3 Sins of XML Usage

Roger Costello
October 2012

Sin #1: Using Java to Process XML

A while back I was talking to a friend and he said, *I hate XML*. I asked why and he said, *It is so painful to process XML using Java*. Yes it is. Java isn't the right language for processing XML. But there are languages that are specifically designed for processing XML, such as XSLT, XPath, and XQuery. My friend was using the wrong tool for the job; it's no wonder he was having such a painful experience with XML.

Java is a fine language, just not for processing XML. Use Java to initiate XSLT, XPath, and/or XQuery processing on XML. After the XML processing is finished then you can use Java to, say, display the results.

Bottom line: Use a domain-specific language (DSL) such as XSLT, XPath, XQuery, and don't process XML using a general-purpose programming language such as Java, C#, or Python.

> *Get data into XML as quickly as possible. Keep it in XML until the last possible minute. Bring all your XML tools to bear on solving the data processing problem*. [Sean McGrath]

Sin #2: Designing XML in an Object-Oriented (OO) Fashion

Years ago Tim Bray said something to the effect, *XML is 180 degrees apart from OO*. I completely agree. And yet it seems that everyone is designing XML in an OO fashion. For example, consider how people design XML Schemas: they create types that inherit from base types. That is so fragile and so inflexible. The right way to design XML is through composition. Consider this: *XPath 3.0 is a composable language*. That should clue you into what the XML experts are thinking – composition is the way to go.

In light of the fact that XSLT, XPath, and XQuery are becoming increasingly functional, and functional languages are about composing, it is time to stop using OO and start using composition. I'll have more to say about design-by-composition in a future article. In the meantime think about this, *How can I design my components so that they can be composed with other component*?

Sin #3: Neglecting the Format of the Data

You spend a lot of time carefully crafting a grammar for your markup. And then you declare each leaf element to be a string. Huh? So the structure of the markup is important but the structure of the data is unimportant? It seems to me that we've lost our way; the data is what it's all about, the markup is just window dressing.

Look at the RFCs produced by IETF, they carefully define the format of the data using BNF rules. They don't just say, *Oh, put a string here*. No, using a BNF they define every part of the data in great detail.

You too should be describing the structure of all data in great detail. In your XML Schema every leaf element should contain a pattern facet that has a regular expression which precisely defines the structure of the data.

Feedback

I received excellent feedback:

Sin #1: Using Java to Process XML

Liam Quin: The real message is, *Use a domain-specific language such as XPath, XSLT, XQuery, and don't process XML directly*.

Kurt Cagle: Most of my tools for processing XML are ultimately Java based. However, I would agree that building a Java pipeline for processing XML and nothing else is highly counterproductive, compared to XQuery, XSLT or XProc.

David Lee: I agree with Kurt. One should reuse an already written and tested Java pipeline for processing XML. Hmm ... one that includes all his favorite xwords ... XQuery, XSLT , XProc ... and more.

Sin #2: Designing XML in an Object-Oriented (OO) Fashion

Liam Quin: If we take the fundamental characteristics of OOP to be data hiding, implicit dispatch and message passing (e.g. from SmallTalk 80), then no, because these are orthogonal issues to XML design. If you mean, XML as objects then yes - XML elements are not generally objects in the OO sense. They don't have methods or classes.

However, class-based inheritance (the usual OO mechanism for implicit dispatch, so you can say the Shape->draw() and the drawing function appropriate to that particular shape gets used) overlaps with ontological inheritance (is-a), and ontological inheritance, the is-a hierarchy, is often very appropriate for XML.

Sin #3: Neglecting the Format of the Data

Liam Quin: How far down you go is always subjective and is subject in particular to cost/benefit speculation. Your PDF document mentions IETF specs, and in a network protocol you may well end up labeling mantissa and exponent of a floating point number separately. On the other hand

<timestamp date="Thu, 25 Oct 2012 14:11:20 -0400">

is perfectly fine in other IETF specs.

Liam Quin: There are times (albeit not very many) when using DOM in Java is actually just fine, and there are times when a simple schema is appropriate, and there are times when OO design is very helpful.