Migrating from XML DTD to XML-Schema using UML

Rational Software White Paper

Table of Contents

Introduction

Today, in developing XML-enabled applications there is a need to ensure that XML documents are constrained to a specific application-defined schema. With the XML 1.0 recommendation from the World Wide Web Consortium (W3C) we are provided the Document Type Definition (DTD) schema language. While the XML 1.0 DTD technology offers a solution today, there is no doubt that the upcoming XML-Schema language will play an important role in the future of XML and will likely eventually replace XML DTDs in the mainstream.

Hence the dilemma: should you invest in an XML DTD solution today or wait for the XML-Schema specification to be finalized? For those who have already invested in XML, the issue is more about protecting their investment by successfully migrating from XML DTDs to XML Schemas at an appropriate time.

This paper outlines one solution to this issue by demonstrating a set of rules developed to automate the generation of W3C XML-Schema from a Unified Modeling Language (UML) model representing the contents of an XML DTD. This white paper also outlines the modeling of W3C XML 1.0 DTD schemas using the UML and provides an overview of related functionality provided in Rational Rose. The paper assumes familiarity with the UML language, XML, XML DTDs and introduces W3C XML-Schema.

The technology described here has been released by Rational Software in the form of an open-source Rational Rose sample script. We encourage customers and partners to modify the script for local needs and experimentation, posting any updates back to the author with enhancements that can be of general value. THIS SOFTWARE IS NOT RELEASED AS A SUPPORTED PRODUCT FROM RATIONAL. The sole intention is to offer it as an early technology to gain experience in the use of UML to model XML Schemas.

This work also builds upon a previous white paper, co-authored with CommerceOne that describes an approach to modeling SOX based XML schemas. That paper did not go so far as to describe a UML Profile as it is considered a tactical effort, SOX as a technology is also expected to be superceded by XML-Schema. Once XML-Schema recommendation is finalized by the W3C, it is the intent of Rational Software to update the UML Profile described here to target XML-Schema directly and obviate the need for such a conversion script.

The Vision for UML Modeling

The UML supports the analysis, design, visualization and development of complex systems. These systems do not necessarily need to be software systems. The UML is also being successfully used for hardware design and business process engineering. The vision is to combine, in a single model or related set of models, the whole system analysis and design from the largest architectural component to the smallest code artifact.

To do this, we need a framework that traces from a common analysis model to the specific artifacts in the differing technology models. For example, a “Customer” class in the analysis model may be reified in one way to make a Java class or Enterprise JavaBean; it may be reified in a different form as a set of elements in a DTD model and finally reified yet again in another form for a set of relation database tables for our data model. This common analysis model allows changes to e replicated across the different design models as they are made. It also allows us to trace from code artifact, through analysis, all the way up to a requirements model..

Unifying Your Development Team

Have you ever worked on a software project where the software engineers did not communicate with the DBA or IT architects? Perhaps one side simply did not understand the other side, due to different terminology, vocabulary and skills?

The UML is slowly eroding many of these traditional barriers as business analysts, infrastructure and IT engineers, data modelers, web architects and software developers share a common toolset. By utilizing a common language, the UML, across different engineering domains (Java, C++, Web, XML, Data modeling…) and different business domains (e-commerce, ERP, systems engineering…) you gain a more productive and also more inclusive team. In this document, we describe the effort to include XML developers in the unified project team by extending the UML to cover modeling XML DTD structures.

The UML Extension Mechanisms

The UML provide a standard, built in mechanism for extensions in support of new domains. Specifically, four related mechanisms - constraint, tagged value, stereotype and profile – are provided.

A Profile is a packaging construct that provides the namespace for the extension and contains the other elements. A stereotype is a way of introducing a new modeling construct into the UML. For example, a “class”, as it is usually thought of in software terms, can be stereotyped “table” to create a construct to model databases. A stereotype can be applied to almost any existing UML element, including relationships between elements. A tagged value is a new “attribute” - a name, type, default value etc, that is usually associated with a stereotype. A constraint is a way of specifying additional semantics on these newly introduced profile elements. For example, for an “XML Element” you might introduce a constraint that says it cannot “contain” Java elements as containing Java elements would be meaningless.

