Chameleon: an Extensible and Customizable Tool for Web Data Translation

Riccardo Torlone Paolo Atzeni

DIA – Università Roma Tre
Via della Vasca Navale, 79
00146 Roma, Italy
{torlone,atzeni}@dia.uniroma3.it

Abstract

Chameleon is a tool for the management of Web data according to different formats and models and for the automatic transformation of schemas and instances from one model to another.

It handles semistructured data, schema languages for XML, and traditional database models. The system is based on a "metamodel" approach, in the sense that it knows a set of metaconstructs, and allows the definition of models by means of the involved metaconstructs. The system also has a library of basic translations, referring to the known metaconstructs, and builds actual translations by means of suitable combinations of the basic ones.

The main functions offered to the user are: (i) definition of a model; (ii) definition and validation of a schema with respect to a given model; (iii) schema translation (from a model to another).

1 Introduction

The Internet and the World-Wide-Web are now encouraging many initiatives for data cooperation and interchange between sources that share little or no advance standardization. As a consequence, many formats are often available, including XML, relational or object models, semistructured data models (Abiteboul et al. [1]). Data is also often described at the design level, within the various systems, using yet other formalisms, such as variants of the ER model or subsets of UML.

Therefore the need arises for an integrated management of data descriptions that allow for easy and flexible translation from a model to another (Bernstein et al. [4]). Here we briefly present *Chameleon*, a tool for the management of Web data described according to a variety of formats and models and the automatic translation of schemas and instances from one model to another. The set of models managed by Chameleon includes the majority of the formats used to represent data in Web-based applications: semi-structured models, schema languages for XML, and traditional conceptual data models. The tool builds up on our previous experience on the management of database models (Atzeni and Torlone [2]) and on a preliminary proposal for the management of schema description languages for XML (Torlone and Atzeni [6]). The tool tries to give a contribution in the field of model management, by implementing a form of the "ModelGen" operator proposed by Bernstein [3].

A distinctive feature of Chameleon, which is not provided by other systems, is that the set of models is not fixed a priori. A new model M can be defined by the user at run-time and translations for M are derived by the system with very limited user intervention. The tool relies on a notion of *metamodel* that embeds, on the one hand, the main primitives adopted by different schema languages for XML (Lee and Chu [5]) and, on the other hand, the basic constructs of traditional database conceptual and logical models. Specifically, the metamodel of Chameleon is made of a set of *metaprimitives*, each of which captures some basic abstraction principle used in a formalism for modelling Web data. Examples of metaprimitives are: class, attribute, base type, sequence, relationship, disjoint union, key, foreign key, and so on. In this framework, a model can be defined by specifying the metaprimitives of the metamodel it adopts together

Proceedings of the 29th VLDB Conference, Berlin, Germany, 2003

with possible limitation in their use (e.g., only binary relationships).

Translations between models are automatically derived in Chameleon by combining a set of predefined and standard basic translations. These include, for instance, the translation of a class into a relational table and the translation of a generalization hierarchy into a set of relationships. When different translations for a given model are possible, the user is asked to choose the preferred one. These translations allow the users to convert schemas between any of the defined models.

An important concept here is the *supermodel*: this is the "most general model," that is, the model that includes all possible constructs. It is instrumental in the definition of translations, because each schema is compatible with the supermodel, and so any translation from a source model to a target model can be seen as a translation from the supermodel to the target model. In this way, if we have translations from the supermodel to a target model, then we know we have translations from any model such a target model.

