOM	Schnell, DOM- und SAX-Parser	Noch Beta Stadium

Tabelle 1: Übersicht über die Parser-APIs

- Java APIs for XML (JAXP) und
- Java Document Objetc Model (JDOM).

Die Tabelle 1 zeigt eine Gegenüberstellung der Vor- und Nachteile dieser Ansätze.

Es existieren zwei prinzipielle Methoden, die gelesenen Daten zu verarbeiten: sequentiell, d.h. gleichzeitig mit dem Einlesen der Daten per Call-Back Verfahren, oder über ein Objektmodell. Für das Objektmodell steht das standardisierte Document Object Model zur Wahl. Für Call-Back-Parser wird das SAX-Interface verwendet.

Sequentielle Verarbeitung Eine XML-Datei wird zeilenweise (Element für Element) gelesen und jeder Bestandteil sofort in seinem jeweiligen Kontext weiterverarbeitet. Der Kontext dieses Tags wird dabei durch die vorangehenden Elemente gebildet. Das Verarbeitungsprinzip entspricht dem Pipeline-Konzept von UNIX. Die Daten „fließen“ hindurch und jeder Pipeline-Abschnitt filtert sie beim Passieren. Ein Filter kann beispielsweise ungewollte Elemente oder Attribute entfernen oder modifizieren.

Objektmodell Die XML-Daten werden in ein Objektmodell überführt. Das Objektmodell wird durch Klassen repräsentiert, die die Informationen aus einem XML-Dokument aufnehmen können. Darüber hinaus stellt es Methoden zum Zugriff und zur Manipulation bereit. Während der Instanziierung eines Objektmodells durch den Parser findet eine Abbildung des Dokuments auf Objekte innerhalb der Programmiersprache statt. Das Objektmodell stellt danach die Spiegelung eines XML-Dokuments im Hauptspeicher des Rechners dar, d. h. das Dokument wird (temporär) gepuffert.

Die Aufgabe des Parsers ist es, eine XML-Datei so aufzubereiten, daß sie von der Anwendung weiterverarbeitet werden kann. Dazu gehört neben der Weiterleitung des Dokumentstreams bzw. dem Anlegen des Objektmodells auch die Überprüfung auf Fehler. Die Zusammenführung der verschiedenen Bestandteile des Dokuments, die sich durch den Einsatz von Entities in unterschiedlichen Dateien befinden können, gehört ebenfalls zu den Aufgaben.

2.1 Parsen mit Simple API for XML

2.1.1 Ereignisbasierte API

Die *Simple API for XML (SAX)* liegt als Programmierschnittstelle für die Programmiersprache Java zum Zeitpunkt dieses Dokuments in der Version 2 vor. SAX ist ein standardisiertes Interface für die Steuerung eines XML-Parsers aus einem Java-Programm. Gegenüber der Programmierschnittstelle des Document Object Model ist die Simple API for XML kein offizieller W3C-Standard, kann aber aufgrund ihrer Verbreitung als Quasi-Standard angesehen werden.

SAX ist eine ereignisgesteuerte Programmierschnittstelle für die Ansteuerung externer XML-Parser. Zu den von SAX2 unterstützten Parsern gehören unter anderem zwei von der Apache Group (`xml.apacher.org`) als Opensource entwickelte Parser:

- `org.apache.crimson.parser.XMLReaderImpl`
- `org.apache.xerces.parsers.SAXParser`

Der Parser `XMLReaderImpl` ist unter anderem seit der Version 1.4 Bestandteil des Java Development Kit (JDK). Dies macht seine Anwendung relativ einfach. Im weiteren Verlauf dieses Dokuments wird für Beispiele die Implementierung SAX2-Interface mit dem Parser Xerces (siehe Kapitel 3.2.1) verwendet. Damit soll die Portierbarkeit und Austauschbarkeit der eingesetzten Parserkomponente demonstriert werden, wenn das SAX2-Interface eingesetzt wird.

Die Schnittstelle der SAX setzt ihren Schwerpunkt wie bereits erwähnt auf die Ansteuerung externer XML-Parser. Das Objekt zur Steuerung ist das im Parser implementierte `XMLReader`-Interface aus dem Paket *org.xml.sax*. Über die Methode *parse()* wird der Parse-Vorgang einer Eingabequelle gestartet. Der Parser kann anschließend beim Eintritt vorher definierter Ereignisse einen Callback auslösen. Zu den Kategorien dieser Ereignisse gehören

- Beginn / Ende von XML-Documenten,
- Beginn / Ende von XML-Entities sowie
- mit XML-Entities markierte Inhalte.

