Java-Codegenerator für DTDs

Robert Lechner

2. April 2004

1 Einleitung

Um die Struktur einer XML-Datei zu beschreiben wird ein DTD (Document Type Definition) verwendet. Zur weiteren Bearbeitung der Daten aus der XML-Datei ist es sinnvoll, dass der XML-Parser eine Datenstruktur liefert, die der Beschreibung im DTD entspricht. Das Programm `dtd2java` liest nun ein DTD und erzeugt daraus eine einfache Datenstruktur, sowie den entsprechenden XML-Parser.

Das erzeugte Modell orientiert sich ausschließlich am DTD; es ist also kein Ersatz für ein von Hand optimiertes Metamodell. Da das Erstellen eines solchen Modells aber sehr aufwendig ist, stellt `dtd2java` eine Möglichkeit dar ohne großen Zeitaufwand einen XML-Parser (mit Datenstruktur) zu erhalten.

2 Übersicht

Das Paket `dtd2java` besteht aus zwei Teilen: einem Parser für das DTD und einem Codegenerator, der aus dem DTD einen XML-Parser mit einem einfachen Modell erzeugt. In Abbildung 1 sind die verwendeten Klassen dargestellt.

2.1 DTD-Parser

Für den DTD-Parser wird der Parsergenerator ANTLR [1] benötigt.
Zum Lesen eines DTD ruft man die statische Methode `DtdParser.parseFile` auf. Man erhält eine Instanz von `DtdFile`, welche alle Elemente des DTD enthält. Entities werden automatisch expandiert; Attributlisten werden zusammengefasst.

[1] ANTLR Translator Generator (http://www.antlr.org)

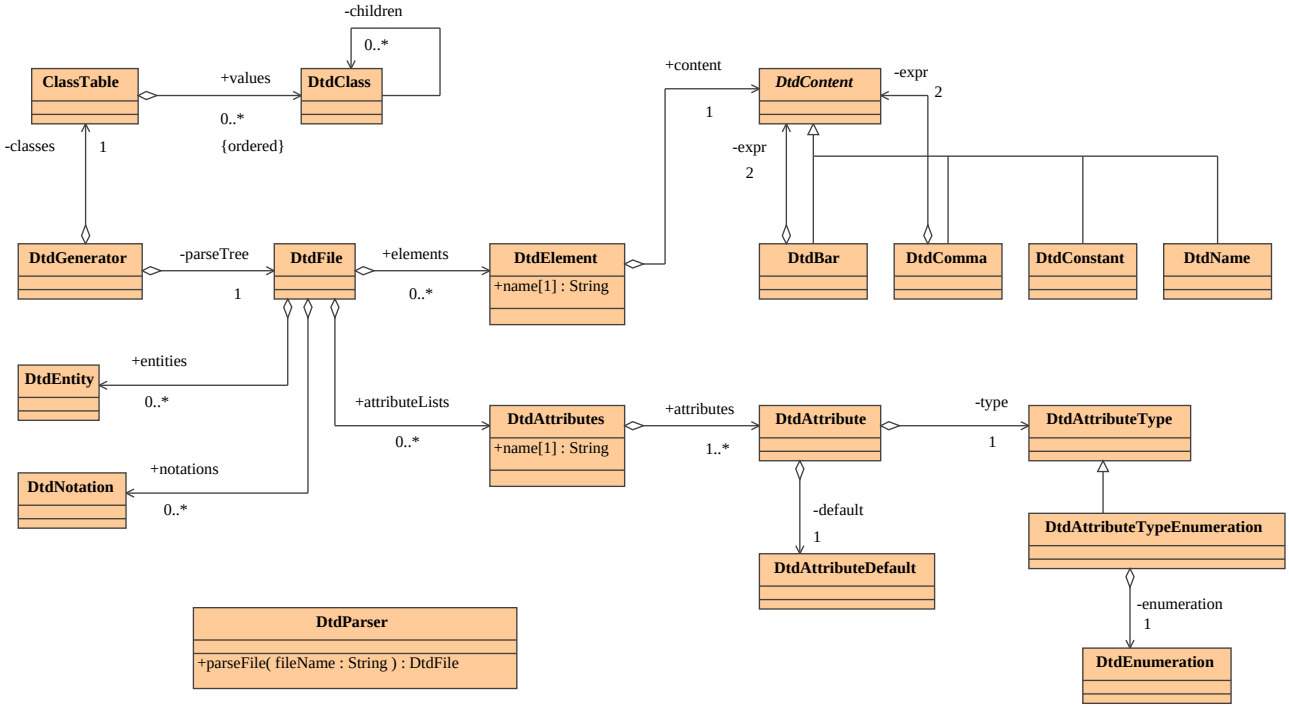

Abbildung 1: Übersicht über das Paket `dtd2java`

2.2 Codegenerator

2.2.1 Aufruf

Es gibt zwei Möglichkeiten den Generator zu starten:

