Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 1

Part I

What Is XML?

This part contains a chapter that focuses on guidelines for achieving good grammar and style when modeling information using XML.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 2

Chapter 1

Information Modeling with XML

Chris Brandin

1.1 Introduction

When XML first came into use, it was seen primarily as a data interchange standard. Since then it has come to be used for more and more things—even serving as the core for development and deployment platforms such as Microsoft's .NET. Increasingly, XML has become the means to model components of information systems, and those components automatically construct themselves around what has been expressed in XML. This represents the real potential of XML—the ability to model the behavior of an entire application in XML once, instead of repeatedly in different ways for each component of an application program.

As long as XML was used as a container for data managed by legacy systems, it was sufficient to consider only syntax when building documents. Now that XML is being used to do more than simply express data, it is important to consider grammar and style as well. Obviously, proper syntax is necessary for parsers to be able to accept XML documents at all. Good grammar insures that once XML information has been assimilated, it can be effectively interpreted without an inordinate need for specific (and redundant) domain knowledge on the part of application programs. Good style insures good application performance, especially when it comes to storing, retrieving, and managing information.

Proper XML syntax is well understood and documented, so that topic will not be discussed here. This chapter does not discuss how to build XML schemas or DTDs, as they are also well documented elsewhere. This chapter is intended as a practical guide to achieving good grammar and style when modeling information in

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 4

XML—which translates to building flexible applications that perform well with minimal programming effort. Grammar is often regarded as being either right or wrong. True, there are "wrong" grammatical practices; but past that, there is good grammar and bad grammar—and everything in between. Arguably, there is no such thing as wrong style, only a continuum between the good and the bad.

1.2 XML as an Information Domain

XML allows us to model information systems in a natural and intuitive way. This is because XML allows us to express information in ways that better match the way we do business. We now have an information-modeling mechanism that allows us to characterize what we want to do, rather than how we have to do it. XML simply does a much better job of reflecting the way the real world operates than the data-modeling mechanisms that preceded it. XML brings a number of powerful capabilities to information modeling:

- **Heterogeneity:** Where each "record" can contain different data fields. The real world is not neatly organized into tables, rows, and columns. There is great advantage in being able to express information, as it exists, without restrictions.
- **Extensibility:** Where new types of data can be added at will and don't need to be determined in advance. This allows us to embrace, rather than avoid, change.
- **Flexibility:** Where data fields can vary in size and configuration from instance to instance. XML imposes no restrictions on data; each data element can be as long or as short as necessary.

XML is also self-describing and informationally complete; applications can use this feature to automatically build themselves with little or no programming required. Companies such as BEA, TIBCO, and Microsoft offer frameworks for building applications, with a minimum of effort, that use XML as the basis for expressing information. In environments like these, XML becomes a universal information-structuring tool where system components no longer need to be programmed separately as discreet silos of functionality. NeoCore offers an XML Management System (XMS) that brings an entirely transparent persistence mechanism to the fold, requiring no separate database design process, indexing instructions, or use-case predefinition. Moreover, NeoCore's XMS carries forward the characteristics of XML that make it powerful as an information domain—heterogeneity, extensibility, and flexibility. All that is required to store, retrieve, and manage information is that it be expressed in XML, and that queries be expressed as XPath or

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 5

XQuery patterns. This can have a profound effect on rapid application development efficiency, especially when changes have to be made. When we build XML-centric systems, we can often accommodate changes by modifying the underlying XML, and information system components will adjust themselves accordingly without the need for reprogramming.

1.3 How XML Expresses Information

XML expresses information using four basic components—tags, attributes, data elements, and hierarchy. Each of these components serves a unique purpose; each represents a different "dimension" of information. In order to illustrate these basic components, we will use a simple XML fragment from an application dealing with readings from colorimeters (devices that measure colors using tri-stimulus readings).

Data elements are represented in **bold** type in Listing 1.1. In XML, a data element equates to "data" as we have traditionally thought of it. If we simply extract the data elements, we get "**0, 255, 255**", which is meaningless unless you know what the data definitions are. XML adds context to data, thereby giving it meaning, by adding tags (represented in regular type in the listing). Tags describe what data elements are. Attributes (represented in ***italics*** in the listing) tell us something about or how to interpret data elements. Colorimeters can represent RGB tri-stimulus values in a variety of resolutions. If the reading had been taken with a resolution of 16 bits, for example, values of "**0, 255, 255**" would represent a very dark cyan, instead of pure cyan. So, we need the "*resolution=8*" attribute to correctly interpret the RGB reading values in Listing 1.1.