XML DTD Modeling using the UML

So why model a DTD using the UML when there are a number of very good XML design tools on the market?

We described above the general reasons why modeling using the UML was a good idea, now let us consider a concrete example. A web browser calls a Java Servlet, that uses JDBC to extract some data from a database,. The Servlet formats that data as XML (according to a known DTD) which it then sends to the client. The question is this: How do we ensure that the data model, Servlet and DTD are in sync? Clearly, a simple answer is to use the same tool to develop all three!

Another important reason is that today’s tools tend to focus on a single DTD at a time, which makes it hard to model dependencies between, or reuse across, DTDs.. In a UML tool which has more advanced modeling capabilities that include packages and namespacing, it is easier to model two or more DTDs side by side, to reuse and understand the relationship between DTDs. Also the type model of the UML lends itself well to modeling XML structures, extending the relatively weak type model of a DTD in such a way that migration to XML-Schema becomes more valuable.

The XML-DTD UML Profile

To support UML based XML DTD modeling in Rational Rose in the 2000e release (May 2000) we have added the stereotypes shown below. We will not describe all the tagged values here as many have been introduced primarily to support automated processing of XML DTDs in the context of UML modeling tools. Those that are important in order to develop a better overall understanding will be described later in the document.

It is important to note that although we discuss this profile in the context of Rational Rose, this profile is not specific to any tool and should be implementable by any UML tool that supports the standard UML Extensibility mechanisms described above.

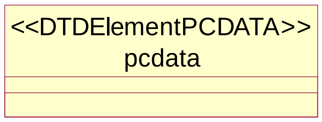
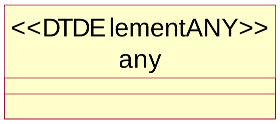
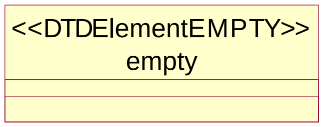
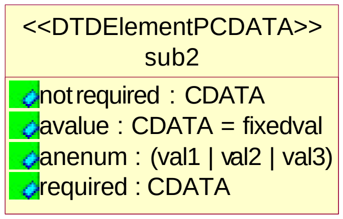
Stereotype	**UML Element**	**Description**
DTDElement	Class	Represents a DTD ELEMENT with either a element-only or mixed content model.
DTDElementANY	Class	Represents a DTD ELEMENT with an open content model, i.e. this DTD does not specify the content model for this element.
DTDElementEMPTY	Class	Represents a DTD ELEMENT with an empty content model (may still have attributes, simply no child elements).
DTDElementPCDATA	Class	Represents a DTD ELEMENT with no child elements, simply text (may still have attributes, simply no child elements).
DTDGroup	Class	Not used directly by the tool, this is a parent if the two following stereotypes.
DTDChoiceGroup	Class	Represents a group in a DTD separated by commas, such as (a, b).
DTDSequenceGroup	Class	Represents a group in a DTD separated by a bar, such as (a | b).
DTDNotation	Class	Represents a DTD notation, this is a relatively simple construct and all details are stored.
DTDEntity	Class	Represents a DTD entity, an entity acts as a “macro” and can be expanded in-line anywhere in the rest of the DTD. It has a number of specific forms and the tool stores these details in tagged values.

Mapping a DTD to UML

We described the UML Profile earlier, but what does that actually look like? How does the UML get used? What can I expect when I reverse engineer a DTD?

The following sections describe the mapping in more detail with examples from a DTD and the resulting UML construct.

