Vorschlag für ein einfaches XML-Management-System

Eckhardt Schön
Technische Universität Ilmenau
Fakultät für Informatik und Automatisierung
Institut für Praktische Informatik und Medieninformatik
Langewiesener Str. 37, 98693 Ilmenau
eckhardt.schoen@tu-ilmenau.de

Abstract. In diesem Artikel wird eine Möglichkeit zur formalen Beschreibung der Verarbeitung von XML-Dokumenten vorgestellt. Auf dieser Basis wird ein Vorschlag für die Architektur eines System zur XML-Bearbeitung entwickelt.

1 Einleitung

In diesem Artikel wird ein System zur Verarbeitung von XML-Dokumenten vorgeschlagen, das etwas von den bisherigen Systemen abweicht, in dem es sich die Spezifik der Verarbeitung von XML-Dateien mit Hilfe von XSL-Transformationen zunutze macht. Um dies besser begründen zu können, werden zu Beginn Transformationen mit Hilfe der eXtensibel Style Language (XSL) näher erläutert und es wird eine formale Beschreibung dafür vorgeschlagen.

Für die Strukturbeschreibung der Dokumente werden ausschließlich XML Schemata benutzt, da die bisher üblichen Document Type Definitions (DTD) eine andere Syntax aufweisen und inzwischen vielfach abgelöst wurden.

Es wird im Folgenden auch keine Unterscheidung zwischen XML-Dokumenten und XML-Daten getroffen, denn das System muss beide verarbeiten können. Wenn von XML-Dokumenten gesprochen wird, sind –Daten immer eingeschlossen. Da sie strenger strukturiert sind, sind sie sogar noch besser zur Verarbeitung im vorgeschlagenen System geeignet.

2 Formale Beschreibung von XSL-Transformationen

Zunächst soll die Syntax für eine formale Beschreibung der Verarbeitung von XML-Dokumenten vorgestellt werden. Eine beliebige Datei im XML-Format wird durch das Symbol <x> dargestellt. Die spitzen Klammern machen es als XML-Dokument besser kenntlich. Spezielle XML-Dokumente werden durch abweichende Buchstaben gekennzeichnet <s> für ein XML-Schema, <t> für eine XSL-Transformation und <h> für XHTML. Weitere Bezeichnungen werden im Laufe des Artikels eingeführt.

Die Verarbeitung von XML-Dokumenten wird durch Operatoren beschrieben, die auf die jeweiligen Dokumente angewendet werden können. Bemerkenswerterweise kommt man mit zwei derartigen Operatoren aus.

$$\mathbf{V}(<s>)\ <x> = \{0, 1\}$$

Der Operator **V** beschreibt die Validierung des XML-Dokumentes <x> gegenüber einem XML-Schema <s>. Das Ergebnis dieser Operation kann nur 1 (Dokument gültig) oder 0 (Dokument nicht gültig) sein. Alle Dokumente, die gültig bezüglich eines Schemas <s> sind, gehören zur selben Dokumentenklasse $\boldsymbol{S}$.

$$\boldsymbol{S} = \{ <x> \mid \mathbf{V}(<s>)\ <x> = 1 \}$$

Die Validierung erfolgt durch einen XML-Parser bzw. durch einen speziellen Validator, was allerdings keine wirkliche Verarbeitung bedeutet. Dazu ist eine XSL-Transformation **T** nötig, die mit Hilfe des Style Sheets <t> das Dokument $<x_1>$ in das Dokument $<x_2>$ transformiert.

$$\mathbf{T}(<t>)\ <x_1> = <x_2>$$

Dabei funktioniert natürlich nicht jede Transformation auf jedem Dokument. Eine XSL-Transformation muss auf der Basis einer Dokumentenklasse $\boldsymbol{S}$ entworfen werden.

Die Abbildung 1 zeigt das Szenarium bei der Verarbeitung eines Dokumentes $<x_1>$ mit Hilfe von XSL-Transformationen.