Schema translations can be analyzed step-by-step and the system identifies steps that can be source of "anomalies" (Torlone and Atzeni [6]). Anomalies occur when a primitive of the source model cannot be translated properly in the target model. More specifically, during the translation of a primitive p in a schema S of a source model M_s into a target model M_t, two undesirable cases can occur. The first situation occurs when there is not any primitive of the same type in M_t, for instance, when we need to translate an unordered sequence (e.g., of an XML schema) into a model that only admits ordered sequences (e.g., a DTD). In this case the system makes use of a (combinations of) primitive(s) of M_t to implement the missing primitive. In the example above, the unordered sequence is transformed into an ordered one by imposing an order. This is an anomaly that we call *loss* of (meta) information. The second situation occurs when there is a primitive p' in M_t of the same type of p but in a restricted form. For instance, when we need to translate a very detailed cardinality (e.g., a cardinality 1:30 of an XML schema) into a model that only admits prefixed combinations of cardinalities (e.g., into a DTD that only allows cardinalities 0:1 (?), 1:1 (the default), 0:N ($*$) and 1:N ($+$)). In this case the system transforms, in the more convenient way, p into p'. In the example above the cardinality 1:30 is transformed into 1:N. This is an anomaly that we call *degradation* of (meta) information.

2 The tool

Chameleon offers the following main functions.

1. *Definition of a new model*. A new model M is defined by specifying:
   (a) the metaprimitives of the metamodel used in M,
   (b) possible limitations in the use of such metaprimitives, and
   (c) the names used in M to denote such metaprimitives (e.g., a class of objects is called *entity* in the ER model and *element* in a DTD).

   A wizard supports the user in the definition of models.

   At the end, the system verifies whether the set of primitives of M is coherent and, if so, it derives a *default translation* for M. This is a translation from the supermodel to M. User intervention can be needed in this phase if several valid alternatives for the translation of a primitive are possible.

2. *Validation of a schema with respect to a given model*. A textual mark-up representation of a schema S can be opened and validated against a previously defined model. If S is valid the system automatically generates an *abstract schema* for S, that is, an XML internal representation of S in terms of the metamodel.

3. *Schema Translation*. A valid schema S in a model M can be translated into any other available model M'. The translation operates on the abstract representation of S and produces another abstract schema S' which makes use only of the metaprimitives available in M'. The final step consists in a renaming of the primitives of S' using the specific syntax adopted by M'.

4. *Translation Analysis*. A detailed log of a schema translation can be accessed by the user to analyze the basic transformations performed over the source schema. Steps that can be cause of anomalies (see Section 1) are highlighted in the report.

5. *Translation Personalization*. A default translation can be modified at any time by changing single steps or adding new ones.

6. *Source look-up*. From a schema obtained as a result of a translation, the user can access at any time the schema it originates from.

7. *A GUI for building and drawing conceptual schemas*. In order to facilitate the management of conceptual schemas the system provides a GUI that allows the users to manipulate diagrammatic representations of conceptual schemas using a ER style.

The tool is fully written in Java and makes use of the JDOM package for the management of XML documents. A screen shot of Chameleon is reported in