Elements & Attributes

DTD Example Source	UML Construct
<!ELEMENT pcdata (#PCDATA)>	
<!ELEMENT any ANY>	
<!ELEMENT empty EMPTY>	
<!ELEMENT empty EMPTY> <!ATTLIST empty notrequired CDATA #IMPLIED avalue CDATA #FIXED 'fixedval' anenum (val1 | val2 | val3) #IMPLIED required CDATA #REQUIRED >	 <<DTDElementPCDATA>> sub2 notrequired : CDATA avalue : CDATA = fixedval anenum : (val1 | val2 | val3) required : CDATA

Groups, Order & Cardinality

The grouping constructs seen below should be fairly self-evident. The one exception to this is that the groups are modeled as nested classes (inner classes in Java terms) within the parent element. We will clarify the rationale for this representation shortly.

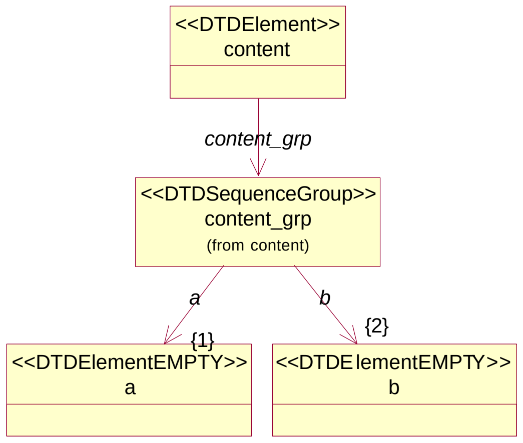
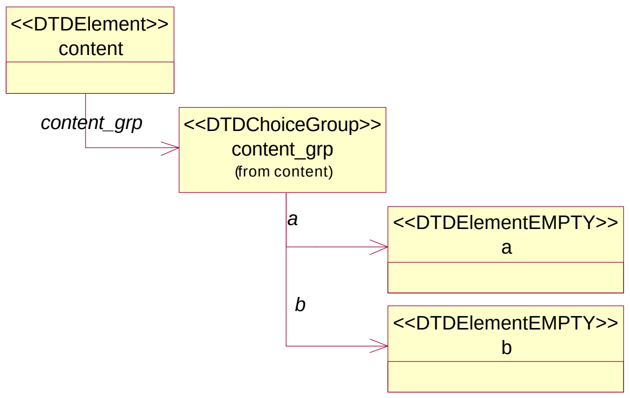
The more interesting constructs are those showing cardinality in the third example, the UML syntax being a bounding expression of the form "lower upper"; so "?" becomes "0..1", "+" becomes "1..*" and "*" becomes "0..*". Finally elements referenced from a sequence group have an absolute order. An instance document validated against a DTD must ensure this order and so the tool must ensure the same order when it forward engineers *from* the model or as it reverse engineers *into* the model. To do this we use UML constraints (expressed in braces "{}") that form a numerical sequence as can be seen in the first and third examples.

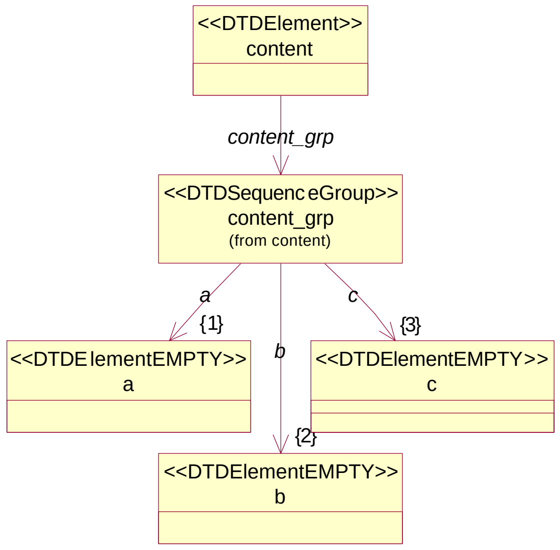
There are some questions on style to be addressed. For example, why are the groups nested classes and why use simple associations rather than an aggregation relationship between constructs? As far as the representation of a group is concerned, a group, although logically defined in the DTD exists only as a component of the content-model of an element definition, therefore managing it within the scope of the element definition (parent class) makes sense. As for the association relationship, this has more to do with the difference in structure between a DTD and an instance document. Consider the following example:

```
<person>            <!ELEMENT person (name, address)>
  <name />          <!ELEMENT name (#PCDATA)>
  <address />       <!ELEMENT address (#PCDATA)>
</person>
```

