

---

Seminarvortrag über die

# Integration von XML in Anwendungen

von

Matthias Ströh

# Gliederung

- Einführung
  - Was ist grundlegend bei der Integration von XML zu beachten?
  - Wie kommt XML in einer Anwendung zum Einsatz?
  - Generelle Ziele der Integration
- Erweiterbarkeit und Wiederverwendbarkeit
- Performanz und Lesbarkeit von XML-Dokumenten
- Konsistenz
- Parser / Schnittstellen (SAX & DOM & Data Binding)
- Sicherheit
- XML-Tools
- DTDs oder XSD-Schemata
- Integration einer XML-Validation in eine fertige Architektur
- Vermeiden von „Übervalidation“
- Gezielte Validation
- Erstellen von Modularen und erweiterbaren XSD-Schemata
- XML und Datenbanken
- Standardisieren von XML-Schemata
- Performanzsteigerung der Applikationen

# Einführung

---

Was ist grundlegend bei der Integration von XML zu beachten?

- **Vorausschauend planen und entwickeln** ist essentiell bei der Arbeit mit XML.
- Integration von XML erfordert viel **Erfahrung und Know-how**.
- XML sollte auch in Stand-Alone-Applikationen für **spätere Erweiterungen** in betracht gezogen werden.
- Es sind **stabile Datenmodelle** zu entwickeln.
- Es sollte **ausreichend Zeit** für die Entwicklung von Datenmodellen zur Verfügung stehen um **später viel Zeit und Arbeit zu sparen**.

# Einführung

---

Was ist grundlegend bei der Integration von XML zu beachten?

Diese Punkte beziehen sich auch auf Tools die mit XML eingesetzt werden und nicht nur auf die Datenrepräsentation.

- Performanz
- Sicherheit
- Erweiterbarkeit
- Wiederverwendbarkeit
- Datenzugriff

# Einführung

---

Beispiele was XML beeinflussen kann:

- **Anforderungen** an die **Bandbreite** zum Übertragen eines XML-Dokuments.
- **Anforderungen** für die **Verarbeitung** (parsen) eines XML-Dokuments.
- Die **Struktur** des zum **Dokument** korrespondierenden XML-Schema.
- Die **Rechenzeitanforderung** zum **Validieren** des Dokuments mit Hilfe des korrespondierenden XML-Schemas.
- Das **Design** von korrespondierenden **Style Sheets**.
- Die **Rechenzeit** um das Dokument mit Hilfe des Style Sheets zu **transformieren**.
- Die **Sicherheit** die für das entsprechende Dokument nötig ist.
- **Erweiterbarkeit** des **Dokuments** im Laufe der Zeit.
- **Einsatzmöglichkeiten** des **Dokuments** oder Teilen des Dokuments in der Applikation.
- Die **Qualität** der **Datenrepräsentation** für Benutzer der Applikationskomponenten und der End-Benutzer.

# Einführung

---

Wie kommt XML in Anwendung zum Einsatz?

- XML ist gut geeignet zum **Austausch von Daten** zwischen mehreren Anwendungen auch **Plattformunabhängig**.
- **Austausch von Daten** zwischen den **Schichten** in einer Applikation.
- Als **Repräsentation** von Daten in einer **gut lesbaren Form**.

# Einführung

---

Generelle Ziele der Integration von XML

- **Reibungsloser Datenaustausch** zwischen mehreren Applikationen oder innerhalb einer Applikation.
- **Gemeinsame Repräsentation von Daten** aus unterschiedlichen Datenquellen
- **Schließen von Lücken** zwischen verschiedenen **Daten- und Geschäftsmodellen** (z.B. Artikeldaten und Kundendaten)
- Erleichterung der **Wartbarkeit und Erweiterbarkeit**
- **Flexibilität**
- **Plattformunabhängigkeit**

# Erweiterbarkeit und Wiederverwendbarkeit

---

Datenmodelle werden sich im Laufe der Zeit ändern. **Flexible** Datenmodelle sind gefragt!

