

FH Aargau: Vorlesung XML Sommersemester 2005

Test XML Schema

MUSTERLÖSUNG (Lösungen in grün & underline)

30.5.2005, 8³⁰ - 9³⁰

Bitte schreiben Sie *sorgfältig* und *gut lesbar*! Danke!

Für diesen Test dürfen Sie *keinerlei Unterlagen* verwenden, insbesondere auch keine elektronischen Hilfsmittel wie Laptop oder Organizer!

1. XML Schema hat zwei hauptsächliche Anwendungen, das eine ist die *Validierung von Dokumenten* gegen das Schema, das andere ist die *Ergänzung des Dokuments durch Typinformationen*, so dass Applikationen mit Typen arbeiten können anstatt mit Namen von Elementen oder Attributen. Geben Sie für *beide Anwendungen* (Validierung und Typannotation) *mindestens zwei* Fälle an, in denen diese Funktion nützlich ist.

- a. Validierung

- i. Erspart Programmieraufwand (die Bedingungen werden vom XML Schema Parser getestet), es sind keine programmierten Tests notwendig.
- ii. XML Schemas lassen sich kombinieren, so dass man verschiedene "Validierungsstufen" definieren kann, die in verschiedenen Schemas definiert werden.
- iii. Portable Validierung, da XML Schema Parser praktisch auf allen Plattformen existieren und man die Validierung überall durchführen kann, ohne Code portieren zu müssen.
- iv. Deklarative Beschreibungen sind einfacher verständlich und besser zu ändern als Code, also ist die Validierung gegen ein Schema besser aus der Sicht von Dokumentation und späteren Modifikationen.

- b. Typannotation

- i. Programme können besser auf Typen arbeiten, weil diese Beziehungen zueinander haben (Typableitung) und der Code diese Beziehungen ausnutzen kann (z.B. abgeleitete Typen → abgeleitete Klassen).
- ii. XML Schema Mechanismen wie Type Substitution oder Substitution Groups sind kompliziert handzuhaben, wenn man auf Elementnamen arbeitet, aus der Typsicht ist der Umgang wesentlich einfacher.
- iii. In verschiedenen Kontexten können Elemente unterschiedliche Inhalte (d.h. Typen) haben. Namensbasiert müsste man also jeweils überprüfen, in welchem Kontext eine Element auftritt,

während man bei typbasierter Verarbeitung direkt auf den Typen Zugriff hat und damit auch auf die komplette Information, die zu diesem Element in diesem Kontext gehört.

2. Im Prinzip unterstützt XML Schema zwei Arten der *Wiederverwendung* von Inhalten (Content Models und Attribute): Einerseits die Verwendung von *Typableitungen*, andererseits die Verwendung von *Named Groups* (Named Model Groups und Attribute Groups). In beiden Fällen werden gleichbleibende Inhaltsteile in verschiedenen Typen verwendet. Erklären Sie den *Unterschied* beider Methoden, und welche Methode man *in welchem Fall* anwendet.
 - Unterschied: Bei der Typableitung ist die Wiederverwendung stärker eingeschränkt, indem der Ableitungsmechanismus (Einschränkung oder Erweiterung) definiert, welche Regeln für die Wiederverwendung gelten müssen. Named Groups können dagegen ohne Einschränkungen verwendet werden, z.B. muss man keine Reihenfolge einhalten (bei der Typerweiterung kann man dagegen neue Inhalte immer nur hinten an bestehende Inhalte anfügen).
 - Typableitungen verwendet man bei wesensmässigen Gemeinsamkeiten zwischen den modellierten Klassen. Da XML Schema keine mehrfache Vererbung erlaubt, beschränkt sich die Wiederverwendung ohnehin darauf, dass ein Typ von genau einem anderen abgeleitet werden kann. Soll ein Typ daher verschiedene Dinge wiederverwenden, so muss dies zwangsläufig über Namen Groups geschehen.
3. XML Schema ist streng genommen *keine XML Schemasprache*, sondern eine *Infoset Schemasprache*, d.h. die Sprache ist auf dem XML Infoset definiert und nicht auf einem XML Dokument. Was bedeutet dieser Unterschied, wenn man *zukünftige Versionen* von XML Schema betrachtet und mögliche *neue Features*, die man dort einbauen will (davon ausgehend, dass auch neue Versionen von XML Schema *auf dem Infoset aufbauen*)?