If you look at the left hand side it would appear as if “name” and address are nested, scoped or contained-by “person”. In fact, this may be the intention of the document designer, but now look at the right hand side i.e.the DTD. All elements live in a global namespace and are referenced by containing element definitions. Thus in the UML, we use a simple association. When we move to XML-Schema, which does allow one element to be defined solely within the scope of another, then we can distinguish between simple association/reference and the containment/scoped relationships.

Example 2 also shows an element of visual style, the more traditional UML layout of example 1 is replaced with a more ‘tree-view’ approach used by many XML design tools.

DTD Example Source	UML Construct
<!ELEMENT content (a, b)>	 <<DTDElement>> content content_grp <<DTDSequenceGroup>> content_grp (from content) a {1} b {2} <<DTDElementEMPTY>> a <<DTDElementEMPTY>> b
<!ELEMENT content (a | b)>	 <<DTDElement>> content content_grp <<DTDChoiceGroup>> content_grp (from content) a b <<DTDElementEMPTY>> a <<DTDElementEMPTY>> b

<!ELEMENT content (a?, b+, c*)>	 <<DTDElement>> content content_grp <<DTDSequenceGroup>> content_grp (from content) a {1} b c {3} <<DTDElementEMPTY>> a <<DTDElementEMPTY>> c {2} <<DTDElementEMPTY>> b

Notations & Entities

The following examples are not terribly illuminating when you see the UML on the right-hand side, however there are a number of properties of the DTD constructs held in tagged values. Tagged values are not generally shown in the visual model by most UML tools, although the UML does define a way to visually represent them, shown in the second example below.

DTD Example Source	UML Construct

`<!NOTATION RatLogo`
`  SYSTEM`
`  "http://…/logo.gif">`
	<<DTDNotation>>
	RatLogo

<!ENTITY Copyright "Copyright (c) …">	<<DTDEntity>> Copyright {ParameterEntity=False, ExternalEntity=False, InternalValue=""}
<!ENTITY % WebDesktop-Include SYSTEM "WebDesktop.dtd">	<<DTDEntity>> % WebDesktop-Include
<!ENTITY % Name-Class "First, MI, Surname">	<<DTDEntity>> % Name-Class

Mapping a DTD Model to XML Schema

The conversion rules are similar to those described by the Perl script[1] published on the W3C XML-Schema site; where differences exist they will be called out in the text below. It is important to demonstrate the power of XML-Schema to the user, without making the resulting output overly complex or difficult to visually map to the original DTD.

Elements & Attributes

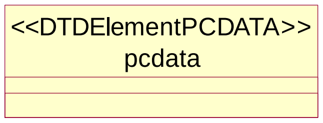
The following demonstrate the more verbose nature of XML-Schema, and also the fact that unlike a DTD, an XML-Schema is expressed in legal XML. There are a number of ways to express an element mapping. We have chosen the most

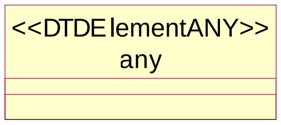
[1] http://www.w3.org/2000/04/schema_hack/

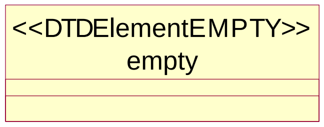
common which nests a complexType definition within the element definition. It is possible to separate these two and have the element definition reference an external complexType definition, however the way shown below is generally considered to better map to the semantics of a DTD.

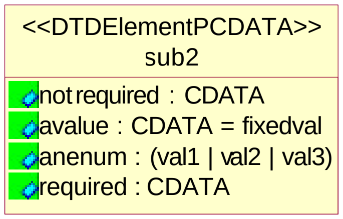
Note below the very different construct for expressing cardinality, the “minOccurs” and “maxOccurs” attributes, the default value for both of which is “1”. In the second example, there are three ways of expressing an “ANY” content model in XML-Schema: an empty complexType definition, a reference to the element “ANY” and the more verbose form shown below. We chose the method below to show the power of XML-Schema and also specifically the “processContents” attribute that can control the validation of an instance.