- Möglichst Einsatz von **Kind-Elementen** statt Attributen.
- **Allgemeine Elementnamen** sind einzusetzen, z.B. keine Namen die sich ändern können.
- **Kleine modulare XML-Schemata** sind zu erstellen, die logisch unterteilt werden.
- **Möglichst kein Hardcoden von Namensräumen**, sondern dynamische Zuordnung zur Laufzeit.
- Möglichst **Rekursionen vermeiden**, da der Überblick über die Schachtelungstiefe verloren geht.
- **Einsatz von XSD-Schemata** um die **Erweiterbarkeit** zu gewährleisten.

# Performanz und Lesbarkeit von XML-Dokumenten

---

- Durch die Flexibilität von XML lassen sich selbstbeschreibende Datenstrukturen erstellen.
  - Bessere Wartbarkeit
  - Weniger Fehlinterpretationen
- Selbstbeschreibende Elementnamen sind oftmals länger und blähen die Größe eines XML-Dokumentes auf. Das wird besonders problematisch, wenn die Dokumente sehr umfangreich sind.
  - das parsen verlangsamt sich
  - Transformationen dauern länger
  - das Transferieren des Dokumentes dauert länger
- Da XML-Dokumente nur selten von Menschengen gelesen werden, ist es vorteilhaft kurze Elementnamen zu wählen. Prädestiniert dazu sind oft vorkommende Elementnamen.

# Performanz und Lesbarkeit von XML-Dokumenten

- Kurze Elementnamen können durch **Kommentare** beschrieben werden.
- Um die Lesbarkeit zu gewährleisten kann für die kurzen Elementnamen auch eine **Legende als Kommentar** in das Dokument eingefügt werden.
- Um lange Elementnamen zu vermeiden können auch Elternelemente eingesetzt werden.

Beispiel:

```
<InventarNummer>1</InventarNummer>  
<InventarNummer>2</InventarNummer>  
<InventarNummer>3</InventarNummer>
```

} 102 Zeichen

Wird zu:

```
<Inventar>  
  <Nr>1</Nr>  
  <Nr>2</Nr>  
  <Nr>3</Nr>  
</Inventar>
```

} 51 Zeichen

- Der Einsatz von Entitäten reduziert die Größe von XML-Dokumenten.
- Eine weitere Möglichkeit XML-Dokumente zu verkleinern ist das Komprimieren, dass sich jedoch negativ auf die Laufzeit auswirkt.

# Performanz und Lesbarkeit von XML-Dokumenten

---

- Sinnvoll kann es auch sein **zwei Versionen des Dokuments** zu führen. Eine zur **maschinellen Verarbeitung** und eine zum Arbeiten für die **Entwickler**.
- Eine **Mischung aus Performanz und Lesbarkeit** ist hier zuzusuchen. Die Lesbarkeit sollte nicht zu sehr in den Hintergrund gestellt werden, denn gute **Lesbarkeit macht XML unter anderem aus**.

# Konsistenz

---

- Konsistenz wird oft bei der Arbeit mit XML nicht beachtet.
- Richtlinien:
  - Wenn es ein Verarbeitungskonzept für die XML-Dokumente gibt, dann sollte sich streng daran gehalten werden.
  - Transformations- und Validationsschritte sollten immer in der gleichen Sequenz stattfinden.
  - Wenn Performanzprobleme auftreten, dann sollten diese Maßnahmen zur Performanzsteigerung auf alle Datenflüsse angewendet werden.
  - Jede selbst erstellte Komponente einer Applikation sollte eine Schnittstelle erhalten um Daten zu senden und zu empfangen.
- Es gibt viele Möglichkeiten Konsistenz zu erreichen, aber eine speziellen Empfehlungen.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

- **DOM** steht für **Document Object Model**
- DOM ist eine vom W3-Konsortium definierte Schnittstelle zum Zugriff auf HTML oder XML-Dokumente.
- DOM ist seit 1998 ein Standard des W3C.
- Hauptaufgaben von DOM:
  - Navigation zwischen den Elementen eines XML-Dokumentes die in DOM als Knoten bezeichnet werden.
  - Erzeugen, Löschen oder Verschieben von Knoten
  - Auslesen, Ändern und Löschen von Textinhalten
- **Vorteil** von DOM ist der **schnelle Zugriff auf das komplette Dokument**.
- **Nachteil** ist, dass das **komplette Dokument in den Speicher** geladen wird. Bei großen Dokumenten ist das Laden **zeitaufwändig** und der **Speicherplatzbedarf** sehr hoch.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