- Erzeugen einer Instanz von `DtdGenerator` und Aufruf der Methode `run`. Man benötigt dazu aber eine Instanz von `DtdFile` (siehe oben).
- Man startet den Generator als Programm:
  `java dtd2java.Main DTD-file Java-package [root-directory]`

Der Parameter `Java-package`, den man in beiden Fällen benötigt, ist der Name jenes package, in welchem der erzeugte Code liegen soll. Das Paket wird im aktuellen oder im angegebenen (`root-directory`) Verzeichnis angelegt.

2.2.2 Eigenschaften

Der generierte Code hat die folgenden Eigenschaften:

- Für jedes Element im DTD (`<!ELEMENT name content >`) wird eine eigene Klasse generiert. Alle Attribute sind als öffentliche Variablen direkt zugänglich. Der Inhalt des Elements wird in einem allgemeinen Container gespeichert.
- DTD-Namespaces werden als Java-Packages dargestellt.

- Der gesamte generierte Code ist `javadoc`-kompatibel dokumentiert.
- Es werden alle benötigten Klassen im angegebenen Paket generiert ⇒ man benötigt zum Ausführen des erzeugten Codes keine weiteren Packages (ausgenommen der Java 1.4 Library).

Zusätzlich werden Dateien für das Tool dot [2] erzeugt.

2.2.3 erzeugter Code

Alle Klassen sind von `DTD_Container` abgeleitet. Über die Methoden `size` und `get` ist der direkte Zugriff auf den Inhalt möglich. Mit der öffentlichen Variable `parent__` kann auf den Vorgänger im Baum zugegriffen werden.

In den Unterklassen wird für jedes Attribut des XML-Elements eine öffentliche Variable erzeugt. In der statischen Variable `xmlName__` steht der vollständige XML-Name des Elements. Mit der Methode `xmlCode` kann der XML-Code erzeugt werden.

Alle Elemente, welche nicht im DTD deklariert wurden, werden als Instanz von `DTD_Generic` gespeichert. Dadurch ist eine Rekonstruktion des XML-Codes auch bei unbekannten Elementen möglich.

Um die Erweiterbarkeit des erzeugten Modells zu gewährleisten, werden alle Instanzen über eine Factory (`DTD_Creator`) erzeugt. Zur Demonstration dient das folgende Beispiel:

XML-Element:

```
package test_model;
import org.xml.sax.*;

public class E9 extends DTD_Container
{
    ...
}
```

[2] Graphviz (http://www.graphviz.org)

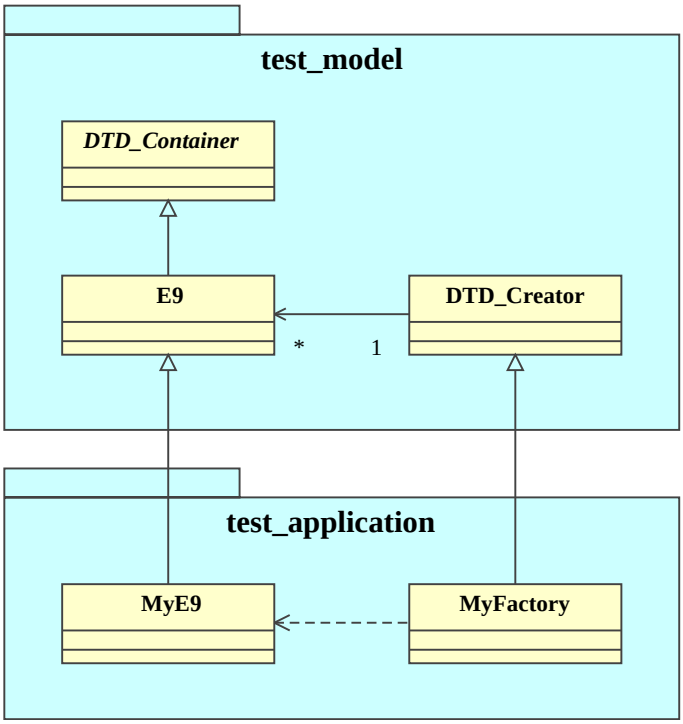

Abbildung 2: Erweiterung des generierten Modells

Factory:

```
package test_model;
import org.xml.sax.*;

public class DTD_Creator
{
  public DTD_Container create( String qName, Attributes attrs )
  {
    ...
    if(qName.equals("E9"))  return new E9(attrs);
    ...
    return new DTD_Generic(qName, attrs);
  }
}
```

eigenes Element:

```
package test_application;
import test_model.*;
import org.xml.sax.*;

public class MyE9 extends E9
{
    public MyE9( Attributes attrs )
    {
        super(attrs);
    }

    ...
}
```

eigene Factory:

```
package test_application;
import test_model.*;
import org.xml.sax.*;

public class MyFactory extends DTD_Creator
{
    public DTD_Container create( String qName, Attributes attrs )
    {
        if( qName.equals(E9.xmlName__) )
        {
            return new MyE9(attrs);
        }
        return super.create(qName, attrs);
    }
}
```