Mit den beschriebenen Ereignisse lassen sich einfache XML-Dokumente relativ zügig bearbeiten. Wie unter 3.1 ausgeführt, sind diese Vorteile aber jeweils mit den Spezifikationen der Anwendung abzugleichen. Durch die einfache Bearbeitung eines XML-Dokuments mit den verschiedenen *startX()* und *endX()* Methoden und einer Methode zur Rückmeldung von Inhalten (*characters()*) in der Implementation des `ContentHandler` entfallen in SAX komplizierte Datenstrukturen zur Typisierung der Elemente eines XML-Dokuments.

Die Klasse `org.xml.sax.InputSource` kapselt den eingehenden Datenstrom für die XML-Applikation. Damit ist das `XMLReader`-Interface unabhängig und kann sowohl auf eine zeichen- oder byte-orientierte Datenquelle zugreifen (z.B. Dateien oder Online-Dokumente). Abhängig vom vorhandenen Datenstrom wählt der Parser, der das `XMLReader`-Interface implementiert, zwischen diesen beiden Varianten aus.

Der Parser entnimmt dem Datenstrom die einzelnen XML-Elemente und ruft den über die Methode *setContentHandler()* definierten Handler für die Verarbeitung der Ereignisse auf. Dieser Handler ist die XML-Applikation, der mit den genannten *startX()* und *endX()* Methoden die Verarbeitung der XML-Elemente obliegt.

Die beschriebene Klassenstruktur und der Datenfluß in einem Parser wird in Abbildung 1 zur Übersicht vereinfacht dargestellt.

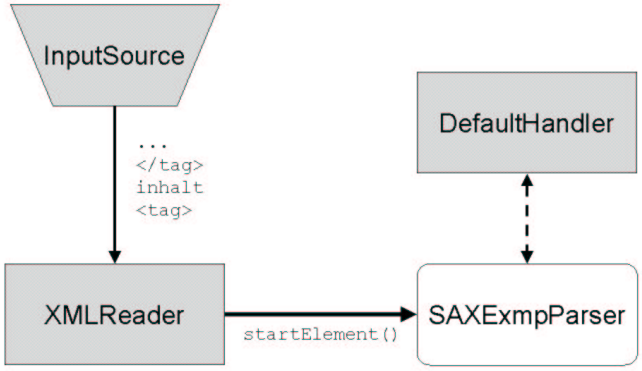

Abbildung 1: Datenfluß eines SAX-Parsers

2.1.2 SAX2-Applikationen in Java

Die im folgenden dargestellte Implementierung eines XML-Parsers mit der Simple API for XML und dem Opensource-Parser Xerces soll die wenigen einfachen Schritte darstellen, die für eine ereignisgesteuert XML-Verarbeitung notwendig sind.

Zu Beginn der Applikation müssen die notwendigen Instanzen der Klassen erzeugt und registriert werden. Wie bereits dargestellt müssen dazu

1. die applikationsspezifischen Ereignisse implementiert werden,
2. eine Implementation des `XMLReader`-Interface geladen werden,
3. die Applikation im Parser registriert werden.
4. und schließlich der Eingabedatenstrom gekapselt werden,

Diese Schritte können zum Beispiel mit den folgenden Java-Codezeilen durchgeführt werden.

```
XMLReader reader = new org.apache.xerces.parsers.SAXParser();

SAXExmpParser handler = new SAXExmpParser();

reader.setContentHandler(handler);
reader.setErrorHandler(handler);

/* Annahme: alle Parameter der Kommandozeile sind Dateinamen
 * Parsing aller übergebenen Dateien
 */
for (int i = 0; i < args.length; i++) {
    FileReader r = new FileReader(args[i]);
    reader.parse(new InputSource(r));
}
```

In diesem Beispiel wurde die Instanz des `XMLReader`-Interface fest an einen bestimmten Parser (hier `org.apache.xerces.parser.SAXParser`) gebunden. Mit der

`XMLReaderFactory` aus Version 2 der Simple API for XML kann die Applikation unabhängig vom Parser entwickelt werden. Mit dem Aufruf

```
XMLReader myReader = XMLReaderFactory.createXMLReader();
```

wird von Java ein auf dem ausführenden System installierter Parser ausgewählt. Aus den oben erläuterten Gründen, wurde in diesem Beispiel der Parser Xerces gewählt und fest in das Programm eincodiert.

