Basis-Architekturen für Web-Anwendungen

Erik Wilde
Institut für Technische Informatik und Kommunikationsnetze (TIK)
ETH Zürich

Überblick

- Web-Technologien
  - technische Grundlagen des Web
- Web-basierte Applikationen
  - Minimalmodell
- Schwächen des Minimalmodells
  - Erweiterung auf Multi-Tier Modell
  - Multi-Tier Modell für verschiedene Interfaces
- Web Services
  - Web-basierte Applikationen und Web Services
  - Multi-Tier Modell für Web Service Support

Web-Technologien

- es gibt 3 Hauptbestandteile des WWW
- Identifikation von Ressourcen
  - *Universal Resource Identifier (URI)*
  - unterteilt in *Scheme* und *Scheme Specific Part*
- Zugriff auf Ressourcen
  - *Hypertext Transfer Protocol (HTTP)*
  - stateless Client/Server Protokoll
- Format von Ressourcen
  - *Hypertext Markup Language (HTML)*
  - Markup-Sprache für Hypermedia Dokumente
- unzählige weitere Technologien

Web-basierte Anwendungen

- Verwendung (mehrerer) Web-Technologien
  - URIs für Dienste
  - HTTP als Transportprotokoll (Firewall Thematik)
  - HTML Interfaces (überall unterstützt)
- Auswahl je nach Anwendung
  - oft neue Anbindung an altes System
    - Modelle nicht immer gleich (z.B. Sessions)
    - Anbindung muss Modelle anpassen (Session Handling)
  - billiger zu entwickeln und anzuwenden
    - existierende (teils freie) Standardkomponenten
    - zunehmende Erfahrung der Entwickler
    - keine Probleme mit Software-Distribution

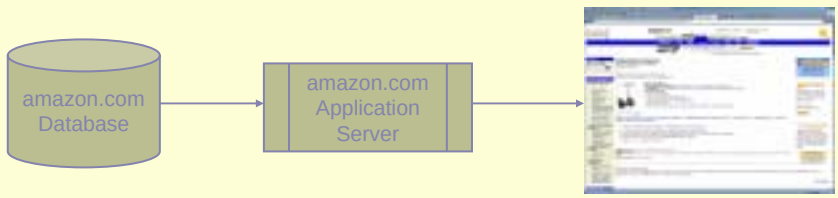
Eigenschaften von Web-Apps

- Interfaces vorgegeben durch Zielumgebung
  - einfaches HTML ohne Zusätze
  - HTML mit Scripting
  - HTML mit Java Applets
- Entwicklungsumgebungen
  - für die Interfaces selber
  - für die Codeentwicklung der Web-App
  - für die Anbindung bestehender Systeme
    - Middleware für Integration verteilter Komponenten
    - Middlerware-Adapter für bestehende Software
- Entwicklung und Betrieb sind bekannt
  - Einsparungen durch bestehendes Know-How

HTML-basierte GUIs

- HTML als Sprache für das Interface-Design
- Vorteile
  - einfach zu entwickeln
  - auf allen Plattformen unterstützt
- Nachteile
  - sehr simples Interface-Design notwendig
    - Erweiterung über Scripting oder Applets
  - relativ hohe Server-Last
    - HTML Clients sind meist *Thin Clients*
- ein Back-End, verschiedene Front-Ends
  - HTML für low-end und allgemeinen Zugriff
  - non-HTML für Power Users und Intranet

Amazon als 3-Tier App



Datenhaltung Applikation Präsentation

Modell-Anpassungen

- viele Applikationen benutzen Sessions
  - Anmelden, Arbeiten, Abmelden
  - das Web (HTTP) funktioniert *stateless*
- Beispiel: Session Handling
  - Cookies (eingebettet in HTTP Messages)
    - in immer mehr Browsern abgeschaltet
  - URI Rewriting (angefügt an die URI)
    - funktioniert in jedem Fall
    - deutlich aufwendiger zu implementieren
- Lösung: abstraktes Session-Konzept
  - Programmierer können Sessions benutzen
  - Session Handling Mechanismus wird konfiguriert

