XML Design Patterns

von
Susanne Katz (sk30)

und
Daniel Brenner (db029)

Für das Seminar „Software Design Patterns“
– Sommersemester 2004 –
Hochschule der Medien (HdM) Stuttgart

Überblick

Anders als bei den objekt-orientierten Design Patterns, die alle etwa den gleichen Abstraktionsgrad haben, gibt es bei den XML Design Patterns unterschiedliche Kategorien:
Generelle Design Prinzipien und spezifische Design Patterns welche im Folgenden erläutert werden.

1. Generelle Design Prinzipien

1.1 Dynamic Dokument

Beschreibung:
Dynamic Dokument ist eine Lösung für XML-Dokumente, bei denen man im Voraus die Struktur nicht kennt oder sich die Anforderungen ändern können. Das XML-Format läßt sich durch automatische Serialisierung generieren. Dieses Verfahren wird in Zusammenhang mit OO-Systemen, wie .NET und Java bereits standartmäßig eingesetzt.

Anwendung:
- Extreme Programming, da man schnell zu Ergebnissen gelangt. Wenn es erforderlich ist kann die XML Datei immer noch umformatiert werden.
- Server-Konfiguraionsdateien (Apache Tomcat server.xml)
- Technische Anleitung
- JDK 1.4 JavaBean XML persistence

Beispiel:
Hier werden die Objekte rekursiv durchlaufen und ein XML-Baum erzeugt.

Aus

```
public Person {
       public String getName() { ... }
       public void setName(String name) { ... }
       public Address getAddress() { ... }
       public void setAddress(Address address) { ... }
    }
    public Address {
       public String getCity() { ... }
       public void setCity(String city) { ... }
       public String getState() { ... }
       public void setState(String state) { ... }
    }
```

wird

```
<person>
      <name>Kyle Downey</name>
      <address>
            <city>Forest Hills</city>
            <state>Queens</state>
      </address>
</person>
```

1.2 Compostion

Beschreibung:
Man sollte vorhandene Namespaces wiederverwenden, d. h. Elemente aus vorhanden Namespaces referenzieren, statt sie selbst zu definieren. Die Voreile bestehen darin, dass man Zeit spart und auf bewährte Modelle zurückgreifen kann. Somit erhält man auch mit weniger Erfahrung ein gutes Modell. Mit diesem Pattern will man sich das Leben so einfach wie möglich machen.

Anwendung:
WSDL – gute Wiederverwendung von XML Schema in dem das ganze <schema> Element eingebettet wird um die Types zu definieren.

Beispiel:
```
<?xml version="1.0" encoding="UTF-8"?>
  <definitions name="ProductService"
   targetNamespace="http://www.ecerami.com/wsdl/ProductService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.ecerami.com/wsdl/ProductService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsd1="http://www.ecerami.com/schema">
    <types>
      <xsd:schema
       targetNamespace="http://www.ecerami.com/schema"
       xmlns="http://www.w3.org/2001/XMLSchema">
        <xsd:complexType name="product">
          <xsd:sequence>
            <xsd:element name="name" type="xsd:string"/>
            <xsd:element name="description" type="xsd:string"/>
            <xsd:element name="price" type="xsd:double"/>
            <xsd:element name="SKU" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:schema>
    </types>

  <message name="getProductRequest">
    <part name="sku" type="xsd:string"/>
  </message>
  <message name="getProductResponse">
    <part name="product" type="xsd1:product"/>
  </message> . . .

</definitions>
```

1.3 Multipart Files

Beschreibung:
Wenn Dokumente zu groß und unübersichtlich werden, kann das Multipart-Files-Pattern angewand werden. Auch bei überlappenden Strukturen kann man es verwenden.
Ähnlich wie das C++ `#include` oder Java `import` kann man sich ein entsprechendes Element definieren, um XML-Dokumente beliebig klein zu stückeln.
Es besteht jedoch das Risiko, dass Cracker referenzierte Inhalte austauschen und/oder nicht berechtigte Personen Zugriff auf Inhalte haben.

Anwendung:
Bei der XML-Programmiersprache XSLT wird das ebenso wie in XML Schemas, WSDL und SOAP (Attachements) eingesetzt.