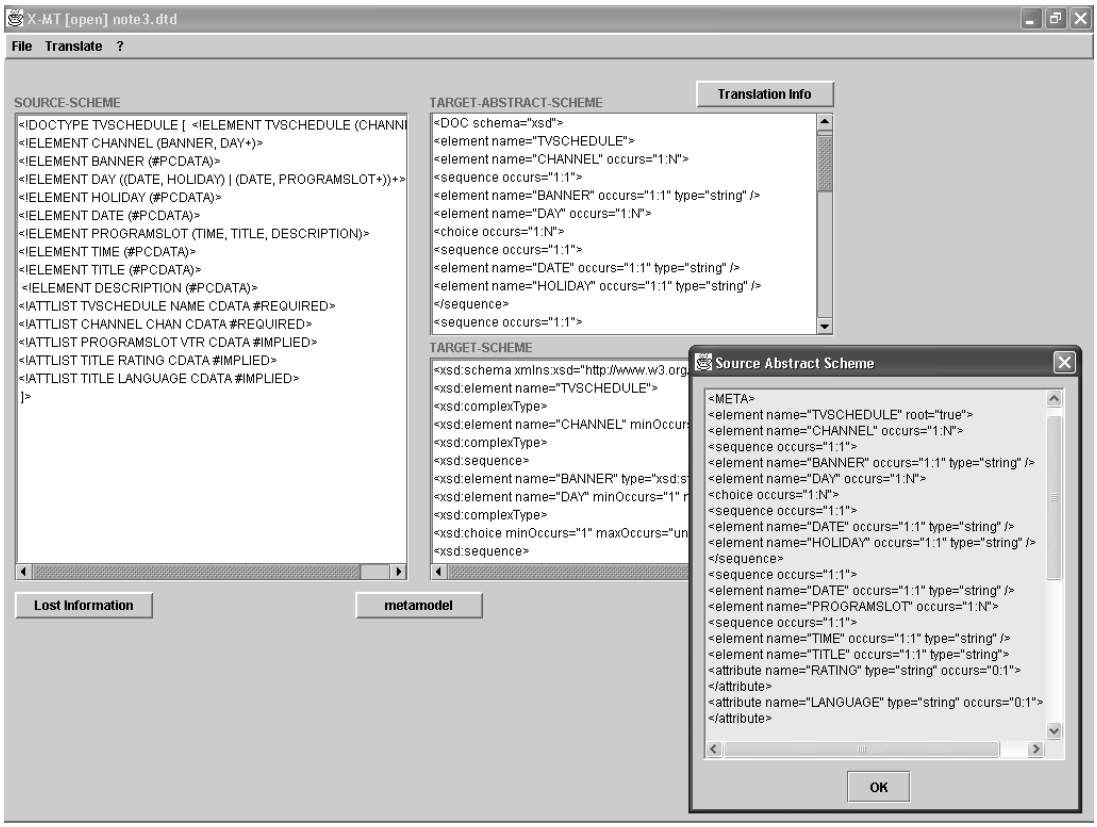

Figure 1: The user interface of Chameleon

Figure 1 that shows the translation of a DTD into an XML schema. The source schema appears on the left of the screen and the related abstract schema in a separate window. The two versions of the target schema are reported in the right side of the main window.

3 An example translation

We have successfully tested our tool with several data models for Web data: four XML schema languages (DTD, XML Schema, DCD and XDR), the ER model, chosen as a representative of common conceptual modela, and the relational model, chosen as a "traditional" data model.

In order to give an intuition of the translation process, let us consider the `Order` XML Schema reported in Figure 2. The corresponding abstract schema is reported in Figure 3: each of the XML primitives in Figure 2 has been converted to the corresponding metaprimitive. For instance, the primitive *all* of XML Schema has been turned into the metaprimitive *unordered sequence*.

Figure 4 shows the abstract schema produced by Chameleon as the translation of the schema of Figure 2 into the DTD model. The final target schema is reported in Figure 5.

Note that the unordered sequence used to define the structure of the element *destination* of the source has been transformed into an ordered sequence, since

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Order" type="OrderType"/>
 <xsd:complexType name="OrderType">
  <xsd:sequence>
   <xsd:element name="destination" type="USAddress"/>
   <xsd:element name="items"  type="Items"/>
  </xsd:sequence>
  <xsd:attribute name="orderDate" type="xsd:date"/>
 </xsd:complexType>
 <xsd:complexType name="USAddress">
  <xsd:all>
   <xsd:element name="street" type="xsd:string"/>
   <xsd:element name="city"   type="xsd:string"/>
   <xsd:element name="zip"    type="xsd:decimal"/>
  </xsd:all>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
 </xsd:complexType>
 <xsd:complexType name="Items">
  <xsd:sequence>
   <xsd:element name="item" minOccurs="0" maxOccurs="10">
    <xsd:complexType>
     <xsd:sequence>
      <xsd:element name="productName" type="xsd:string" />
      <xsd:element name="quantity" type="xsd:integer" />
      <xsd:element name="USPrice"  type="xsd:decimal"/>
     </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
  </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