Die Beispielapplikation *SAXExmpParser* erbt alle Methoden der *DefaultHandler*-Klasse des Parsers Xerces. Für eine übersichtliche Verarbeitung der verschiedenen Ereignisse des Parsers werden in der Beispielapplikation die gewünschten Methoden überschrieben und mit kurzen Bildschirmausgaben versehen. Mit diesen Ausgaben lassen sich zum Beispiel die Ereignisse `startDocument()` und `endDocument()` wie folgt kodieren:

```
public void startDocument() {
    System.out.println("Start document");
}

public void endDocument() {
    System.out.println("End document");
}
```

Die Verarbeitungslogik einer realen Implementierung wird in den jeweiligen Ereignismethoden aufgerufen. Für die einzelnen Tags eines XML-Dokuments werden die Methoden `startElement()` und `endElement()` vom Parser aufgerufen. Mit Version 2 der Simple API for XML erhalten diese Methoden beim Aufruf über den Parameter *String uri* den Namespace des Elements mitgeteilt. Dies ermöglicht die Unterstützung weitergehender Standards wie z.B. XPath oder XML-Schema. Namespaces können damit in der Verarbeitungslogik der Implementierung eingesetzt werden.

Die Verarbeitung von XML-Inhalten wird in der Methode *characters()* implementiert. Unabhängig vom vorliegenden Datenstrom werden die Daten in einem Array von Zeichen gespeichert und können beliebig verarbeitet werden.

Eine Implementierung eines rudimentären SAX2-Parsers kann mit diesen Klassen und Methoden innerhalb kurzer Zeit erstellt werden. Hierbei liegt der Implementierungsaufwand nicht in der Programmierung der Parserschnittstelle oder der Parsersteuerung sondern in der Verbarbeitungslogik der XML-Inhalte. Aufgrund der fehlenden internen Darstellung der XML-Tags als n-ärer Baum muss eine entsprechenden Logik in den ereignisverarbeitenden Methoden des Parsers vorgesehen werden. Mit einfachen Strukturen können für diese Verarbeitung temporäre Kellerspeicher verwendet werden.

Sind die Dokumentstrukturen komplexer, dann empfiehlt sich höchstwahrscheinlich eine Baumstruktur, womit das Document Object Model vorzuziehen wäre, da diese Datenstruktur mit dem Parsing-Vorgang automatisch generiert wird. Natürlich ist es kein Widerspruch, in einem SAX2-Parser einen Baum zur Dokumentenverarbeitung aufzubauen. Die Vorteile eines ereignisgesteuerten Parsers gehen damit aber verloren. 3.1 geht auf diese Problematik näher ein.

2.2 Parsen mit Document Object Model

Das Document Object Model ist ein standardisiertes Datenzugriffsmodell und kann beliebige XML-Dokumente aufnehmen. DOM ist eine Zusammenfassung von Schnittstellendefinitionen und bildet eine Beschreibung eines application programming interface (API) für XML Dokumente. Die Objekte selbst sind keine statischen Datenstrukturen, sondern sie passen sich in ihrer Ausprägung dem jeweiligen XML-Dokument an. DOM bildet dazu ein XML-Dokument als Baum in der hierarchischen Struktur ab, wie es selbst aufgebaut ist. Mit Hilfe der aus den Schnittstellen hergeleiteten Objekte kann die Struktur des XML-Dokumentes in eine Baum-Struktur in der jeweiligen Programmiersprache (vorzugsweise objektorientiert) abgebildet werden. Aus jedem Bestandteil eines XML-Dokuments wird ein Knoten ("Node") im Baum. Diese Schnittstellen werden hierarchisch zusammengefaßt und gehen von diesem „node" Element aus. Beginnend mit diesem Element definiert DOM Schnittstellen für alle weiteren XML-Tags. Außerdem werden „Low-Level-Objekte" konstruiert, wie z. B. Element, Attribut oder Text. Der Zugriff erfolgt durch die Bewegung im Baum und Aufruf von Methoden, die beispielsweise den Typ, Namen und Wert eines Knotens liefern.