Beispiel:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE REPORT SYSTEM „report.dtd" [
<!ENTITY copyRight SYSTEM „copyRight.xml"> <!--External General Entity -->
]>
<REPORT>
  <GRAFIK src="zirkel.gif">Unsere Hauptprodukte</GRAFIK>
  <PARA>Und dies ist das Ende des Kurzreports.</PARA>
  &copyRight; <!--Verwendung Entity -->
</REPORT>
```

Eine Entity `copyRight` wird definiert. Die Beschreibung von `copyRight` erfolg in der Datei „copyRight.xml", dies wird durch das Schlüsselwort `SYSTEM` signalisiert. Es handelt sich hier um eine External General Entity.

2. Spezifische Design Patterns

Spezifische Design Patterns entsprechen etwa dem Abstraktionsgrad von OO Patterns.

2.1 Überblick

Die spezifischen Design Patterns werden in sieben Kategorien eingeteilt.

- Document Roots:
  Patterns, die einem helfen herauszufinden, welche/s Element/e als Root fungieren sollen.
- Metadata:
  Sind hilfreich bei der Entscheidung, wo Metadaten im Dokument platziert werden sollen.
- Abstraction:
  Beschreibt wie verschiedene Abstraktionsgrade eingesetzt werden.
- Organization:
  Beschreibt wie die Struktur von Dokumenten gehandhabt wird.
- Flexibility:
  Beschreibt wie man Dokumente mehr oder weniger flexibel hält.
- Consistency
  Beschreibt wie man konsistente Bezeichner (markup) erhält.
- Miscellaneous
  Diverse nicht kategorisierte Patterns.

Genau nachzulesen, mit den dazugehörigen Patterns unter:
www.xmlpatterns.com/categoryAllPatterns.shtml

Im Anschluss wird aus jeder Kategorie ein Pattern vorgestellt.

2.2 Multiple Root Document Types (Document Roots)

Beschreibung:
Es werden mehrere Root-Elemente verwendet um unterschiedliche Dokumente in einem System zu verarbeiten.

Anwendung:
Wird eingesetz wenn man sehr ähnliche Dokumente hat, die anhand einer eindeutigen Identität unterschieden werden. Der Identifizierer ist dabei das Root-Element.

Beispiel:
Die zwei unten angegebenen Dokumente zeigen zwei verschiedene Banktransaktionen (Geld Einzahlung und Entnahme). Dabei sind die Root-Elemente verschieden, jedoch die anderen Elemente gleich.
Beide XML-Dokumente benützen die selbe DTD, aber deklarieren verschiedene Root-Elemente.

```
<!DOCTYPE „Einzahlung“ SYSTEM „bank.dtd“>
<Einzahlung>
      <Kontonr id=“123“/>
      <Betrag Währung=“EUR“>100.00</Betrag>
      <Datum>05-10-2000</Datum>
</Einzahlung>

<!DOCTYPE „Entnahme“ SYSTEM „bank.dtd“>
<Entnahme>
      <Kontonr id=“123“/>
      <Betrag Währung=“EUR“>100.00</Betrag>
      <Datum>05-10-2000</Datum>
</Entnahme>
```

2.3 Head-Body (Metadata)

Beschreibung:
Vielleicht ist dies das bekannteste Design Pattern in XML. Es kommt dann zum Einsatz, wenn man viele Metadaten zu einem Dokument hat und diese strukturieren möchte.

Anwendung:
Es wird z.B. in (X)HTML und SOAP verwendet.

Beispiel:
Die Metadaten für das Dokument kommen in das `<Head>`-Element und die Daten, die den Inhalt beschreiben kommen in das `<Body>`-Element

```
<Document>
  <Head>
    <Author>John Doe</Author>
    <Author>Frank Black</Author>
    <CreationDate>June 16, 1999
    </CreationDate>
  </Head>
  <Body> This is a document.
  </Body>
</Document>
```

2.4 **Container Element** (Abstraction)

Beschreibung:
Daten werden eher hierarchisch anstatt flach strukturiert. Die Verarbeitung des Dokuments wird effektiver.

Anwendung:
Sollte in allen XML-ähnlichen Dokumenten angewand werden, jedoch nur dann, wenn die Daten klar strukturiert werden können.

Beispiel:

Aus

```
<ComputerConfiguration>
      <RAM>128 MB</RAM>
      <WordProcessor>WordPerfect</WordProcessor>
      <HardDriveSize>8GB</HardDriveSize>
      <XMLParser>Xerces</XMLParser>
</ComputerConfiguration>
```

wird dann