- **SAX** steht für **Simple API for XML**.
- Ist ebenfalls eine **Schnittstelle zum Zugriff auf XML-Dokumente**.
- **Aktuelle Version ist SAX 2.0** die 2000 veröffentlicht wurde.
- SAX ist aus den **Defiziten von DOM** hervorgegangen.
- **Vorteil** von SAX ist die **serielle Verarbeitung** von umfangreichen XML-Dokumenten. Es wird nicht das komplette XML-Dokument zu einem Zeitpunkt eingelesen.
- SAX ist in vielen Produkten zu finden.
- **Nachteil** von SAX ist, dass **nicht die Möglichkeit besteht wahlfrei auf das XML-Dokument zuzugreifen**.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

- Beim Data Binding werden XML-Schemata durch einen Schema-Compiler automatisch in die Klassen einer objektorientierten Programmiersprache übersetzt.
- Data Binding lässt eine schnelle Integration von XML in Applikationen zu.
- Das spart Zeit und Kosten.
- Es entsteht nur wenig Code.
- Arbeiten mit Data Binding ist wenig fehleranfällig.
- Schlecht mit DTD einsetzbar, weil DTDs nicht aussagekräftig genug sind.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

Wann sollte DOM benutzt werden?

- Wenn das XML-Dokument weniger als 1000 Elemente besitzt.
- Wenn man die Dokumentstruktur zur Laufzeit verändern möchte.
- Wenn die Applikation direkten Zugriff auf das komplette XML-Dokument benötigt.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

Wann sollte SAX benutzt werden?

- Wenn das XML-Dokument mehr als 1000 Elemente besitzt.
- Für Dokumente mit einer begrenzten Schachtelungstiefe.
- Wenn DOM zu langsam ist.
- Wenn nur Teile des XML-Dokumentes benötigt werden.

# Parser / Schnittstellen (SAX & DOM & Data Binding)

---

Wann sollte Data Binding benutzt werden?

- Wenn mit statischen XML-Dokumenten gearbeitet wird.
- Wenn eine klassenorientierte Schnittstelle benötigt wird.
- Wenn der Datenzugriff der Programmlogik vereinfacht werden soll.
- Wenn die Objektorientierung der Programmlogik nicht gebrochen werden soll.

# Sicherheit

---

- XML-Dokumente können verschiedener natur sein.
  - Lokale Nutzung der Dokumente
  - Weltweiter Versand und Nutzung der Dokumente
  - Sensible Daten
  - Daten, die die Firma bei Manipulation schädigen können

# Sicherheit

---

- Gefahr geht von öffentlichen DTDs oder Schemata aus, da die Möglichkeit einer böswilligen Änderung besteht.
  - Kann Applikationen weltweit betreffen
  - Lösung:
    - Kein Zugriff auf öffentliche DTDs oder Schemata von kritischen Anwendungen es sei denn, sie sind unter eigener Kontrolle.
    - Falls der Server der DTD oder des Schema keine hohe Sicherheit aufweist, sollte eine lokale Kopie der DTD oder des Schema erstellt werden.

# Sicherheit

---

- Gefahren durch unzureichende Firewalls
  - Dadurch fehlende Prüfung auf Integrität von XML-Daten.
  - Lösung:
    - Integration von eigenen Validationsroutinen in die Applikation die den Inhalt analysieren.
    - Parser und XSLT-Prozessoren können zum Teil Kontrollen zur Sicherheit übernehmen.

# Sicherheit

---

- **Element- und Attributlänge sollten geprüft werden.** Mit überladenen Elementen oder Attributen können Hacker den Parser, sowie Komponenten von Applikationen zum Absturz bringen.
- Behandle Fehler, die vom Parser oder XSLT-Prozessor generiert werden, wie man es auch von anderen Programmiersprachen her kennt. => komplette Fehlerbehandlung, die auf jegliche Art Fehler angemessen reagiert.