Die API beschreiben die Art und Weise wie auf solche Dokumente zugegriffen werden kann bzw. diese bearbeitet werden. Mittels DOM können komplette Dokumente erstellt und deren Struktur verändert werden. Der Baum des XML-Dokumentes kann dann durch objektorientierte Zugriffe auf die einzelnen Elemente verändert werden.

DOM wurde in der plattformunabhängigen Sprache Interface Definition Language (IDL) der Object Management Groupe (OMG) verfasst und ist so auf verschiedenen Plattformen einsetzbar. Zielsetzungen der DOM-Spezifikation war die Unabhängigkeit von Plattform und Programmiersprache. Es bildet aber nur eine Vorlage, die in die verschiedene Programmierumgebungen zu implementieren ist. Für viele Sprachen wie C++, Java oder Perl sind DOM Impementierungen als Open Source verfügbar.

Die in DOM definierten Schnittstellen beschreiben Methoden mit denen der Anwender auf die einzelnen Knoten der DOM-Struktur zugreifen kann. So ist es möglich, programmgesteuert den Inhalt und Struktur von XML-Dokumenten zu manipulieren. Prinzipiell ist es egal, ob an der Input- und der Output-Seite des DOM-Parsers ein XML-Dokument, oder eine völlig andere Ausgabemethode bzw. Weiterverarbeitung steht. Wichtig ist, daß die Daten, die über eine DOM-Struktur abgebildet werden, selbst einen logischen Zusammenhang aufweisen. Es ist Aufgabe der Anwendung die DOM implementiert, die Daten zu interpretieren und im jeweiligen Kontext zu verarbeiten.

DOM wurde vom World Wide Web Consortium spezifiziert. Im September 1997 wurde das erste Dokument, das eine DOM-API beschreibt, veröffentlicht. Ausgehend davon entstand DOM Level 1, das im Oktober 1998 in der ersten Version fertiggestellt wurde. Wenig später wurde DOM Level 2 spezifiziert. Im Mai 2000 entstand eine erweiterte Version des DOM requirements document. Dieses bildet die Grundlage für die nächste DOM Generation, DOM Level 3. Die einzelnen Levels sind untereinander abwärtskompatibel. Sie unterscheiden sich untereinander nur im Funktionumfang:

- DOM Level 1 Core: definiert die Methoden und Eigenschaften zur Manipulation von XML.
- DOM Level 1 HTML: Diese Spezifikation erweitert die Level 1 Code Spezifikation um zusätzliche Funktionalität für HTML.
- DOM Level 0: Diese Spezifikation existiert nicht offiziell, wird jedoch häufig im Zusammenhang mit den Implementierungen im Microsoft Internet Explorer und Netsape Navigator benutzt.

Unter `http://www.w3.org/TR/REC-DOM-Level-1/expanded-toc.html` befinden sich die aktuellen Spezifikationen des W3C.

Die DOM Level 1-Spezifikation ist in drei Teile gegliedert:

- Fundamental Interfaces sind Schnittstellen zu den Objekten, die HTML- und XML-Dokumente gemeinsam haben, wie Elemente, Attribute, Kommentare, Text, usw.
- Extended Interfaces für XML-Objekte wie Entities, Notations und CDATA-Abschnitte.
- HTML Interfaces für den HTML-Befehlssatz. Es gibt für jedes HTMLElement ein eigenes Objekt. Attribut werden druch Objekt-Eigenschaft abgebildet.

Um die Spezifikation zu erfüllen, müssen Entwickler lediglich die Fundamental Interfaces implementieren. Darüber hinaus müssen wahlweise die HTML Interfaces, die Extended Interfaces oder beide implementiert werden. Diese Struktur hat den Vorteil, daß Skript-Programmierer für HTML auf die herkömmliche Weise auf das Dokument zugreifen können, ohne daß sie XML implementieren müssen. Andererseits können Nutzer von XML auf den erweiterten Funktionsumfang (Extended Interfaces) zugreifen, ohne daß sie durch die HTML-Methoden beeinträchtigt werden, die sie außerdem nicht benötigen.