Der erzeugte Parser wird mit der statischen Methode `DTD_Root.parse` aufgerufen. Dabei kann optional auch eine eigene Factory übergeben werden. Im obigen Beispiel würde der Aufruf wie folgt lauten:

```
java.io.File xmlFile = new java.io.File( ... );
DTD_Container root = DTD_Root.parse( xmlFile, new MyFactory() );
```

3 Klassen

Es folgt eine Übersicht über alle automatisch erzeugten Klassen (ohne die Klassen der DTD-Elemente). Die Klassen der DTD-Elemente sind von `DTD_Container` abgeleitet.

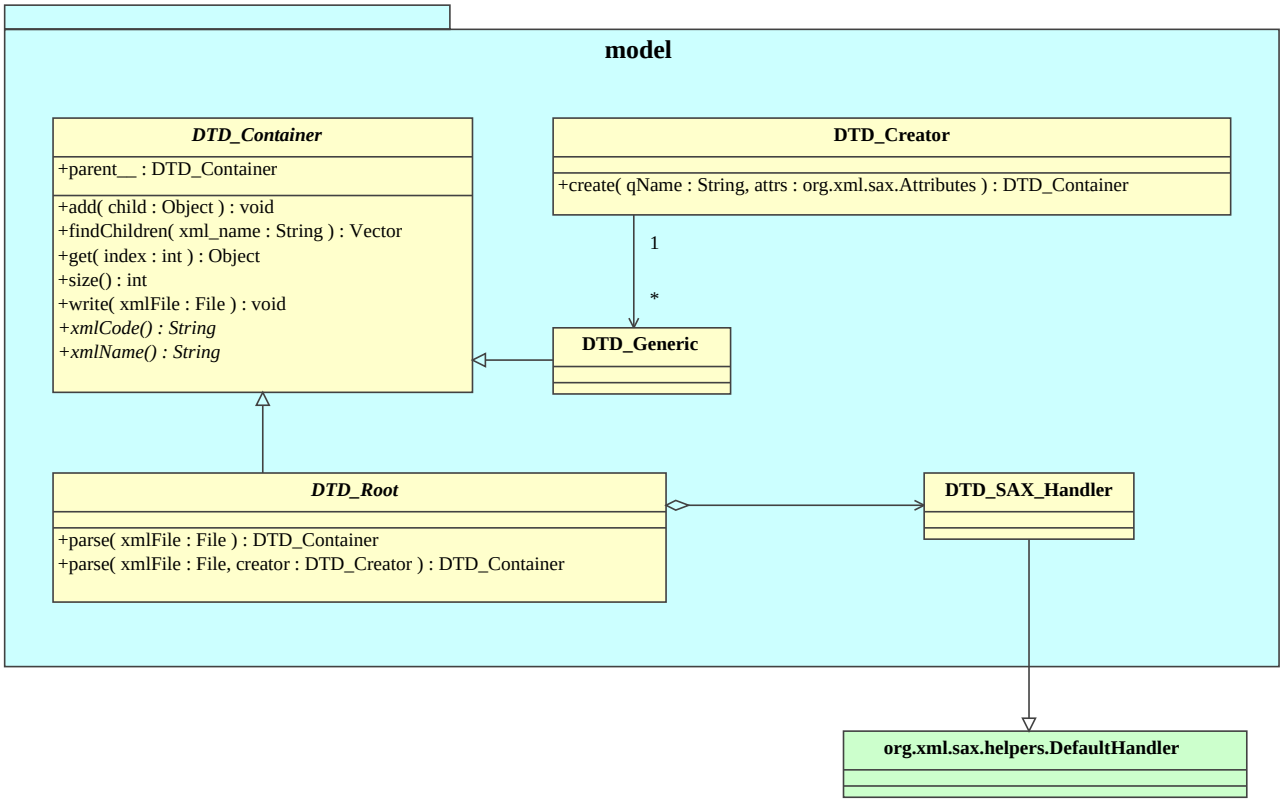

Abbildung 3: alle automatisch erzeugten Klassen