# XML-Tools

---

- Viele XML-Tools sind auf dem Markt verfügbar.
- Welches Tool eine Anwendung wie sinnvoll unterstützt muss für jede Anwendung geprüft werden.
- Tools sind oft nicht gut Dokumentiert, Arbeitsweise im Detail nicht bekannt. => ausprobieren

# XML-Tools

---

Worauf sollte bei Tools geachtet werden?

- Bewertung des automatisch generierten Codes. Kann der Codegenerator an die Applikation angepasst werden?
- Fügt das Tool eigene Ergänzungen an den Code an die unnötig oder sogar hinderlich sind?
  - Abhängigkeit von dem Tool
  - Aufgeblähte Dokumente
- Generiert das Tool eigene Dateien die es zum Arbeiten benötigt?
  - Gefahr vom Verlust der Plattformunabhängigkeit
  - Abhängigkeit von dem Tool

# XML-Tools

---

- Schemakonversionen und Generation sind Standardfunktionen in vielen XML-Tools. Bei wichtigen Dokumenten sollten die Dokumente immer auf ihre Korrektheit geprüft werden.
- Beim Verwalten von großen Dokumenten ist eine Editor mit Volltextsuche sehr wichtig. XPath und XQuery sind hier ebenfalls sehr hilfreich.
- Einige Tools besitzen Schnittstellen um Applikationen, die mit XML arbeitet zu Administrieren. Solche Tools können viel Programmieraufwand ersparen.

# XML-Tools

---

- Eine ausführliche Fehlermeldung ist bei XML-Tools wichtig. Z.B. beim Validieren eines XML-Dokuments gegen eine DTD. Wo ist was passiert?
- Gibt es im Unternehmen bereits XML-Tools, die kostenlos verwendet werden dürfen?

# DTDs oder XSD-Schemata

---

Wann sollte man DTDs verwenden?

- Wenn die **Struktur von XML-Dokumenten einfach** ist und man sicher ist, dass sie es bleibt.
- Wenn der **Validationscode verschickt werden würde**, da DTDs in der Regel kleiner sind und somit **Bandbreite sparen**.
- Wenn in der Applikation **verwendete Tools keine XSD-Schema unterstützen**.
- Auf Grund des Alters von DTD sind **viele Entwickler vertrauter mit DTDs**.
- Es existieren **mehr fertige DTDs** die erworben oder wieder verwendet werden können (ist gerade im Umbruch).
- **XSD-Schemata** benötigt **mehr Rechenleistung**, da sie auf Grund ihrer Komplexität mehr Overhead erzeugen.

# DTDs oder XSD-Schemata

---

Wann sollte man XSD-Schemata verwenden?

- Wenn man **von Grund auf eine Applikation entwickelt**, da sich XSD-Schema voraussichtlich durchsetzen wird.
- Wenn die **XML-Dokumente komplex** sind, da man mit XSD-Schema flexible Validationsregeln erstellen kann.
- Wenn man **Vorteile** aus **unterschiedlichen Datentypen** ziehen kann.
- **XSD-Schema** kann das Erstellen von **komplizierten Datenrepräsentationen erleichtern**.
- Wenn man eine enge **Zusammenarbeit mit relationalen Datenbanken** anstrebt sind XSD-Schemata besser geeignet als DTDs.
- Wenn das **Nachrichtenprotokoll SOAP eingesetzt wird**, da XSD gut von SOAP unterstützt wird.
- Wenn der **Validationscode wahrscheinlich erweitert** wird.

# DTDs oder XSD-Schemata

---

Wann ist ein Mix aus XSD und DTDs Möglich?

- XSDs können, in einem Mix aus XSDs und DTDs, die Hauptrolle übernehmen und DTDs können eine untergeordnete Rolle übernehmen wie z.B. die Übertragung von Daten.
- Wenn eine Applikation bereits mit DTDs arbeitet können Neuerungen durch XSDs hinzugefügt werden.
- Erweiterte Validation von XML-Dokumenten durch XSD.  
Grundanforderungen an die Validation werden durch DTDs erfüllt und zusätzliche Anforderungen durch XSDs.
  - Dadurch kann eine Applikation auch Stück für Stück von DTDs auf XSD umgestellt werden ohne hohen Zeitaufwand und ohne hohe Kosten in einem kurzen Zeitraum bewältigen zu müssen.