2.2.1 Datentypen in DOM

DOM definiert zwei eigene Datentypen um die angestrebte Plattformunabhängigkeit zu gewährleisten:

- DOMString ist eine Folge von 16 Bit Werten (unsigned short). Codiert wird im UTF-16 Format.
- DOMTimeStamp zählt Millisekunden im Format unsigned long.

2.2.2 Java Implementierung

In Java ist DOM sowohl im Package org.w3c.dom als auch (zusammen mit SAX) im Package javax.xml.parsers verfügbar. Die API erzeugt wahlweise ein DOM oder SAX-Objekt. Sie funktioniert mit jedem Parser, d.h. er kann jederzeit zwischen beiden gewechselt werden. DOM und SAX könnte so parallel benutzt werden, was aber nicht zu empfehlen ist.

Im folgenden werden die Interfaces der org.w3c.dom Implementiertung in Java beschrieben.

Document: Das Document Interface repräsentiert das gesamte XML-Dokument. Es ist somit die Wurzel des gesamten Dokumentenbaumes und bildet den Zugang zu diesen Daten.

Node: Das Node-Interface ist das oberste Interface im DOM-Modell. Es stellt einen einzelnen Knoten der DOM-Struktur dar. Alle folgenden Interfaces, bis auf Text, leiten sich von „node“ ab.
Das Nodes-Interface implementiert den Zugriff auf die Attribute „nodeName“, „nodeValue“ und „attributes“. Hat ein Objekt keine Attribute, wird „null“ zurückgeliefert. Außerdem bietet das nodes-Interface Methoden zum Einfügen, Ersetzen, Löschen und Anhängen von Kind-Objekten. Auf Attribute „nodeName“, „nodeValue“ und „attributes“ kann dabei unabhängig von der Art des jeweiligen Nodes ohne casten zugegriffen werden. Über 12 verschiedene Fields kann der Typ des Nodes bestimmt werden. Abhängig davon, werden auch die Attribute entsprechend gefüllt.

Element: Element bildet das Interface für ein einzelnes XML-Tag. Es bietet Zugriffsmethoden auf die Attribute des Tags und den untergeordneten Elementen. Ab DOM-Level-2 gibt es die Möglichkeit zusätlich Namespaces zu verwenden.

Text: Schnittstelle zu einem Textelement. Enthält dieser Text-Markups, werden auch diese geparst.

2.2.3 Zukunft von DOM

DOM gehört zu den zukunftsträchtigen Parser-Modellen. Zwei wichtige Kritikpunkte existieren jedoch in den bisherigen Implementierungen. Zum einen existiert nicht die Möglichkeit sequenziell durch den Baum zu laufen, was für einzelne Anwendungen eine einfachere Lösung als der objektorientierte Zugriff darstellen würde. Zum anderen sieht DOM nicht die Möglichkeit einer direkten XML-Ausgabe vor. JDOM wird mit dem Ziel entwickelt, diese bisherigen Schwächen auszugleichen. Bisher ist JDOM allerdings noch im Beta-Stadium. Ende Februar 2001 wurde JDOM durch den Java Community Process (JCP) als offizieller Java Specification Request akzeptiert.

3 Zusammenfassung und Ausblick

3.1 Bewertung beider Ansätze

Grundätzlich verwenden verschiedene Implementationen von SAX und DOM dieselben Interfaces. Dennoch ergibt sich häufig die grundsätzliche Entscheidungsfrage, ob SAX oder DOM in dem jeweiligen Kontext besser einzusetzten ist.

Ein Vorteil des sequentiellen Ansatzes von SAX ist die Einfachheit, sowie die höhere Geschwindigkeit bei weniger umfangreichen Dateien. Hier ist häufig nur ein einmaliger Durchlauf durch das Dokument notwendig. Der sequentielle Ansatz ist außerdem sehr speichersparend, weil die Daten nicht komplett in den Speicher geladen werden müssen. Wenn das Dokument jedoch in eine komplexe Struktur zu transformieren ist, ist es dennoch unabdingbar einen Baum des Dokuments im Speicher aufzubauen.

Ein großer Nachteil des SAX-Modelles ist der fehlende wahlfreie Zugriff. Weil das

Dokument nicht im Speicher liegt, müssen die Daten in der Reihenfolge abgearbeitet werden, wie sie ankommen. Auf Basis eines Objektmodells ist dies sehr komfortabel möglich. Dadurch ergibt sich der Nachteil beim Einsatz von DOM auf große XML-Dokumente: es ist sehr hauptspeicherintensiv.