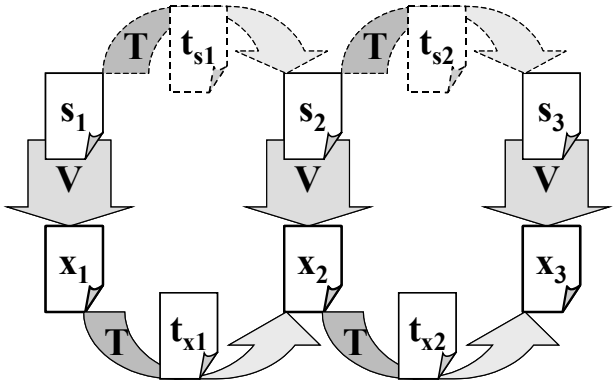


Abbildung 1 XSL-Transformationen von XML-Dokumenten

In der Abbildung sind eine Reihe von Zusammenhängen zu erkennen. Die Transformation $<t_{x1}>$ sollte so konzipiert sein, dass sie für jedes Dokumente $<x_1>$ funktioniert, das der Dokumentenklasse $\boldsymbol{S_1}$ angehören. Es ist wenig sinnvoll, eine XSL-Transformation für ein einzelnes Dokument zu entwerfen. Dagegen sollte die Tatsache, dass $<x_1> \in \boldsymbol{S_1}$ ist, hinreichend dafür sein, dass die Transformation $<t_{x1}>$ auf das Dokument angewendet werden kann.

Das Ergebnis $<x_2>$ der Transformation gehört i.allg. zu einer anderen Dokumentenklasse, die durch das Schema $<s_2>$ definiert wird. Da die Schemata selbst auch XML-Dokumente sind, gibt es prinzipiell auch eine Transformation, die die Schemata ineinander überführt.

$$\mathbf{T}(<t_{s1}>)\ <s_1> = <s_2>$$

Die Transformationen $<t_{x1}>$ und $<t_{s1}>$ hängen zwar nicht semantisch wohl aber syntaktisch zusammen. Da alle Schemata <s> Elemente der Dokumentenklasse „XML-Schema“ (beschrieben selbst durch ein Schema bzw. eine DTD) sind, überführt $<t_{s1}>$ Dokumente derselben Klasse ineinander. Für die explizite Formulierung der Schema-Transformation $<t_{s1}>$ gibt es i.allg. keine Notwendigkeit, da der Aufwand größer ist als für die Neudefinition des Schemas $<s_2>$. Eine Untersuchung über die genauen semantischen Zusammenhänge und syntaktische Restriktionen ist durchaus von Interesse, wird aber hier nicht durchgeführt.

Prinzipiell ist sogar eine Meta-Transformation $<t_t>$ denkbar, die die Transformationen selbst ineinander überführt.

$$\mathbf{T}(<t_t>)\ <t_{s1}> = <t_{x1}>$$

Diese könnte zwar eine praktische Relevanz besitzen, scheint aber nur sehr schwer realisierbar zu sein, obwohl alle Transformationen derselben Dokumentenklasse angehören.

Die Abbildung 1 zeigt die Verarbeitung von XML-Dokumenten als Hintereinanderausführung von XSL-Transformationen.

$$\mathbf{T}(<t_{x2}>)\ \mathbf{T}(<t_{x1}>)\ <x_1> = <x_3>$$

Dies ist die Grundlage für das zu entwickelnde System zur Verarbeitung von XML-Dokumenten.

In diesem Zusammenhang ist es wichtig anzumerken, dass es i.allg. keine Rücktransformation $<t_{x1}^{-1}>$ gibt, die folgendes bewirkt:

$$\mathbf{T}(<t_{x1}^{-1}>)\ <x_2> = <x_1>\ .$$

Schon das Sortieren einer vorher unsortierten Liste oder das Filtern eines Datenbestandes sind nicht rückgängig zu machen.