Thin und Thick Clients

- Unterscheidung von Intranet und Internet
  - Intranet erlaubt Kontrolle über Browser
  - Internet muss mit "allen" Browsern funktionieren
- HTML ist eine passive Beschreibungssprache
  - Interaktionen brauchen immer den Server
    - ineffizient und langsam
- HTML kann erweitert werden um Dynamik
  - Scripting Sprachen (eingebettet im HTML)
  - Java Applets (eigenständige Java Programme)
  - muss vom Client unterstützt werden

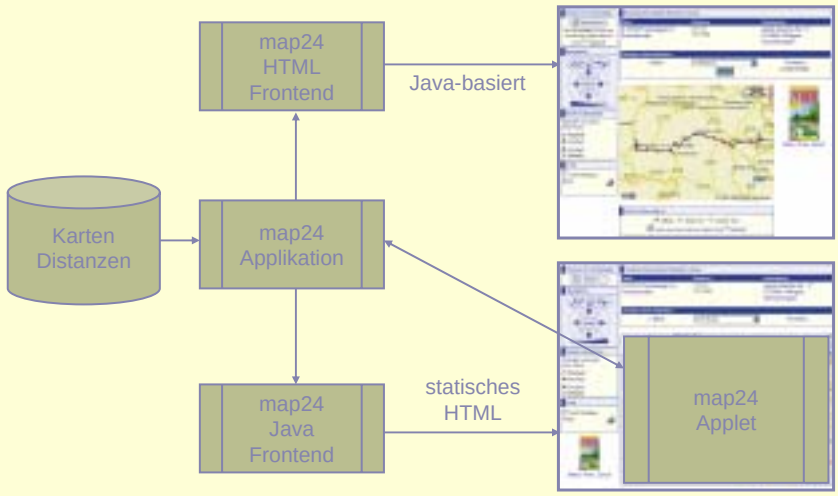
Web-Applikation mit Applet

Statische HTML-Version

Nachteile des 3-Tier Modells

- Zugriff ist Bestandteil des *Application Tier*
  - Vermischung von zwei Aspekten
    - Applikationslogik
    - Zugriffsmechanismen auf die Applikationslogik
- Probleme mit Wartung und Erweiterung
  - monolithisches System
  - Erweiterung um neue Zugriffsmethoden
- besserer Weg: weitere Gliederung
  - *Application Tier* für Applikationslogik
  - *Access Tier* für Zugriffsmethoden

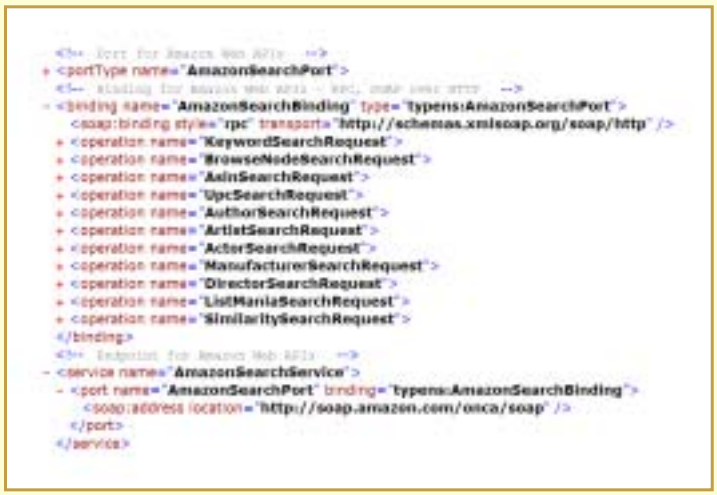
map24 als Multi-Interface App



Web Services (API)

- Erfolg des Internet durch das Web
  - einfache Benutzung für Jedermann
  - allgemein verfügbare Programme (Browser)
- Einschränkungen des Web
  - einfache Applikationen
    - z.B. HTTP als *Stateless Protocol*
    - einfaches Interface
    - Konzentration auf menschliche Benutzer
- Web Services dehnen das Web aus
  - Web-Technologien für Applikationen
  - die gleiche Idee, eine andere Benutzergruppe