Da XML hierarchische Strukturen darstellt, ist es wichtig zu wissen, in welcher Beziehung Elemente zueinander stehen, da aus diesem Kontext die Bedeutung der Tags ausgedrückt wird. Während diese Informationen beim sequentiellen Ansatz explizit in Variablen innerhalb des Programmcodes gehalten werden müssen, um nachfolgenden Elementen die korrekte Bedeutung zuzuweisen, ist sie in einem Objektmodell direkt durch deren hierarischen objektorientierten Aufbau enthalten.

Für eine Datenbankanbindung ist der sequentielle Ansatz nicht geeignet. Die Daten liegen hierbei in einem nicht-XML-Format vor, z.B. in Tabellen. Zur Weiterverarbeitung müssen diese Daten auf eine hierarchische Struktur abgebildet werden. Dieser Vorgang ist in einem Objektmodell einfacher zu realisieren, da während der Abbildung noch keine Rücksicht auf die nachfolgenden Verarbeitungsschritte genommen werden muß. Durch den Aufbau des Objektmodells als hierarisches Gebilde, bildet sich quasi automatisch ein korrektes XML-Dokument.

3.2 Applikationsentwicklungen

3.2.1 XML-Parser Xerces

Der Opensource-Entwicklerverbund der Apache Group hat zum Thema XML eine Reihe von Projekten ins Leben gerufen. Dazu gehören Implementierungen eines Parsers, XML Stylesheet-Preprozessoren und Content Management Systeme auf der Basis von Java und XML. Ein Projekt zur Parser-Entwicklung kann zum Zeitpunkt dieser Ausarbeitung bereits mit einem Major-Release der 2. Version aufwarten.

Der XML-Parser *Xerces* (`xml.apache.org/xerces`) liegt in der Version 2.0.1 vor und unterstüzt

- XML Version 1.0, 2nd Edition,
- XML Namespaces,
- die Komponenten Core, Events und Traversal des DOM,
- SAX Version 2 mit den Komponenten Core und Extension
- Java APIs for XML (JAXP) in der Version 1.1 und
- XML Schema

Mit diesem Funktionsumfang gehört Xerces zu den Parsern mit dem größten Funktionsumfang und Standardkonformität. Xerces wird parallel für Java und C++ entwickelt. Des weiteren werden zwei Wrapper entwickelt, die Zugriff auf den Parser aus Perl heraus erlauben und die Kompatibilität mit dem Microsoft MSXML Parser herstellen.

In neueren Versionen wird die Unterstützung für neue Versionen der oben genannten APIs und Standards weiter ausgebaut. In der aktuellen Version kann es in besonderen Applikationen zu Einschränkungen des Parsers kommen. Darunter fallen Beschränkungen in der Unterstützung von XML Schema im Zusammenhang mit dem von IBM verwendeten Datenformat EBCDIC. Hier ist nicht der Parser verantwortlich, sondern der verwendete Java Development Kit. Laut Aussage der Entwickler treten keine Beschränkungen zusammen mit einem IBM JDK auf.

Weitere Einschränkungen betreffen das Document Object Model Level 3. Die Implementierung dieses Entwicklungsstandes in Xerces ist nur teilweise vorhanden (Core, Abstract Schemas sowie Load and Save). In neueren Versionen wird auch DOM Level 3 vollständig unterstützt werden.

3.2.2 Content Management System Cocoon

Eine weitere Entwicklung der Apache Group im Kontext von XML ist das sogenannte Web Publishing Framework *Cocoon* (`xml.apache.org/cocoon`). Cocoon wurde 1999 als Servelet für den Apache Webserver entwickelt, hat sich aber inzwischen zu einem eigenständigen Projekt weiterentwickelt. Das Ziel des Projektes ist die Trennung von Inhalt, Design und Verarbeitungslogik von XML-basierten Internet-Seiten. Die ersten Entwicklungen von Cocoon setzten das Docoument Object Model Level 1 ein. Seit der Aufnahme in die Apache Group wird die Simple API for XML aus Geschwindigkeits- und Skalierungsüberlegungen eingesetzt.