Die Verarbeitung von XML-Dokumenten lässt sich als Folge von XSL-Transformationen $\{<t_{x1}>, <t_{x2}>, \ldots <t_{xn}>\}$ auffassen. Die Version $<x_{n+1}>$ eines Dokumentes kann eindeutig durch das Ausgangsdokument $<x_1>$ und die Folge der Transformationen beschrieben werden. Das bedeutet praktisch, dass nicht jede Version eines Dokumentes gespeichert werden muss, um die History nachträglich rekonstruieren zu können. Die Transformationsdateien sind i.allg. viel kleiner als das Dokument selbst und werden u.U. für eine gewisse Menge von Ausgangsdokumenten in gleicher Weise verwandt. Damit lässt sich eine Menge Speicherplatz sparen, ohne dass Informationen verloren gehen.

3 Präsentation und Metadaten

Will man ein System zur Verarbeitung und zum Management von XML-Dokumenten entwerfen, so muss man berücksichtigen, dass die Dokumente dort nicht isoliert auftreten. Jedes Dokument <x> gehört sinnvollerweise zu einer Klasse, die durch das Schema <s> beschieben wird und mit Hilfe eines Styles <c> visualisiert werden kann. Dabei kann es sich um ein Cascading Style Sheet (CSS) oder eine XSL:FO-

Datei (Formating Objects) handeln. Letztere ist im Hinblick auf die Konsistenz des Systems zu bevorzugen. Auch <c> sollte auf der Basis von <s> entwickelt werden und damit auf alle Dokumente dieser Klasse anwendbar sein. Die Präsentation lässt sich formal durch einen Operator **P** beschreiben.

$$\mathbf{P}(<c>)\ <x> = <h>$$

Die Ausgabedatei könnte wie hier XHTML sein, aber auch PDF oder ein anderes Format, das mit einem passenden Viewer visualisiert werden kann.

Für ein sinnvolles Managen der Dateien wird i.allg. ein Satz von Metadaten assoziiert, der ebenfalls im XML-Format abgelegt werden kann. Die Datei <m> enthält sowohl Informationen, die den Inhalt des Dokumentes beschreiben (Autor, Titel, Schlüsselworte, ...) als auch Statusinformationen (Dokument-ID, Ablageort, Änderungsdatum, zugehöriges Schema, Version, ...).

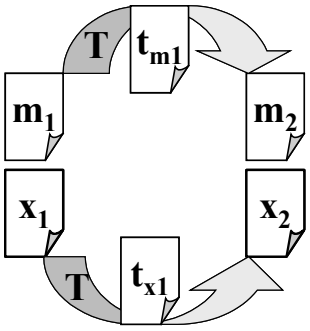


Abbildung 2 Transformationen von Dokument und Metadaten

Zu jeder Transformation des Dokumentes muss eine Transformation der Metadaten parallel ausgeführt werden, die zumindest die Statusinformationen aktualisiert.

$$\mathbf{T}(<t_{x1}>)\ <x_1> = <x_2> \ \Rightarrow\ \mathbf{T}(<t_{m1}>)\ <m_1> = <m_2>$$

Die Metadaten genügen einem Schema $<s_m>$, dass in der Regel unabhängig von der Dokumentenklasse $<s_1>$ bzw. $<s_2>$ ist. Es sollte so allgemein gehalten werden, dass es für die Metadaten aller Dokumente im System nutzbar ist. Damit kann eine einheitliche Verwaltung sichergestellt werden.

Die Transformationen $<t_{x1}>$ und $<t_{m1}>$ sind zwar semantisch miteinander verbunden, ein syntaktischer Zusammenhang wird sich jedoch nur in ausgewählten Fällen herstellen lassen.

Über die Art der Metadaten-Strukturierung wurde bisher nur festgestellt, dass sie als XML-Datei vorliegen muss. Es ist sicher sinnvoll hier ein standardisiertes Format zu benutzen, wie es mit dem Resource Description Framework (RDF) existiert.