Amazon als Beispiel

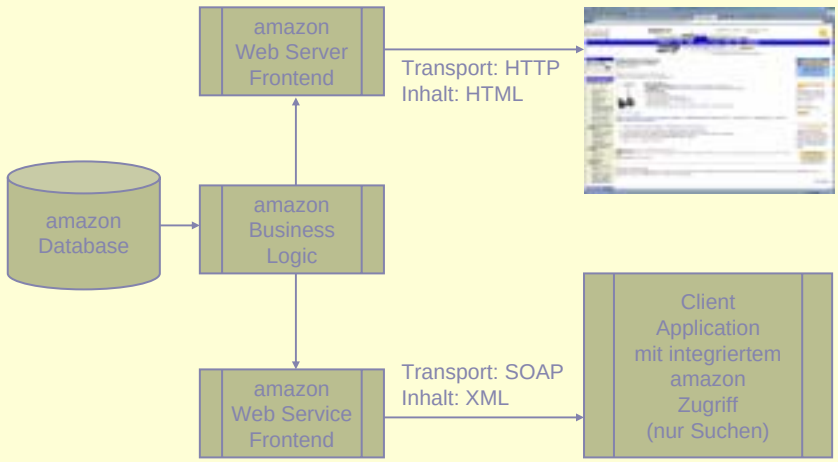


```
<!-- Port for Amazon Web APIs -->
+ <portType name="AmazonSearchPort">
<!-- Binding for Amazon Web APIs - RPC, SOAP over HTTP -->
- <binding name="AmazonSearchBinding" type="typens:AmazonSearchPort">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
  + <operation name="KeywordSearchRequest">
  + <operation name="BrowseNodeSearchRequest">
  + <operation name="AsinSearchRequest">
  + <operation name="UpcSearchRequest">
  + <operation name="AuthorSearchRequest">
  + <operation name="ArtistSearchRequest">
  + <operation name="ActorSearchRequest">
  + <operation name="ManufacturerSearchRequest">
  + <operation name="DirectorSearchRequest">
  + <operation name="ListManiaSearchRequest">
  + <operation name="SimilaritySearchRequest">
</binding>
<!-- Endpoint for Amazon Web APIs -->
- <service name="AmazonSearchService">
  - <port name="AmazonSearchPort" binding="typens:AmazonSearchBinding">
    <soap:address location="http://soap.amazon.com/onca/soap" />
  </port>
</service>
```


Amazon WSDL Definition

- Schnittstelle für der Amazon Web Service
- alle Datentypen für die Applikation
- alle Messages (Ein-/Ausgaben)
  - definiert mit XML Schema
- `portType` für die Operationen
  - gruppiert die Messages zu Operationen
- `binding` für einen Transportmechanismus
  - bindet eine Operation an einen Transportdienst
- `service` für die Adresse des Dienstes
  - für HTTP eine (oder mehrere) URI

Amazon als Multi-Tier App



Zusammenfassung

- Web-basierte Applikationen
  - verwenden Web-Technologien
  - basieren auf bestimmten Design Patterns
- Web Services als neue Technologie
  - erweitern den Web-Horizont
  - grössere Vorteile sauber designter Applikationen
- Implementierungsunterstützung
  - existierende Entwicklungsumgebungen
  - existierende Design-Methoden

XML Validation Pipelines

Erik Wilde
Institut für Technische Informatik und Kommunikationsnetze (TIK)
ETH Zürich

Überblick

- XML Validierung
  - was ist Validierung?
  - wozu braucht man Validierung?
  - was sind Probleme der Validierung?
- XML Pipelines
  - Validierung als Pipeline
  - Arten von XML Pipelines
  - XML Validierungspipelines

