From XML to PowerPoint via COM

Paul Prescod

ISOGEN Consulting Engineer
paul@prescod.net

Python is the ultimate glue language. Unix users are very familiar with using Python to talk to programs through sockets and text files. On Windows it is possible to talk to proprietary Microsoft software through the COM communication layer. COM is completely object oriented and fits the "Python model" very nicely. This demonstration shows how I keep my slide materials in a completely open file format (XML) and pump it into that most proprietary and recalcitrant Microsoft program, PowerPoint. Using technologies like PythonCOM, Python programmers can build a bridge from the open software, open standard world to the closed packages that most businesses still rely upon.

The XML for a slideshow looks like this:

```
<slide>
<title>Compared to Perl</title>
<points>
<point>Easier to learn</point>
<point>Easier to read</point>
<point>Easier to extend</point>
<point>More portable.</point>
</points>
</slide>
```

This is parsed into a grove. A grove is a parsed representation for a data file (in this case XML). This software predates the popular DOM API otherwise that would be the obvious choice due to the excellent support provided by packages like PyDOM and 4DOM.

A tree-walker pattern traverses the grove and triggers different methods based on the element's type. Handlers are registered like this:

```
visitor.add( "emph", Emph)
```

This registers a method named "Emph" as a handler for "emph" elements. The handler uses methods and properties from the PowerPoint COM API:

```
def Emph( node, context):
    r=context.textFrame.TextRange
    r.InsertAfter( node.data() )
    r.Font.Bold=1
    r.Font.NameOther="Emph"
```

These calls are partially documented but their real runtime semantics must be discovered through trial and error. Between the COM and PythonCOM dynamic dispatch, the system is also not very high performance. PowerPoint is not designed as a component in an efficient batch publishing system.

If elements could only dispatch based on element type name then it might make sense to automatically look up a method named "emph" for element types named "emph". I chose not to implement this short-cut because an explicit handler registration system allows more sophisticated handlers to be registered.

In this case I am looking for a "title" element type in a "slide" element type. I dispatch to a method named "Title".

```
visitor.add(
      ElementPattern( gi="title",
            ancestors=["slide"] ),
      Title )
```

Images and other objects can be embedded without any special processing. The XML merely refers to the image.

```
<img-slide filename="diagram.wmf">
<title>A Diagram</title>
</img-slide>
```

One general issue in this form of transformation is that slide shows are inherently visual. Editing them in a text file is somewhat analogous to editing a diagram in a text file. While it is possible, it is probably not appropriate for the average executive or sales manager. For those that need to handle large volumes of slide materials, though, the benefits in terms of manageability outweigh the costs.

This sort of project could be accomplished with PowerPoint Visual Basic macros but Python's flexibility is what makes maintenance reasonable despite the baroque details of the PowerPoint API.