4 Architekturvorschlag für ein XML-verarbeitendes System

Im Folgenden wird ein Vorschlag für die prinzipielle Architektur eines Systems zur Bearbeitung von XML-Dokumenten vorgestellt. Die Architektur lässt sich zu einem Dokumenten- oder Content-Management-System weiterentwickeln, was aber hier nicht geschehen soll. Das Ziel ist lediglich, XML-Dokumente zu bearbeiten, zu verwalten und zu speichern. Dabei gehen wir in diesem ersten Ansatz davon aus, dass alle Dokumente zu einer Klasse gehören, die allerdings beliebig sein kann. Das heißt man könnte mit dem System sowohl XHTML, als auch SVG oder XML-EDI verarbeiten. Nur der Editor und der Viewer müssten ausgetauscht werden.

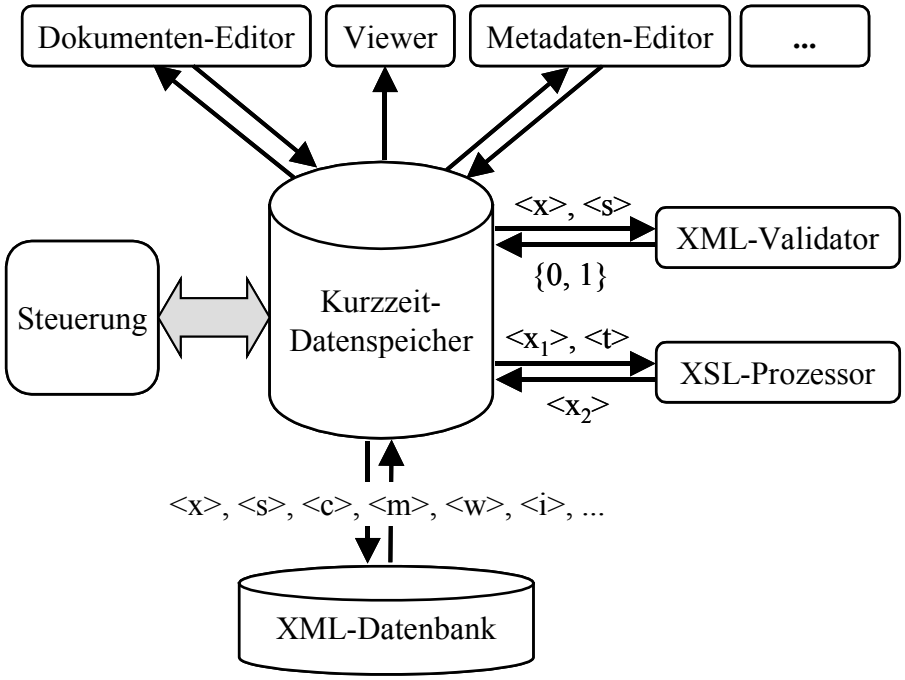


Abbildung 3 Beispielarchitektur

Die Architektur (Abb. 3) umfasst die üblichen drei Schichten. In der untersten Schicht erfolgt die Datenhaltung der XML-Dokumente, am besten in einer entsprechenden Datenbank. Darüber befindet sich die Mittelschicht, die im Wesentlichen vier Komponenten umfasst: einen Kurzzeit-Datenspeicher (z.B. File-System), eine Steuerungskomponente, einen XML-Validator und einen XSL-Prozessor. Von letzteren beiden könnten aus Performance-Gründen auch mehrere Instanzen aktiv sein.

In der Anwendungsschicht sind die Komponenten angesiedelt, über die der Nutzer oder externe Prozesse mit dem System kommunizieren. Wichtig zu bemerken ist, dass alle Informationen in Form von XML ausgetauscht werden.

Auch der Kurzzeitspeicher muss lediglich XML-Dateien speichern. Dazu können Index-Dateien <i> gehören, die durch einen Indexierer aus den Metadaten aufgebaut

werden, Dateien <w> für ein Workflow-Management oder Konfigurationsdateien <k> für die Steuerung des System.

Die folgende Tabelle stellt einige mögliche Komponenten auf Anwendungsschicht zusammen und gibt an, welche Art von XML-Dateien als In- und Output der Komponenten auftreten.