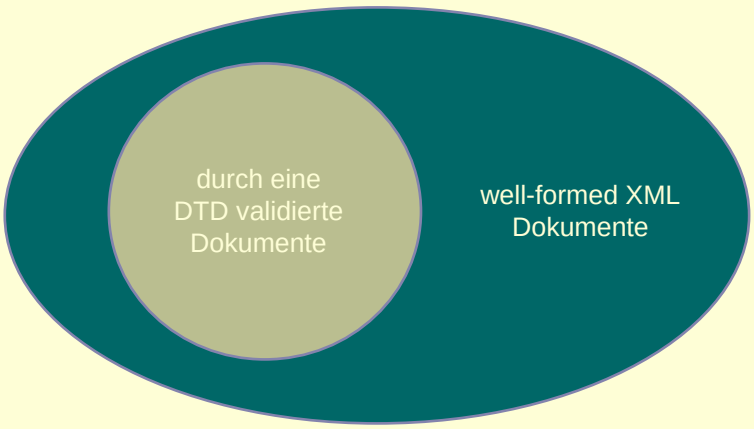
Wozu Schemasprachen?

- Einschränkung der akzeptierten Dokumente
  - Verwendung von Standardsoftware zur Validierung
- Schemasprachen können nie alles
  - DTDs unterstützen kaum Datentypen
    - dass ein Wert eine positive Zahl sein muss
  - XML Schema unterstützt keine Co-Constraints
    - dass ein Wert kleiner sein muss als ein anderer
  - Schematron kann nicht mit externen Daten arbeiten
    - dass ein Wert in einer Datenbank existieren muss

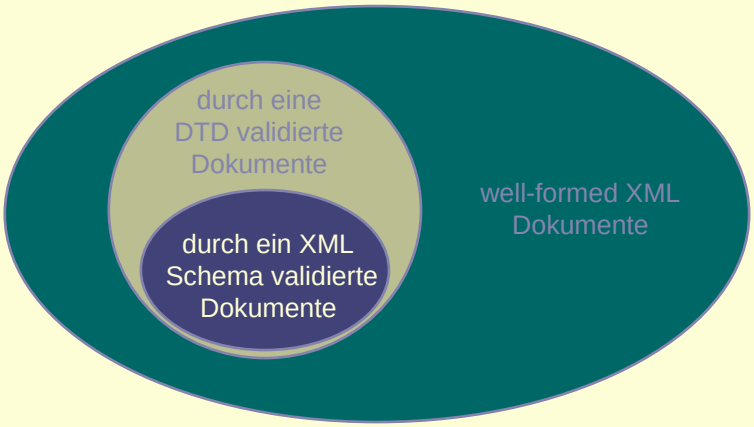
Well-formed und valid XML

- XML unterscheidet zwischen zwei "Levels"
  - *well-formed* gehorchen dem XML-Standard
  - *valid* sind well-formed und gehorchen einer DTD
- well-formed Dokumente sind korrektes XML
  - falls keine DTD vorhanden (nicht immer nötig!)
  - falls DTD nicht verfügbar
  - falls keine Weiterverarbeitung notwendig
- valid Dokumente sind korrekt gemäss DTD
  - Validierung anhand einer DTD
  - sinnvolle Kontrolle zur Weiterverarbeitung

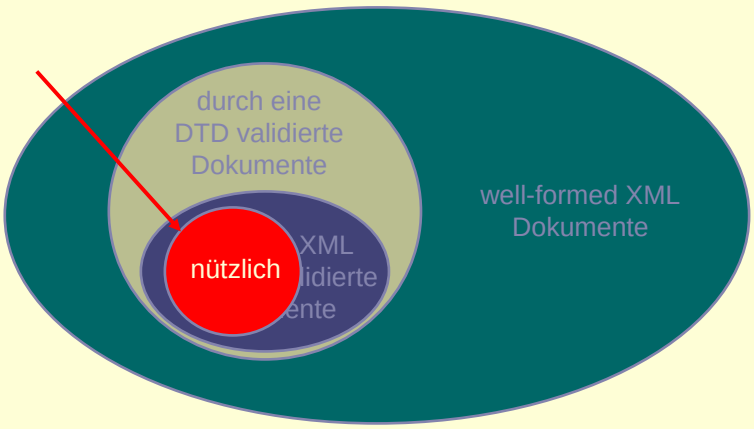
Schemasprachen als Konzept



XML Schema als DTD++



Schema-valid & Application-valid



Data Quality Assurance

"Every application has a responsibility to ensure that only valid data is inserted into the repository. After all, what value would an application offer if the data it replied upon were corrupted?"