Listing 1.1 Simple XML Fragment

```
<colorimeter_reading>
      <RGB resolution=8>
             <red> 0 </red>
             <green> 255 </green>
             <blue> 255 </blue>
      </RGB>
</colorimeter_reading>
```

Now we have data (data elements), we know what they are (tags), and we know how to interpret them (attributes). The final step is to determine how to string it all together, and that is where hierarchy comes in. So far, we have represented three dimensions of information explicitly. The last dimension, how everything relates, is implied spatially. This means that much of what we need to know is contained in how we order the components of XML information. In order to give data meaning,

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 6

a complete context must be provided, not just the most immediate tag or attribute. For example, if we simply say "*red=0*", it will not mean much because we have not provided an adequate context. If we include all tags in the hierarchy leading up to the reading of "0", we achieve a more complete context: "`<colorimeter_reading><RGB><red>` 0". Although we have a complete understanding of what the data element represents and its value, some ambiguity as to how to interpret the value still remains. The attribute "*resolution58*" belongs to the tag "`<RGB>`". Because "`<RGB>`" is a part of our context, any attribute belonging to it (or any attribute belonging to any tag in our context for that matter) applies. Now we know how to interpret the value of the data element as well. Related information is represented in the hierarchy as siblings at various levels; as a result, hierarchy tells us how data elements are related to each other.

1.4 Patterns in XML

In order to effectively model information using XML, we must learn how to identify the natural patterns inherent to it. First, we must determine whether we have used XML elements properly. To do this we will analyze the XML fragment shown in Listing 1.2.

Listing 1.2 Example XML Fragment

```
<colorimeter_reading>
      <device> X-Rite Digital Swatchbook </device>
      <patch> cyan </patch>
      <RGB resolution=8>
            <red> 0 </red>
            <green> 255 </green>
            <blue> 255 </blue>
      </RGB>
</colorimeter_reading>
```

We examine each data element and ask the following question:

- Is this data, or is it actually metadata (information about another data element)?

We examine every attribute and ask the following questions:

- Does the attribute tell us something about or describe how to interpret, use, or present data elements?

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 7

- Is the attribute truly metadata, and not actually a data element?
- Does it apply to all data elements in its scope?

We examine every tag and ask the following question:

- Does this tag help describe what all data elements in its scope are?

We examine the groupings we have created (the sibling relationships) and ask:

- Are all members of the group related in a way the parent nodes describe?
- Is the relationship between siblings unambiguous?

If the answer to any of the preceding questions is "no," then we need to cast the offending components differently.

After insuring that information has been expressed using the components of XML appropriately, we examine how everything has been stitched together. To do this we create an information context list from the XML fragment. This is done by simply taking each data element and writing down every tag and attribute leading up to it. The resulting lines will give us a flattened view of the information items contained in the XML fragment. A context list for the example XML fragment in Listing 1.2 would look like the one shown in Listing 1.3.