```
<ComputerConfiguration>
      <Software>
            <Wordprocessor>Wordperfect</Wordprocessor>
            <XMLParser>Xerces</XMLParser>
      </Software>
      <Hardware>
            <RAM>128 MB<RAM>
            <HardDriveSize>8 GB</HardDriveSize>
      </Hardware>
</ComputerConfiguration>
```

2.5 **Marketplace** (Organization)

Beschreibung:
Das Problem ist, dass manche Daten nicht exakt kategorisiert werden können (siehe Beispiel). Es wird dadurch gelöst, in dem man die Daten auf einer gemeinsamen Ebene gruppiert und die Kategorisierung im Element anhand von Attributen durchführt.

Anwendung:
Kann in allen XML-ähnlichen Dokumenten angewand werden, sobald Schwierigkeiten mit der Kategorisierung der Daten auftreten.

Beispiel:
Genauso wie man sich beim Design von Tabellen für relationalen Datenbanken nicht entscheiden kann, ob ein Manager nun ein Manager oder ein Angestellter der Firma ist, ist es unmöglich zu entscheiden, ob das `<Manager>`-Element in `<Managers>` oder in `<Employees>` sein sollte.

```
<Employees>
      <Managers>
            <Person type="salaried"> Riff Raff </Person>
            <Person type="contractor"> Frank Furter </Person>
      </Managers>
      <Workers>
            <Person type="salaried"> Brad Majors </Person>
            <Person type="contractor"> Janet Weiss </Person>
      </Workers>
</Employees>
```

Dementsprechend gibt es in XML auch die gleichen Lösungsansätze für dieses Problem. Die einfachste ist eine flache Liste, in die alle Angestellten eingetragen sind, mit ihrem Beschäftigungsstatus als Attribut.

```
<Employees>
      <Person type="salaried" level="manager"> Riff Raff </Person>
      <Person type="contractor" level="manager"> Frank Furter </Person>
      <Person type="salaried" level="worker"> Brad </Person>
      <Person type="contractor" level="worker"> Janet </Person>
</Employees>
```

2.6 **Role Attribute** (Flexibility)

Beschreibung:
Meistens können die Designer einer DTD nicht alle Bedürfnisse der Dokumentautoren vorhersehen. Um dem Autor mehr Flexibilität zu ermöglichen bekommt das Element ein Attribut, dass als Platzhalter fungiert.

Anwendung:
Zum Beispiel in XHTML werden bei den Elementen `class` und `meta` Role Attribute verwendet.

Beispiel:
Ein Role-Attribut wird deklariet, welches die Benutztung des Elements weiter spezialisiert.

```
<!ELEMENT Document (Paragraph)*>
<!ELEMENT Paragraph (#PCDATA)>
<!ATTLIST Paragraph Role CDATA>
```

Durch die Verwendung der obigen DTD kann man folgendes Dokument erzeugen.

```
<Document>
      <Paragraph Role="warning">
            Do not attempt this at home.
      </Paragraph>
      <Paragraph>
            The following is a set of instructions for creating a
            bungee jumping cord from elastic bands ...
      </Paragraph>
</Document>
```

2.7 **Consistent Element Set** (Consistency)

Beschreibung:
Bei Consitent Element Set werden wiederkehrende Elementinhalte in einer Entity ausgelagert, um die Lesbarkeit und das Verständnis der DTD zu erleichtern. Ähnliche Elementinhalte werden zu einem konsistenten Elementinhalt zusammengefasst. Dabei muss man Kompromisse eingehen, die die Details der Struktur betreffen, aber es lohnt sich für gewöhnlich diese einzugen.
Außerdem wird dadurch die Verarbeitung der XML-Daten über Software einfacher.

Anwendung:
Zum Beispiel in XHTML sind Element-Sets definiert für block-content (zur Benutzung auf paragraph level) und inline content (wird innerhalb der paragraphen verwendet).

Beispiel:

DTD mit inkonsistenten Elementinhlaten (unübersichtlich).

```
<!ELEMENT Foo (A|B|C|E)*>
<!ELEMENT Bar (A|B|C|D)*>
<!ELEMENT Baz (A|C|D|E)*>
```

DTD mit konsistenten Elementinhlaten (einfacher).

```
<!ENTITY % content „A|B|C|D|E“>
<!ELEMENT Foo (%content;)*>
<!ELEMENT Bar (%content;)*>
<!ELEMENT Baz (%content;)*>
```

Quellen:

www.xml.com
www.xml.com/pub/a/2002/11/20/schemas.html
www.xml.com/pub/a/2003/03/26/patterns.html
www.xmlpatterns.com
www.utoronto.ca/ian/books/xmlbook/patterns.html
www.informatik.hu-berlin.de/~obecker/Lehre/SS2003/XML/gz/designpatterns.htm