Chuck Cavaness (Programming Jakarta Struts)

Data Quality Handling

- Data Quality wird meist angenommen
  - "meine Daten sind immer korrektes XML"
- Data Quality in einem Workflow
  - meist stetig abnehmend
  - wird meistens hingenommen
- Data Quality Probleme kosten Geld
  - in der Entwicklung (Konzept und Infrastruktur)
  - und in der Anwendung (Processing Cycles)

Data Quality Firewalls

- entsprechend "normalen" Firewalls
  - existieren auf verschiedenen Netzwerkebenen
  - warum nicht auch auf XML-Applikationsebene?
- an strategischen Stellen im Workflow
  - reduziert die Qualitätsverluste
  - z.B. zwischen Abteilungen/Gruppen/Einheiten
- muss konfiguriert werden
  - Menge an eingebauten Validierungstools
  - u.U. Schnittstelle für Business Logic Validierung
- nicht trivial zu implementieren
  - genaue Definition von Qualität ist nötig
  - organisatorische Zuordnung (Produzent/Konsument)

XML "Validity Levels"

- XML selber definiert zwei Levels
  - *well-formed* und *valid* XML Documents
- XML Schema definiert *schema validity*
  - sehr fein strukturiert (jedes Element/Attribut)
- andere Standards funktionieren anders
  - RELAX NG processing ist nicht näher definiert
  - Schematron definiert Assertions und Reports
    - Teile von XSLT-Code
- Anwendungen brauchen bessere Konzepte

Mögliche XML "Validity Levels"

- not XML (syntax error, u.U. recoverable)
- well-formed, aber (external) entity errors
- well-formed
- character set validated
- character normalization validated
- valid (DTD)
- schema valid (partially oder fully)
- valid with additional rules (Schematron)
- business logic valid (DB lookups, ...)

XML Character Validation

- viele Software hat Zeichensatz Limiten
  - Legacy Software ist häufig nur 8-bit fähig
- XML basiert auf Unicode
  - die spezifische Codierung ist nicht festgelegt
    - UTF-8 und UTF-16 müssen unterstützt werden
- XML kennt wenige Einschränkungen
- an anderen Stellen volle Freiheit
  - #PCDATA oder `xs:string` erlaubt alles
  - XML Schema und *mixed Content* problematisch

XML und Unicode

- XML erlaubt "beliebige" Unicode Zeichen
  - einige wenige Einschränkungen

```
<文書 改訂日付="１９９９年３月１日">
  <題>サンプル</題>
  <段落>これはサンプル文書です。</段落>
  <!-- コメント -->
  <段落>&会社名;</段落>
  <図面 図面実体名="サンプル"/>
</文書>
```

CRVX

- *Character Repertoire Validation for XML*
  - einfache Schemasprache für Zeichensatz-Validierung
  - `http://dret.net/projects/crvx/`
- Character Handling ist ein einfaches Problem
  - aber sehr häufig und wichtig zu beachten
- CRVX Prototyp
  - basierend auf XSLT 2.0
    - XSLT 2.0 unterstützt Regular Expressions
  - kompiliert ein CRVX Schema in ein XSLT 2.0
  - dieses XSLT 2.0 ist der Validator
    - keine spezielle Software notwendig

Pipeline Verarbeitungsmodell

- XML ist ein Austauschformat
  - gemeinhin ist Austausch über das Netz gemeint
  - Austausch auf einem Rechner ist ein Spezialfall
- Pipelines aus der Unix Welt
  - Kopplung von vielen Tools
  - viele einfache Tools = viele Anwendungen
  - definierte Ausführungsumgebung
    - Ein- und Ausgaben und ein Fehlerkanal
    - Prozessmanagement