Listing 1.3 Context List for Example XML Fragment

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><RGB resolution=8><red> 0
<colorimeter_reading><RGB resolution=8><green> 255
<colorimeter_reading><RGB resolution=8><blue> 255
```

If we convert these lines to what they mean in English, we can see that each information item, and its context, makes sense and is contextually complete:

1. This colorimeter reading is from an X-Rite Digital Swatchbook.
2. This colorimeter reading is for a patch called cyan.
3. This colorimeter reading is RGB-red and has an *8-bit* value of 0.
4. This colorimeter reading is RGB-green and has an *8-bit* value of 255.
5. This colorimeter reading is RGB-blue and has an *8-bit* value of 255.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 8

Next we examine the groupings implied by the tag hierarchy:

- `"<colorimeter_reading>" contains "<device>", "<patch>", and "<RGB>" (plus its children).`
- `"<RGB>" contains "<red>", "<green>", and "<blue>".`

"`<colorimeter_reading>`" represents the root tag, so everything else is obviously related to it. The only other implied grouping falls under "`<RGB>`". These are the actual readings, and the only entries that are, so they are logically related in an unambiguous way.

Finally, we examine the scope for each attribute:

- `"resolution=8" has the items "<red>", "<green>", and "<blue>" in its scope.`

"*resolution=8*" logically applies to every item in its scope and none of the items not in its scope, so it has been appropriately applied.

A self-constructing XML information system (like NeoCore XMS) will use the structure of and the natural patterns contained in XML to automatically determine what to index. Simple queries are serviced by direct lookups. Complex queries are serviced by a combination of direct lookups, convergences against selected parent nodes, and targeted substring searches. With NeoCore XMS no database design or indexing instructions are necessary—the behavior of XMS is driven entirely by the structure of the XML documents posted to it. Index entries are determined by inference and are built based on the natural patterns contained in XML documents. NeoCore XMS creates index entries according to the following rules:

- An index entry is created for each data element.
- An index entry is created for each complete tag context for each data element —that is, the concatenation of every tag leading up to the data element.
- An index entry is created for the concatenation of the two preceding items (tag context plus data element).

For the XML fragment in Listing 1.2, the following items would be added to the pattern indices (actually, this list is not complete because partial tag context index entries are also created, but a discussion of those is beyond the scope of this chapter):

1. **X-Rite Digital Swatchbook**
2. **cyan**
3. **0**

4. **255**
5. **255**
6. <colorimeter_reading><device>
7. <colorimeter_reading><patch>
8. <colorimeter_reading><RGB><red>
9. <colorimeter_reading><RGB><green>
10. <colorimeter_reading><RGB><blue>
11. <colorimeter_reading><RGB ***resolution=8***>
12. <colorimeter_reading><device> **X-Rite Digital Swatchbook**
13. <colorimeter_reading><patch> **cyan**
14. <colorimeter_reading><RGB ***resolution=8***><red> **0**
15. <colorimeter_reading><RGB ***resolution=8***><green> **255**
16. <colorimeter_reading><RGB ***resolution=8***><blue> **255**

Entries 1–5 are data only, entries 6–11 are tag context only, and entries 12–16 are both.

At this point it is important to consider how performance will be affected by the structure of the XML document. Because the inherent patterns inferred from the XML itself can be used to automatically build a database, the degree to which those patterns match likely queries will have a big effect on performance, especially in data-centric applications where single data elements or subdocuments need to be accessed without having to process an entire XML document.

1.5 Common XML Information-Modeling Pitfalls

We could easily arrange the XML fragments from the previous section in other, perfectly acceptable ways. There are many more, albeit syntactically correct, unfortunate ways to arrange the information. Common mistakes made when creating XML documents include:

- Inadequate context describing what a data element is (incomplete use of tags)
- Inadequate instructions on how to interpret data elements (incomplete use of attributes)
- Use of attributes as data elements (improper use of attributes)
- Use of data elements as metadata instead of using tags (indirection through use of name/value pairings)

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 10

- Unnecessary, unrelated, or redundant tags (poor hierarchy construction)
- Attributes that have nothing to do with data element interpretation (poor hierarchy construction or misuse of attributes)

These mistakes sap XML of its power and usefulness. Time devoted to good information modeling will be paid back many times over as other components of applications are developed. We can put a great deal of intelligence into XML documents, which means we do not have to put that intelligence, over and over again, into every system component.

Because XML is very flexible, it is easy to abuse. Sometimes the best way to illustrate how to do something is by counterexample. Much, if not most, of the XML we have seen is not well designed. It is not difficult to design XML with good grammar and good style, and doing so will save a lot of time and effort in the long run—to say nothing of how it will affect performance. The following sections contain a few examples of poorly constructed XML fragments.

1.5.1 Attributes Used as Data Elements

This may be the most common misuse of XML. Attributes should be used to describe how to interpret data elements, or describe something about them—in other words, attributes are a form of metadata. They are often used to contain data elements, and that runs counter to the purpose of attributes.

Listing 1.4 contains no data elements from readings at all; the attributes apply to nothing. Attributes that apply to nothing, obviously, describe how to interpret nothing.

Listing 1.4 XML with No Data Elements