Figure 2: An XML Schema

```
<META source="xsd">
 <element name="Order" type="OrderType">
  <sequence cardinality="1:1">
   <element name="destination" type="USAddress" cardinality="1:1">
    <unorderedSequence cardinality="1:1" >
     <element name="street" type="string" cardinality="1:1" />
     <element name="city" type="string" cardinality="1:1" />
     <element name="zip" type="decimal" cardinality="1:1" />
    </unorderedSequence>
    <attribute name="country" type="string" cardinality="0:1">
     <fixed>US</fixed>
    </attribute>
   </element>
   <element name="items" type="Items" cardinality="1:1">
    <sequence cardinality="1:1">
     <element name="item" cardinality="0:10">
      <sequence cardinality="1:1">
       <element name="productName" type="string" cardinality="1:1" />
       <element name="quantity" type="integer" cardinality="1:1" />
       <element name="USPrice" type="decimal" cardinality="1:1" />
      </sequence>
     </element>
    </sequence>
   </element>
  </sequence>
  <attribute name="orderDate" type="date" cardinality="0:1" />
 </element>
</META>
```

Figure 3: The abstract representation of the schema in Figure 2

```
<META source="xsd" target="dtd">
 <element name="Order" root="true">
  <sequence cardinality="1:1">
   <element name="destination" cardinality="1:1">
    <sequence cardinality="0:N">
     <element name="street" type="string" cardinality="1:1" />
     <element name="city" type="string" cardinality="1:1" />
     <element name="zip" type="string" cardinality="1:1" />
    </sequence>
    <attribute name="country" type="string" cardinality="0:1" >
     <fixed>US</fixed>
    </attribute>
   </element>
   <element name="items" cardinality="1:1">
    <sequence cardinality="1:1">
     <element name="item" cardinality="0:N">
      <sequence cardinality="1:1">
       <element name="productName" type="string"  cardinality="1:1" />
       <element name="quantity" type="string" cardinality="1:1" />
       <element name="USPrice" type="string" cardinality="1:1" />
      </sequence>
     </element>
    </sequence>
   </element>
  </sequence>
  <attribute name="orderDate" type="string" cardinality="0:1" />
 </element>
</META>
```

Figure 4: The translation of the schema in Figure 3

```
<!DOCTYPE Order[
  <!ELEMENT Order (destination,items)>
  <!ELEMENT destination (street,city,zip)>
  <!ELEMENT street (#PCDATA)>
  <!ELEMENT city (#PCDATA)>
  <!ELEMENT zip (#PCDATA)>
  <!ELEMENT items (item*)>
  <!ELEMENT item (productName,quantity,USPrice)>
  <!ELEMENT productName (#PCDATA)>
  <!ELEMENT quantity (#PCDATA)>
  <!ELEMENT USPrice (#PCDATA)>
  <!ATTLIST Order orderDate CDATA #IMPLIED>
  <!ATTLIST destination country CDATA #FIXED "US">
  ]>
```

Figure 5: The target schema of the translation

unordered sequences are not representable by a DTD. Moreover, the cardinality of the element *item* has been transformed from 0:10 to 0:N, since in a DTD the only allowed cardinalities are 0:1 (?), 0:N (*), and 1:N (+). Finally, all the base types have been changed to strings, since they are the only base type available in a DTD. These are the two cases we referred to in the introduction as "loss" and "degradation".

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann, 1999.

[2] P. Atzeni and R. Torlone. Management of Multiple Models in an Extensible Database Design Tool. In *Fifth International Conference on Extending Database Technology (EDBT '96), Lecture Notes in Computer Science 1057*, Springer–Verlag, pag. 79–95, 1996.

[3] P. A. Bernstein. Applying Model Management to Classical Meta Data Problems. *CIDR 2003* (available at `http://www-db.cs.wisc.edu/cidr/`)

[4] P. A. Bernstein, A. Y. Levy, and R. A. Pottinger. A Vision for Management of Complex Models. *SIGMOD Record*, 29(4):55–63, December 2000.

[5] D. Lee and W. W. Chu. Comparative Analysis of Six XML Schema Languages. *SIGMOD Record*, 29(3):76–87, 2000.

[6] R. Torlone and P. Atzeni. A Unified Framework for Data Translation over the Web. In *Second International Conference on Web Information System Engineering (WISE 2001), Kyoto, Giappone*, IEEE Computer Society Press, pages 350–358, 2001.