Unix Pipes

- Pipes werden in der Shell implementiert
  - Shell ist die Ausführungsumgebung
- Tools haben ein gemeinsames Datenmodell
  - Zeichenströme mit LF als Zeilentrennung
  - Probleme der Zeichencodierung (ASCII vs. UTF-8)
- die Shell übernimmt die Steuerung
  - Erzeugung der Prozesse (fork/exec)
  - Verbindung der Prozesse (stdin/stdout/stderr)

XML Pipelines

- XML ist ein zeichenbasiertes Format
  - das Datenmodell ist ein Zeichenstrom
- es gibt viele andere Informationsmodelle
  - DOM1, DOM2, DOM3
  - SAX1, SAX2
  - XML Information Set
  - XPath 1.0, XPath 2.0 (= XQuery 1.0)
- Pipelines verlieren häufig Informationen
  - inkompatible Modelle in der Pipeline

XML 1.0 Pipelines

- Implementierungen
  - beliebige Tools, die XML 1.0 unterstützen
- Kopplung über viele Protokolle möglich
  - zeichenbasiert, also auch Unix Pipes
  - typische lose Kopplung von Komponenten
- extrem ineffizient
  - oft wiederholtes Parsen/Serialisieren
- Probleme mit out-of-band Signalen

SAX Pipelines

- SAX ist ein event-basiertes API
  - keine Datenstruktur, sondern Events
- je nach Aufbau ist Nebenläufigkeit möglich
  - effizientere Ausführung der Pipeline
  - auch SAX kann intern DOM verwenden
- existierende Implementierungen
  - Cocoon, das Apache XML Publishing Framework
  - Jelly, ein weiteres Apache-Projekt

DOM Pipelines

- DOM Bäume sind implementierungsspezifisch
  - DOM definiert nur ein Interface
  - das Speichermodell ist vollkommen offen
- viele Tools basieren auf DOM
  - egal ob XML Dokument oder "synthetisiertes" DOM
  - Pipeline benutzt DOM-Objekt (d.h. Speicherbereich)
- Implementierungen
  - DOM-Tools können kombiniert werden

Heterogeneous Pipelines

- oftmals unterschiedliche Rahmenbedingungen
  - einige DOM-basierte Tools
    - effizient durch DOM-Objekte zu verbinden
  - Anbindung von Business Rules Validation
    - auf externem System implementiert
    - PDOM oder XML Dokument Anbindung
- flexibles Pipeline Framework
  - Komponenten deklarieren ihre Schnittstellen
  - Pipeline Definition basiert auf Kompatibilität

Validation Pipelines

- Spezialfall allgemeiner Processing Pipelines
  - u.U. stärker eingeschränkter Satz an Tools
    - Parser, XML Schema Processor, Schematron, CRVX, ...
- Business Rule Validation ist kompliziert
  - u.U. Anbindung an externe Datenbestände
  - oder allgemein Anbindung an externe Applikation
    - Pipeline Komponente mit einem Validation API
    - kann an beliebigen Code gebunden werden
  - keine Schema-Sprache, aber ein Binding definieren
- noch in Entwicklung befindlich

Sequence Matters

- Validation Pipelines kombinieren Schemas
  - sequentielle Abarbeitung der Pipeline
- die Reihenfolge kann wichtig sein
  - z.B. CRVX und Default-Werte aus einem Schema
- u.U. mehrfache Ausführung von Komponenten
  - Validation als Graph mit Ablaufsteuerung
    - Pipeline-Sprachen werden u.U. komplex
    - siehe Web Services Flow Language (WSFL)

Zusammenfassung

- Data Quality ist wichtig
  - oftmals vernachlässigt
  - Probleme verursachen langfristig hohe Kosten
- Validierung gehört zu einer Anwendung
  - auf verschiedenen logischen Ebenen
  - mit verschiedenen Tools
  - vereinigt in einer gemeinsamen Umgebung
- Entwicklung noch in der Frühphase
  - XML Pipelines sind noch in der Entwicklung
  - Schemasprachen sind im Entstehen

Besten Dank

Fragen?

mailto:net.dret@dret.net

http://dret.net/netdret/

http://dret.net/netdret/publications#fhf-webapps