```
<colorimeter_reading>
      <device> X-Rite Digital Swatchbook </device>
      <patch> cyan </patch>
      <RGB resolution=8 red=0 green=255 blue=255 />
</colorimeter_reading>
```

If we examine each attribute, especially the data portion (the part to the right of the equal sign), we can determine whether they actually represent data, or metadata:

- ***resolution=8***: This is a true attribute because the value “8” does not mean anything by itself; rather it is an instruction for interpreting data elements, and therefore it is metadata.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 11

- ***red=0***: This is clearly actually data because it is a reading from the colorimeter; moreover, in order to be correctly interpreted, it requires the ***"resolution=8"*** attribute. This attribute does not tell us how to interpret data—it is data. Consequently it should be recast as a tag/data element pair.
- ***green=255, blue=255***: The previous analysis of ***"red=0"*** applies.

1.5.2 Data Elements Used as Metadata

This is often a result of emulating extensibility in a relational database. Instead of creating columns accounting for different fields, a database designer will create two columns: one for field type and one for field contents. This basically amounts to representing metadata in data element fields and is shown in Listing 1.5.

Listing 1.5 XML Data Elements Used as Metadata

```
<colorimeter_reading>
      <device> X-Rite Digital Swatchbook </device>
      <patch> cyan </patch>
      <RGB>
            <item>
                  <band> red </band>
                  <value> 0 </value>
            </item>
            <item>
                  <band> green </band>
                  <value> 255 </value>
            </item>
            <item>
                  <band> blue </band>
                  <value> 255 </value>
            </item>
      </RGB>
</colorimeter_reading>
```

If we decompose this document into an information context, we get Listing 1.6.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 12

Listing 1.6 Information Context for Listing 1.5

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><RGB ><item><band> red
<colorimeter_reading><RGB ><item><value> 0
<colorimeter_reading><RGB ><item><band> green
<colorimeter_reading><RGB ><item><value> 255
<colorimeter_reading><RGB ><item><band> blue
<colorimeter_reading><RGB ><item><value> 255
```

Listing 1.6 translates to approximately the following in English:

1. This colorimeter reading is from an X-Rite Digital Swatchbook.
2. This colorimeter reading is for a patch called cyan.
3. This colorimeter reading item is RGB band red.
4. This colorimeter reading item is RGB and has a value of 0.
5. This colorimeter reading item is RGB band green.
6. This colorimeter reading item is RGB and has a value of 255.
7. This colorimeter reading item is RGB band red.
8. This colorimeter reading item is RGB and has a value of 255.

The last six lines are contextually weak. Lines 3, 5, and 7 don't contain any readings; they contain metadata about the lines following them. Lines 4, 6, and 8 don't adequately describe the readings they contain; they are informationally incomplete and ambiguous. In fact, lines 6 and 8 are exactly the same, even though the readings they represent have different meanings.

1.5.3 Inadequate Use of Tags

This is often a result of emulating extensibility in a relational database. Instead of building separate tables for different data structures, a database designer will create one table for many different data structures by using name/value pairs. This represents unnecessary indirection of metadata and an inappropriate grouping of data elements, to the detriment of performance (because what should be direct queries become joins) and reliability (because grouping is ambiguous). This is shown in Listing 1.7.

Listing 1.7 Use of Name/Value Pairs

```
<colorimeter_reading>
      <device> X-Rite Digital Swatchbook </device>
      <patch> cyan </patch>
      <mode> RGB </mode>
      <band> red </band>
      <value> 0 </value>
      <band> green </band>
      <value> 255 </value>
      <band> blue </band>
      <value> 255 </value>
</colorimeter_reading>
```

If we decompose this document into an information context, we get Listing 1.8.