# DTDs oder XSD-Schemata

---

## Syntaktischen Grenzen von XSD-Schemata

- Eingeschränkte Bedingungen
- Keine Null-Wert Attribute
- Validation von großen numerischen Werten

## Performanzgrenzen von XSD-Schemata

- Bei hoher Komplexität der XSD-Schemata
- Geschwätzige Syntax
- Lange Ergebnis-Schemata
- Nicht gut zum versenden geeignet
- Können sehr laufzeitintensiv sein

# DTDs oder XSD-Schemata

---

## Alternativen zu DTDs und XSD-Schema

- Schema Adjunct Framework (SAF)
- Schematron
- RELAX und RELAX NG
- Schema for Object Oriented XML (SOX)
- Document Schema Definition Languages (DSDL)
  
- Wenn gewisse Funktionalität von diesen Ressourcen benötigt wird, könnte man die Validation in Zonen aufteilen und jede Zone übernimmt den Teil, für den sie am besten geschaffen ist.

# DTDs oder XSD-Schemata

---

## Verbessern von XSD-Schemavalidation

- XSLT hat zwar nicht die Hauptaufgabe Dokumente zu validieren, hat aber Funktionen die dabei nützlich sein können.
- Eigentlich will man möglichst vermeiden Funktionalität in die Applikation auszulagern, dass geht aber nicht immer. In dem Fall kann man mit Kommentaren z.B. vom Typ Appinfo, die die Routine in der Applikation nicht als Kommentar behandelt, zusätzliche Validationsregeln übergeben.

# Integration einer XML-Validation in eine fertige Architektur

---

Applikation oft auf mehrere Applikationsserver verteilt. Verleitet dazu die Schemata auch auf jedem Server zu speichern, was nicht wünschenswert ist.

- Applikationsdesign in dem Fall nicht gut, da **Redundanz entsteht** und der **Wartungsaufwand sich erhöht**.
- Bei mehreren Kopien von Schemata besteht ein **höheres Risiko einer Veränderung** einer der Kopien.
- Durch viele Schemata auf vielen Servern können **unbefugte Personen leichter an vertrauliche Informationen kommen**.
- Alternative wäre jedem XML-Dokument sein korrespondierendes Schema mitzuschicken. Problem dabei kann eine unzureichende Infrastruktur sein.

# Integration einer XML-Validation in eine fertige Architektur

---

Was sollte man beachten?

- Am besten sollte man die Schemata in der Datenbank der Applikation speichern.
- Die am häufigsten verwendeten Schemata beim Start der Applikation in den Hauptspeicher laden.
- Möglichst immer die Schemata im Hauptspeicher haben, wenn sie benötigt werden.

# Vermeiden von „Übervalidation“

---

Wenn Daten durch eine Applikation verschickt werden, dann ist es üblich sie vorher zu validieren. Bei einem regen Datenaustausch zwischen zwei Applikationen kann es vorkommen, dass ein Dokument öfters validiert wird ohne dass es verändert wurde.

Lösung: In jedem XML-Dokument ein Element aufnehmen mit einem Attribut, z.B. „Validiert“ das bei jeder Änderung auf „no“ gesetzt wird und nach einer erfolgreichen Validierung auf „yes“.

# Gezielte Validation

---

Dokumente mit statischem Anteil müssen nicht jedes Mal komplett validiert werden.

- Einmal komplette Validation, anschließend nur noch die Teile, die sich im Laufe der Zeit verändern.
- Solche Validationen sollten in Routinen stattfinden, in denen Teile eines Dokuments geändert werden.

# Erstellen von Modularen und erweiterbaren XSD-Schemata

---

Die Möglichkeit ein Schema aus mehreren Schemata zu erstellen bietet viel Flexibilität.

- In XSD-Schema werden andere Schemata mit „Include“ oder „Import“ eingefügt.
- Mit „redefine“ können alte Elemente oder Attribute überschrieben werden.
- Einfach erweiterbare Dokumente
- Wiederverwendbarkeit
- Mit Hilfe des „any“-Elements kann eine gewisse Freiheit für Erweiterungen eingeräumt werden.