Das Cocoon-Servelet wird in den Tomcat Webserver der Apache Group integriert und stellt über diesen Server dynamisch generierte HTML-Seiten dar. Als Datenquellen können Relationale Datenbanken, XML-fähige Datenbanken oder das Dateisystem genutzt werden. Zu den zahlreichen Ausgabeformaten gehören neben HTML auch das Portable Document Format (PDF) und das Richt Text Format (RTF).

Der Tomcat Webserver verarbeitet die für Cocoon neu definierten sogenannten *Extensible Server Pages (XSP)*. Bestandteil dieser XSP sind eine *logic*- sowie eine *expr*-Komponente. Die *logic*-Komponente enthält Java-Code, der beim Aufruf der Seite vom Webserver ausgeführt wird. In der *expr*-Komponente wird das Ergebnis des Codes referenziert. Das folgenden Beispiel stellt ein einfaches XSP-Dokument mit einem Zugriffszähler dar:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<xsp:page language="java" xmlns:xsp="http://apache.org/xsp">

  <xsp:logic>
  static private int counter = 0;
  private synchronized int count()
  {
    return counter++;
  }
  </xsp:logic>

  <page>
  <p>I have been requested <xsp:expr>count()</xsp:expr> times.</p>
  </page>

</xsp:page>
```

Die Verarbeitung der Extensible Server Pages erfolgt in verschiedenen unabhängigen Pipelines innerhalb des Cocoon-Servelets. Zu einer Verarbeitungspipeline gehören die Komponenten

- File Generator,
- XLST Transformer und
- HTML Serializer.

Die eingehende Anfrage wird an den `File Generator` geleitet, der eine XSP-Seite aus den Teilen *Logic* und der *XML-Definition* zusammensetzt. Dieses dynamisch generierte XM-Dokument wird mit SAX an den `XLST Transformer` geleitet, der je nach Inhalt ein Stylesheet auf das Dokument anwendet und damit das Design für das Zieldokument erzeugt (z.B. für eine PDF-Datei). Im letzten Schritt wird diese Vorlage durch einen `Serialier` in das Zielformat konvertiert. Im Falle eines HTML-Dokuments wird diese Dokument dann an den Webserver gereicht, der es an den Webbrowser des Clients sendet.

Cocoon ist gemeinsam mit dem Tomcat Webserver eines der interessantesten Projekte der Apache Group. Zur Zeit ist sein Status noch nicht vergleichbar mit komerziellen Entwicklungen wie zum Beispiel dem IBM Websphere-Server, aber Cocoon ist ein sehr positives Beispiel für die Fähigkeiten der OpensourceEntwicklergemeinde. Mit der weiteren Verbreitung von XML wird auch die Bedeutung dieses Projektes steigen.

3.3 Simple Object Access Protocol

Einer der wichtigsten Bereiche in denen XML für verteilte Systeme an Bedeutung gewinnen wird, ist der Einsatz des Simple Object Access Protocol (SOAP). SOAP ist eine Alternative zu XML-RPC. SOAP ist ein vom W3-Consortium verabschiedeter Standard, der auch von Microsoft mitentwickelt wurde.
SOAP basiert in Idee und Konzept auf XML-RPC und bildet eine Weiterentwicklung davon. SOAP dürfte sich gegenüber XML-RPC durchsetzen, obwohl XML-RPC bereits in vielen Scriptsprachen implementiert ist. SOAP wird bisher (noch) vorranging in Windows-Systemen eingesetzt. So ist geplant, daß Applikationen (wie z.B. Microsoft Office) mittels dieses Protokolls mit Servern kommunizieren werden. Microsoft wird dadurch als Application Service Provider auftreten und so Dienste im Sinne einer verteilten Anwendung bereitstellen.

Die hier vorgestellten Parser bilden die Basis für zukünftige, auf XML-basierende „höherwertige“ Dienstleistungen.

Literatur

[W3C98] World Wide Web Consortium (Hrsg.), Extensible Markup Language (XML) 1.0, Wien, 1998

[SAX] www.saxproject.org

[DOM] www.w3.org/DOM

[JAXP] java.sun.com/xml

[JDOM] www.jdom.org

[Xerces] xml.apache.org/xerces

[Cocoon] xml.apache.org/cocoon