Komponente	Input	Output
Viewer	<x>, <c>	-
Editor	<x>, <s>, <c>, <m>	<t_x>, <t_m>
Metadaten-Editor	<m>	<t_m>
Schema-Editor	<s>	<t_s>
Workflow-Manager	<w>	<t_w>
Indexierer	<i>, <m>	<t_i>
Such-Tool	<s>, <i>	<q>
Administrationstool	<k>	<t_k>
Import-Komponente	Nicht-XML-Datei	<x>
Export-Komponente	<x>	Nicht-XML-Datei

Tabelle 1 Ein- und Ausgabedateien von Komponenten

Während manche Komponenten recht universell für die verschiedensten Dokumente eingesetzt werden können, müssen andere genauer auf die Dokumentenstruktur abgestimmt werden. Das betrifft besonders auf den Editor zu, wenn er einen gewissen Eingabekomfort haben soll. So lassen sich z.B. Vektorgrafiken im SVG-Format besser mit einem Grafiktool bearbeiten, als mit einem textorientierten Editor.

Wichtig bei allen Editoren ist, dass sie keine geänderten XML-Dokumente an das System zurückliefern, sondern lediglich die gewünschten Änderungen protokollieren und in die XSL-Syntax umsetzen. Die Änderungen am Dokument werden erst in der Mittelschicht unter Verwendung des XSL-Prozessors vorgenommen. Das hat den Vorteil, dass das System prinzipiell für alle Arten von XML-Dokumenten geeignet ist. Um eine weitere Dokumentenklasse bearbeiten zu können, genügt es, einen geeigneten Editor (und eventuell Viewer) zu integrieren. Wenn man sich mit geringerem Komfort begnügt, ist auch das nicht unbedingt notwendig. Jedes System sollte über einen Standard-XML-Editor verfügen, der das Editieren beliebiger Dokumente auf Textebene ermöglicht.

Die Veränderungen der Dokumente (Versionen) bleiben bei diesem Konzept gut nachvollziehbar, ohne dass alle geänderten Dokumente selbst abgespeichert werden müssen (siehe Abschnitt 2). Es reicht aus, einzelne Versionen in größeren Abständen im Datenspeicher abzulegen. Dieser Vorteil betrifft selbstverständlich alle Arten von Dokumenten (<x>, <m>, <i>, …), da alle in Form von XML-Dateien vorliegen und deshalb in gleicher Art und Weise behandelt werden können.

Eine detailliertere Beschreibung der Vorgänge kann leider an dieser Stelle nicht erfolgen.

5 Zusammenfassung

Auf der Basis der formalen Beschreibung von XSL-Transformationen wurde ein System zur Verarbeitung und Speicherung von XML-Dokumenten vorgeschlagen. Das Neue an der vorgestellten Architektur ist, dass die eigentliche Veränderung der Dokumente in der Mittelschicht erfolgt, während ein Editor lediglich die Anweisungen für die Transformation erstellt. Auf diese Weise lassen sich alle Änderungen an Dokumenten auch im Nachhinein leicht nachvollziehen. Die Versionierung wird erleichtert und es müssen nicht mehr alle Versionen eines Dokumentes gespeichert werden. Es genügt, die Ausgangsdatei und die XSL-Dateien zu speichern, um die komplette History verfolgen zu können. Damit wird das einheitliche Dokumentenformat auch für eine Vereinfachung der Architektur und der Abläufe im System genutzt. Ein weiterer Vorteil ist, dass sich das System leicht auf neue Dokumentenklassen erweitern lässt.

Literatur

Extensible Stylesheet Language (XSL), Version 1.0, W3C Recommendation 15 October 2001: http://www.w3.org/TR/xsl/

J. Euzenat, L. Tardif : XML transformation flow processing, http://transmorpher.inrialpes.fr/paper/

S. Gallard: Using XML/XSL for Web Publication, Intranet-Journal 10/2000, http://www.intranetjournal.com/articles/200010/xml_10_04_00a.html

St. Jablonski, Ch. Meiler: Web-Content-Managementsysteme, Informatik-Spektrum 04/2002, S.101