UML Construct	Resulting XML-Schema

	`<element name = 's3'>`
	` <complexType content = 'textOnly'>`
	` </complexType>`
	`</element>`

	`<element name = 'any'>`
	` <complexType>`
	`   <any minOccurs = '0' maxOccurs = 'unbounded'`
	`        processContents = 'skip' />`
	`  <anyAttribute processContents = 'skip' />`
	` </complexType>`
	`</element>`

	`<element name = 'empty'>`
	` <complexType content = 'empty'>`
	` </complexType>`
	`</element>`

 <<DTDElementPCDATA>>
sub2
notrequired : CDATA
avalue : CDATA = fixedval
anenum : (val1 | val2 | val3)
required : CDATA
	`<element name = 'empty'>`
	` <complexType content = 'textOnly'>`
	`  <attribute name = 'notrequired' type = 'string' />`
	`  <attribute name = 'avalue' type = 'string'`
	`          use = 'fixed' value = 'fixedval' />`
	`  <attribute name = 'anenum'>`
	`   <simpleType base = 'NMTOKEN'>`
	`    <enumeration value = 'val1' />`
	`    <enumeration value = 'val2' />`
	`    <enumeration value = 'val3' />`
	`   </simpleType>`
	`  </attribute>`
	`  <attribute name = 'required'`
	`          type = 'string' use = 'required' />`
	` </complexType>`
	`</element>`

Groups, Order & Cardinality

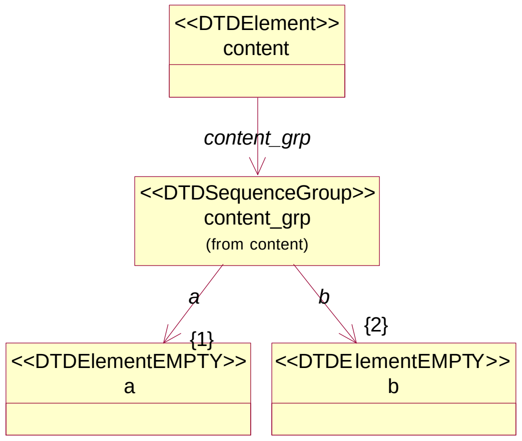
The following examples demonstrate a number of interesting constructs. Specifically, we showthat groups become XML elements rather than simply a parenthesized list, that the nested “complexType” defines the content model for the element and that the content model is expressed as (in this case) element references.

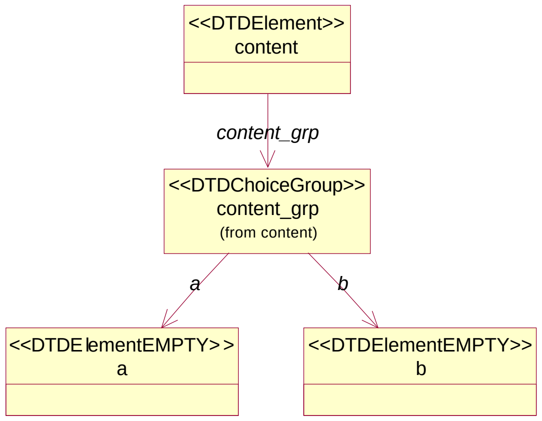
It is possible in XML-Schema to define elements within another element definition, so the following would be legal:

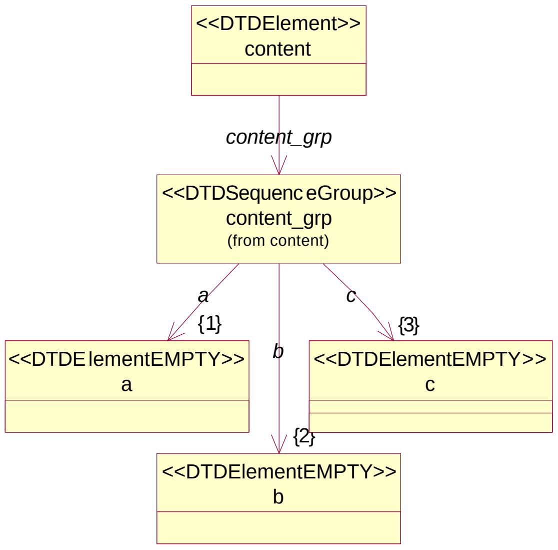
```
<element name = 'content'>
 <complexType content = 'elementOnly'>
  <sequence>
```

```
   <element name = 'a'>
    <complexType content = 'elementOnly'>
     …
    </complexType>
   </element>
  </sequence>
 </complexType>
</element>
```

UML Construct	Resulting XML-Schema

 <<DTDElement>>
content
content_grp
<<DTDSequenceGroup>>
content_grp
(from content)
a
b
{1}
{2}
<<DTDElementEMPTY>>
a
<<DTDElementEMPTY>>
b
	<element name = 'content'>
	 <complexType content = 'elementOnly'>
	  <sequence>
	   <element ref = 'a' />
	   <element ref = 'b' />
	  </sequence>
	 </complexType>
	</element>

 <<DTDElement>>
content
content_grp
<<DTDChoiceGroup>>
content_grp
(from content)
a
b
<<DTDElementEMPTY>>
a
<<DTDElementEMPTY>>
b
	<element name = 'content'>
	 <complexType content = 'elementOnly'>
	  <choice>
	   <element ref = 'a' />
	   <element ref = 'b' />
	  </choice>
	 </complexType>
	</element>

 <<DTDElement>> content
content_grp
<<DTDSequenceGroup>> content_grp (from content)
a
{1}
b
{2}
c
{3}
<<DTDElementEMPTY>> a
<<DTDElementEMPTY>> c
<<DTDElementEMPTY>> b
	`<element name = 'content'>`
	` <complexType content = 'elementOnly'>`
	`  <sequence>`
	`   <element ref = 'a'`
	`    minOccurs = '0' />`
	`   <element ref = 'b'`
	`    maxOccurs = 'unbounded' />`
	`   <element ref = 'c'`
	`    minOccurs = '0'`
	`    maxOccurs = 'unbounded' />`
	`  </sequence>`
	` </complexType>`
	`</element>`

Notations & Entities
In XML-Schema, there exists a notation construct that maps almost one-to-one with that in a DTD. This is shown in example 2. As for entities, no such construct exists in XML-Schema,. Our sample conversion script does however try to make some assumptions about the use of an entity as it is specified in the model.

UML Construct	Resulting XML-Schema

<<DTDNotation>>
RatLogo
	`<notation`
	` name = 'RatLogo'`
	` system = 'http://…/logo.gif' />`

<<DTDEntity>> Copyright	`<!-- !ENTITY Copyright 'Copyright (c) 2000, Rational Software Corporation' -->`

<<DTDEntity>>
% WebDesktop-Include
	`<include`
	` id = 'WebDesktop-Include'`
	` schemaLocation = 'WebDesktop.dtd' />`

<<DTDEntity>>
% Name-Class
	`<complexType`
	` name = 'Name-Class' />`
	`<!—First, MI, Surname -->`

Rational Rose XML Support

W3C XML 1.0 DTD Support was added to the "Rational Rose 2000e, Enterprise Edition" product; this was the industry's first UML-based tool for developing DTDs and demonstrates Rational Software's commitment to make XML-enabled applications a reality for our customers. The 2000e product was a landmark in that it also included the first UML-based tool for analyzing web artifacts and the industry's first UML-based data modeling solution.

For more information see:

XML 1.0	http://www.w3.org/TR/1998/REC-xml-19980210
XML-Schema	http://www.w3.org/XML/Schema.html
Rational Rose	http://www.rational.com/rose/