Listing 1.8 Information Context for Listing 1.7

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><mode> RGB
<colorimeter_reading><band> red
<colorimeter_reading><value> 0
<colorimeter_reading><band> green
<colorimeter_reading><value> 255
<colorimeter_reading><band> blue
<colorimeter_reading><value> 255
```

Translated to English, Listing 1.8 becomes:

1. This colorimeter reading is from an X-Rite Digital Swatchbook.
2. This colorimeter reading is for a patch called cyan.
3. This colorimeter reading is in RGB mode.
4. This colorimeter reading is red.
5. This colorimeter reading has a value of 0.
6. This colorimeter reading is green.
7. This colorimeter reading has a value of 255.
8. This colorimeter reading is blue.
9. This colorimeter reading is has a value of 255.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 14

The last six lines are contextually weak, and line 3 represents nothing but context. Lines 3, 4, 6, and 8 do not contain any readings; they contain metadata about the lines following them. Lines 5, 7, and 9 don't describe the readings they contain at all; they are informationally incomplete and ambiguous. In fact, lines 7 and 9 are exactly the same and contained within the same group, even though the readings they represent have different meanings and should belong to different groups. We could add tags to encapsulate reading elements into groups so that the bands and reading values are unambiguously related to each other. But first, we should determine whether each data element truly represents data. If we examine the data elements, we can determine whether they really represent data or metadata, and whether they have an adequate context:

- **X-Rite Digital Swatchbook**: This is clearly data.
- **cyan**: This is also clearly data.
- **RGB**: Although this could be considered data in the academic sense, it is not of much value by itself. Furthermore, it is needed to understand the meaning of data elements following it.
- **red**, **green**, and **blue**: These are also data in the academic sense only. They lack adequate context as well. For example, a colorimeter reading in the red band could mean a number of different things.
- **0**, **255**, and **255**: These are the actual colorimeter readings; they are clearly data. They are, however, nearly devoid of critical context—namely the color mode and the color band they represent.

1.6 A Very Simple Way to Design XML

One great advantage of XML information modeling over traditional data modeling is that it serves as a much more intuitive analog of reality. Because of this, a very simple method for designing XML documents produces surprisingly good results. In fact, it will produce better results than many, if not most, "industry standard" XML schemas. Forget that you will be using a computer to manage information—in fact, forget almost everything you know about computers. Instead, imagine that you will be managing your information manually, and design simple forms accordingly. First, make the preprinted parts of the forms into tags; second, make the parts you fill in into data elements; and third, change things like units into attributes. Obviously, doing so will not produce totally optimum results, but it will serve quite well—and it's a good way to start.

Let’s look at a simple example—a telephone number directory. We will start with a manual entry form.

Telephone Directory Listing

Name:	John A. Doe
Address:	123 Main Street
City:	Pleasantville
State:	Maryland
Zip Code:	12345
Telephone:	(999) 555-1234

If we convert this directly into an XML document (spaces become underscores), we get Listing 1.9.

Listing 1.9 Telephone Directory Listing as XML

```
<Telephone_Directory_Listing>
      <Name> John A. Doe </Name>
      <Address> 123 Main Street </Address>
      <City> Pleasantville </City>
      <State> MD </State>
      <Zip_Code> 12345 </Zip_Code>
      <Telephone> (999) 555-1234 </Telephone>
</Telephone_Directory_Listing>
```

Now we will make some small changes. First, we will separate the name into first, middle initial, and last name, and group them together. We will also group the address and separate the telephone number and area code into its own group. Separating fields, such as the name, makes it possible to use the components as individual query terms that will be serviced with direct lookups instead of requiring partial content scans within fields. This significantly improves performance in cases where a query, for example, might be for “John Doe” instead of “John A. Doe”. The resulting XML is shown in Listing 1.10.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 16

Listing 1.10 Telephone Directory Listing in XML after Changes

```
<Telephone_Directory_Listing>
      <Name>
            <First> John </First>
            <MI> A. </MI>
            <Last> Doe </Last>
      </Name>
      <Address>
            <Street> 123 Main Street </Street>
            <City> Pleasantville </City>
            <State> MD </State>
            <Zip_Code> 12345 </Zip_Code>
      </Address>
      <Telephone>
            <Area_Code> 999 </Area_Code>
            <Number> 555-1234 <Number>
      </Telephone>
</Telephone_Directory_Listing>
```

The XML document in Listing 1.10 would serve as a good basis for a telephone directory. When thinking about additional information that may have to be added to some listings (additional address lines, additional telephone numbers, etc.), it is important to remember that XML is extensible; a field has to be added only when it is necessary—not globally to all listings.

Many businesses are basically forms driven. For example, clinical trials in the pharmaceutical industry start with forms that have to be approved before the computer systems managing the information can be designed. Because forms can be converted into XML so easily, it is now possible to build systems that are driven primarily by business objectives in intuitive ways, rather than by abstract computing paradigms.

1.7 Conclusion

One of the most promising things about XML, and the new breed of tools built on it, is that we can build applications that are driven by a single information model rather than multiple data models accommodating each application function. We can change the behavior and functionality of application programs by changing the

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 17

underlying XML rather than by changing code. Additionally, we can optimize performance by changing the way information is expressed. Even in environments not fully leveraging XML as a central information model, it is important to design good XML for the sake of readability and maintainability. Building good applications efficiently requires that we learn not only to use XML correctly, but that we learn also to use it well.

Chaudhri_CH01.qxd 2/11/03 11:34 AM Page 18