# XML und Datenbanken

---

- Ältere Datenbanken ohne XML-Unterstützung
  - sind beschränkt mit XML einsetzbar.
  - Arbeiten mit XML ist auf einfache Strings beschränkt.
  - Viel eigene Arbeit ist gefordert.
- Es gibt Softwareprodukte die helfen Integrationsdefizite zu beheben. Ältere Datenformate können z.B. dynamisch in XML umgewandelt werden.
  - Die Nutzung dieser Produkte kann oft Arbeit und Laufzeit sparen.
  - Produkte unterliegen nicht der Qualitätskontrolle der Datenbankhersteller.
  - Genaue Funktionsprüfung erforderlich.
- Datenbanken mit XML-Erweiterungen
  - bieten eingeschränkte Funktionalität auf der Plattform des Herstellers.
  - Vorteil ist die sichere Zusammenarbeit von Datenbank und erweiterter Funktionalität.
- Erstrebenswert sind Datenbanken die von Haus aus mit XML Arbeiten.

# Standardisieren von XML-Schemata

---

- Unüberlegtes Erstellen von XML-Schemata ist in vielen Unternehmen ein Problem.
- Mitarbeiter wissen nicht um die Wichtigkeit der Standardisierung.
- Häufige Vorgehensweise:
  1. Applikation erstellen die XML benutzt 😊
  2. Schemata erstellen die gebraucht werden 😊
  3. Versuchen zwei Applikationen miteinander zu verbinden. 😊
  4. Feststellen, dass die erstellten XML-Schemata nicht zusammen passen. ☹
- Notlösungen:
  1. Schemata anpassen
  2. Konversion der XML-Dokumente zwischen den Applikationen
  3. Weiterarbeiten und hoffen das alles gut geht.
- Notlösungen werden aber auch in Zukunft wieder zu Problemen führen.

# Standardisieren von XML-Schemata

---

Wie kann man Ordnung schaffen?

- **XML-Datenverwahrer ernennen**, der XML im Unternehmen kontrolliert und regelt.
- **Datenpool erstellen** in dem alle XML-Schemata, die im Unternehmen existieren, gesammelt werden.
- Richtlinie für die Entwicklung von Datenmodellen und Dokumentstrukturen entwickeln.
- Tools, die automatisch Code generieren, suchen. Z.B. Datenkonverter die automatisch aus XML-Dokumenten XML-Schemata erstellen. Diese sind nicht immer Fehlerfrei. Außerdem sollte man die Arbeitsweise des Tools und wo der Code zum Einsatz kommt Dokumentieren.
- **Neue Applikationen** die mit XML arbeiten sollten **vom Datenverwahrer überprüft werden**.
  - Doppelte Datenmodelle vermeiden
  - Mögliche Modularisierungen
  - Namens- und Strukturkonventionen einhalten
  - Möglichkeiten der Wiederverwendung

# Standardisieren von XML-Schemata

---

Wie kann man Ordnung schaffen?

- Kommunizieren der Prozesse, Standards und Technologien.
- Mitarbeiterschulungen.
- Standards und Prozessbeschreibungen können über das Intranet bereitgestellt werden.
- Versionskontrolle der Schemata ist wichtig für die Integrität der Daten.

# Performanzsteigerung der Applikationen

---

## Strategische Redundanzen

- Kein indizierter Zugriff in Standard-XML-Architektur auf Daten.
- Verschiedene Sichten auf Daten benötigt viel Laufzeit.
- Es gibt Produkte die einen indizierten Zugriff und Warteschlangenfunktionen zur Verfügung stellen.
- Wenn man weiß, welche Datensichten benötigt werden kann Redundanz eingefügt werden und ein XML-Dokument erstellt und separat auf der Festplatte gespeichert werden. Automatische Datenaktualisierung wird benötigt.
- Eigentlich spricht Redundanz gegen Datenmodellierungsprinzipien, kann in dem Fall aber Performanz verbessern.

# Performanzsteigerung der Applikationen

---

Was sollte man beachten?

- Nicht zuviel Redundanz, sonst geht der Vorteil wieder verloren.
- Redundante Dokumente in eine logische Gruppe zusammenfassen und keine weiteren Abhängigkeiten zulassen.

---

Vielen Dank für die Aufmerksamkeit.