

The reusable generation

by Chris Sells



Over the years, I've had a number of consulting clients, some of whom have asked me to produce code to get them started in a kind of a wizard-by-the-hour fashion. The following three are a representative sample of the kinds of code I was asked to produce. Each case was very different, but all of them have a common theme: one client asked me to port their MFC COM Control to ATL; one client asked me to build a vertical slice of their n-tier system; and one client asked me to implement a persistent object model to represent a

'message' in their system that would be passed from point to point as work was done.

You may have a hard time finding the similarity between these three coding projects, so I'll tell you: each was already defined by a set of metadata that, when combined with the client's implementation rules, were sufficient to predetermine most or even all of the code. What my clients were asking me to do in each case was to produce the correct code from the metadata. The object model and the MFC COM Control were both described in COM IDL (Interface Definition Language). The n-tier system was to be a direct mapping of their database, which was described in SQL Schema. My clients wanted my help manually translating that metadata into representative code, which I did.

In the case of the MFC COM Control, I ignored the metadata completely and went right for the C++ source code, manually translating MFC constructs to ATL constructs. In the case of the n-tier system, I built a vertical slice based on the design we had worked out, only referring to the schema to determine the columns in the tables I was supposed to be typing into my code. In neither case was my work reusable for that client or any other client. If I had wanted to provide an MFC to ATL translation service to another client, I would be back to manually translating the code. And the client that got my vertical

slice had to manually replicate it across tens of tables using the standard code reuse methodology formally known as 'copy and paste'.

In neither case did I feel like I had done all that I could do for the client, because I had not left them with a good technique to carry on their remaining work in my absence; nor had I built up a tool or repeatable service that I could offer to other clients. So, for my next client, I tried something different.

The client that wanted me to implement an object model knew exactly what the object was supposed to look like, because they'd described it in COM IDL. In this case, I built a Visual Basic program that would programmatically navigate the compiled version of the IDL, a COM TypeLibrary, and generate the appropriate code based on their specific implementation requirements, eg persistence, exposing properties, exposing COM collections etc.

The beauty of this technique was that I only had to write the code for each specific feature once and it would be automatically replicated in every case. When I found bugs, I fixed them in the VB program and regenerated the code. Using this technique, I was literally able to generate a 98% implementation of their entire object model in 24 hours, in comparison to a member of their own staff who'd be working on it for eight months. Granted, their staff hadn't written books on COM like I had, which was why they came to me in the first place, but a 1000% gain in coding efficiency is still pretty impressive. The client certainly thought so.

The real hero, of course, was code generation. Reuse techniques like classes, components and templates are great, but they don't solve the whole problem. There is still a great deal of code to be written to take advantage of these reuse techniques. Think about the last project you were on. Was there some SQL Schema or XML Schema or COM IDL file that you were manually translating into an implementation? Was there enough grunt work involved that you down-shifted into 'copy and paste' mode? If so, you're already using code generation, but in its most primitive form.

I advise stepping up to a more general-purpose technique and some real code generation tools. My favourite code generation technique involves reusing Microsoft Active Service Pages (ASP) syntax, with full debugging support, pluggable scripting languages and access to any COM object in the known universe and, instead of generating HTML, generating C++ or VB or Java or XML or whatever it is you're after. Another popular choice is eXtensible Stylesheet Language Transformations (XSLT). Whichever you decide, you owe it to yourself to give code generation a try. Your clients will thank you for it. Mine did. 

Chris Sells is a long-time consultant, author and speaker on a variety of Microsoft technologies. He can be reached at <http://www.sellsbrothers.com>