Da die Sprache auf dem Infoset aufbaut, kann man alle die Dinge, die nicht im Infoset enthalten sind, auch nicht benutzen, so wird man z.B. keine Features einbauen können, die die Attributreihenfolge oder die Verwendung von Entities im Dokument betreffen, da diese Information im Infoset gar nicht vorhanden ist.
4. XML Schema definiert *Identity Constraints*, mit denen die *ID/IDREF Features* der DTDs bereitgestellt und erweitert werden. Identity Constraints benutzen *XPaths*, um die Knoten zu selektieren, auf die sich die Constraints beziehen. Es geht dabei also um *verschiedene Teile des Dokumentenbaumes*, die selektiert werden. Beschreiben Sie die *drei wichtigen Aspekte* (wie in der Vorlesung behandelt) für Identity Constraints hinsichtlich ihrer Definition, der Selektion von Knoten und der Definition ihrer Constraints.
 - a. Definition des Constraints: Identity Constraints werden in Elementdefinitionen definiert, aus diesem Grund ist es sehr wichtig, an welchem Ort die Constraints definiert werden (z.B. im Document Element oder an anderen Orten).
 - b. Selektion der Knoten: Ausgehend von der Definition werden Knoten selektiert, über die sich der Constraint erstrecken soll. Diese werden per relativem XPath selektiert, der folglich ausgehend von der Definition des Constraints alle Knoten selektieren muss, für die der Constraint gelten soll.

- c. Selektion der Felder: Ausgehend von den selektierten Knoten (!) werden die Felder selektiert, die den Constraint erfüllen sollen, auch hier wieder über einen relativen XPath. Dies können Elemente oder Attribute sein, meistens werden Attribute verwendet. Es können auch mehrere Felder angegeben werden.
- 5. Aus einer Anwendung bekommen Sie die Anforderung, dass *Mixed Content eingeschränkt* werden soll. Mit XML Schema ist dies nicht möglich, da *Mixed Content typfrei ist* und deswegen keine Einschränkungen definiert werden können. Es wird verlangt, die *verwendeten Zeichen* einzuschränken (nur ASCII), und die *Länge*, und zwar soll die Gesamtanzahl aller Zeichen im Mixed Content eines Element limitiert werden. Beschreiben Sie mindestens zwei mögliche Wege, diese Anforderung zu implementieren (ohne XML Schema, das in diesem Fall nutzlos ist). Es geht nicht um Programmdetails, sondern um ein Konzept, diese Anforderung zu implementieren.
 - a. DOM/Java: Mit einem XML Parser wird das Dokument geparsed, und dann wird mit einem Javaprogramm getestet, ob die Bedingungen erfüllt sind.
 - b. Schematron: Mit einem Schematron Schema werden die Bedingungen definiert, und dann werden Dokumente gegen das Schema validiert. Ob das mit einem native Schematron Tool oder der XSLT-Variante geschieht, ist dabei gleichgültig.
 - c. XSLT: Mit einem kleinen XSLT-Programm werden die entsprechenden Tests programmiert, und das XSLT Programm übernimmt damit die Funktion eines Validators.
- 6. Sie erhalten die Aufgabe, ein *XML Schema Validator* zu schreiben (also das, was das SQC Tool macht, das wir benutzt haben). Eingabe soll ein XML Schema sein (d.h. genau genommen ein *XML Dokument, das ein XML Schema definiert*), Ausgabe das *Resultat Ihrer Validierung*. Beschreiben Sie in ganz groben Schritten Ihr Vorgehen (einige wenige Schritte reichen, es geht um die Idee, wie man XML Schema Strukturen am besten verarbeiten könnte).
 - a. Parsing: Mit einem XML Parser wird das XML Dokument eingelesen, damit steht das XML Dokument als DOM-Tree zur Verfügung.
 - b. Validierung: In diesem Schritt wird das XML Dokument des XML Schema gegen das Schema für Schemas validiert. Dies findet zwar nicht alle möglichen Fehler, stellt aber zumindest sicher, dass der grundsätzliche Aufbau des Schema Dokuments korrekt ist.
 - c. Transformation: Ausgehend vom DOM-Tree wird ein Modell aufgebaut, das die inhaltlichen Zusammenhänge der XML Schema Komponenten repräsentiert, sich also nicht mehr an der XML Syntax des XML Schema orientiert (wie das Eingabedokument), sondern am strukturellen Modell von XML Schema.
 - d. Checking: Auf dieser XML Schema Struktur (die nichts mehr direkt mit dem ursprünglichen XML Dokument zu tun hat) können nun alle Bedingungen überprüft werden, die gelten müssen, damit ein XML Schema korrekt ist. Erst jetzt kann man sicher sein, dass das XML Schema wirklich ein korrektes XML Schema ist.