XPath Filename Expansion in a Unix Shell

Kaspar Giger
ETH Zürich

Erik Wilde
ETH Zürich

ABSTRACT

Locating files based on file system structure, file properties, and maybe even file contents is a core task of the user interface of operating systems. By adapting XPath's power to the environment of a Unix shell, it is possible to greatly increase the expressive power of the command line language. We present a concept for integrating an XPath view of the file system into a shell, the *XPath Shell (XPsh)*, which can be used to find files based on file attributes and contents in a very flexible way. The syntax of the command line language is backwards compatible with traditional shells, and the new XPath-based expressions can be easily mastered with a little bit of XPath knowledge.

Categories and Subject Descriptors: D.4.9 [**Operating Systems**]: Systems Programs and Utilities — *Command and Control Languages*; E.5 [**Files**]: Organization/Structure

General Terms: Design, Languages

1. INTRODUCTION

By mapping file system information onto an XML structure and giving users an interface to select nodes in this tree, it is possible to provide an interface for file selection which is much more powerful than traditional Unix shells. The *XPath Shell (XPsh)* presented here is based on the *File System XML (FSX)* approach [6], which not only represents file system information, but optionally also includes XML views of file contents.

The main function of a shell is to parse the command line, expand parts of it, and then execute the command. By augmenting the filename expansion process with a syntax inspired by XPath [2], followed by an evaluation of the filename expression based on an XPath selection in the FSX tree, complex filename selections can be easily specified.

While the FSX is useful as a conceptual foundation, it is not ideally suited as the document to which a shell filename expression is applied directly. For example, each file is represented as a `file` element with its filename as the `name` attribute. However, to select a file named `test` in a shell, you want to simply type `test`. In an FSX-based XPath, one would have to write `fsx:file[@name="test"]`. Because of this and similar effects, and because an important goal of the syntax is to be backwards compatible with traditional shells, the shell's syntax is XPath-inspired and is mapped to a proper XPath for filename expansion.

Another important problem arises with the inclusion of file contents as suggested by FSX. While this is conceptually useful and technically possible, it becomes prohibitively expensive for any file system with a considerable number of files. Thus, while we assume that only file contents required for XPath evaluation will be actually included, while unused content remains unexpanded. Even then the evaluation can be rather expensive.

Section 2 describes the syntax and semantics of the expressions allowed in our shell. The semantics are described through a mapping onto the FSX tree, which is a representation of the file system structure and file contents. Implementation issues are discussed in Section 3, which especially describes the challenge of "integrating" file contents when evaluating an XPath.

WWW 2006, May 22–26, 2006, Edinburgh, Scotland.
ACM 1-59593-332-9/06/0005.

2. SYNTAX AND SEMANTICS

To keep the syntax of the XPath shell commands short and straightforward, we do not allow pure XPath expressions. Our *syntax* is derived from the XPath syntax, but includes some restrictions and extensions which allow a better integration of expressions into a Unix shell. The goal was to design a syntax of which the simple parts can be used by anybody having worked with Unix shells, whereas the more complex parts can be easily understood by people knowing XPath, and can be easily explained to all others.

Our syntax, described by a *yacc* grammar, is defined based on the original XPath grammar. The *semantics* are defined by mapping the syntax to XPath expressions that are applied to an FSX tree representing the file system structure and file contents.

We first have to introduce an additional XPath function Named `match` to allow Unix filename pattern matching within XPath. With that improvement one can still write filenames such as `*.txt`. For usability and compatibility reasons, we map the filenames to any FSX-node and do not allow the syntax `AxisSpecifier::NodeTest`:

```
name → fsx:*[match(@name,dirname)]
```

Additionally, we introduce two new and powerful concepts to the shell world: The *descendant-or-self* and the *ancestor-or-self* axes. They are accessible through the double-slash (`//`) and triple-dot (`...`) abbreviations:

```
//  → /descendant-or-self:node()/
... → ancestor-or-self:node()
```

Another powerful extension new to Unix shells is the integration of file contents into the file system tree. As proposed by FSX, all files matching a given pattern (based on the file's MIME type) are expanded to a content and a metadata tree

and then appended to the file node. However, we do not allow to `cd` into the files within our shell (like *xsh* [4] does), because many commands would then have to be prohibited. For supporting content integration we introduce three new operators: The *content* operator (`??`) which leads to the root element of the file content, the *metadata* operator (`@@`) which leads to the root element of the file metadata and the *containing-file* operator (`!!`) which leads to the file that contains the context node. The mapping is defined as follows:

```
?? → self::fsx:file/fsx:content
@@ → self::fsx:file/fsx:metadata
!! → ancestor::fsx:content/parent::fsx:*
```

All other XPath concepts are adopted. For example, predicates allow to select files depending on their attributes (like `*.txt[@size < 100]`) or other more complex constraints.

Namespaces can be used in three ways: By using Clark Notation [3], by exporting the namespace name URI to a shell variable (incidentally named `xmlns_prefix`) or by using the built-in namespaces that are defined by the adaptors.

3. IMPLEMENTATION

The ideal (and naïve) implementation of the XPath shell would be to build a filename expression mapper, for mapping all shell filename expressions to XPaths. Then the file system (including the contents of the files) could be transformed into a DOM tree to which the XPath is then applied. This ideal implementation has to fail, because on the one hand it would be very slow, on the other hand its memory consumption would be very large.

One easy solution to this could be to limit the depth level as Paraschenko proposes for his *xfind* utility, an XPath extension of the UNIX `find` command [5]. However, this severely limits the applicability of the XPath shell for selecting files in a complete file system, and thus this solution is not satisfying.

Another possible approach would be to write a dedicated XPath parser interpreting the XPath stepwise. In each step, one location step is interpreted and all matching nodes are then used as the context nodes for the next location step. The memory consumption will be limited. It will also be quite fast, because it evaluates only required nodes and does not necessarily create the whole DOM tree of the file system.

But for our first approach, we use the existing XPath processor of the Sablotron Library [1] and underlay an XML pull parser that generates the nodes dynamically when the XPath processor needs them. Using this approach, nodes (and in particular content nodes) would only be accessed as required by the XPath implementation, and thus only nodes required for the XPath evaluation would have to be expanded by accessing the file system.

The unsatisfying effect of using an existing XPath processor is that it would be very difficult to make it work more efficiently for our special purpose. And unlike the Unix `find` program we have to initialize the library, then parse the input completely and verify it with the given grammar.

3.1 Content Integration

As explained in Section 2, we want to integrate the contents into our file system tree. We use dynamic adaptors as proposed in FSX. These adaptors are implemented in independent shared libraries with a well-defined interface. This allows other programmers to build their own adaptors.

The adaptors have to implement at least the following six functions:

```
char **implemented_mime_types()
int  n_implemented_mime_types()
char *get_ns_prefix()
char *get_ns_uri()
void expand_metadata( context_node )
void expand_content( context_node )
```

The user defines which adaptor library should be used for which file types (MIME types) in a separate configuration file. When the content of a file is required for evaluating a content operator expression, the shell dynamically loads the library and expands the file to a DOM tree.

This expansion is obvious for XML-like files but not for files with binary content. For JPEG files as an example, we map the EXIF header to an XML document representing the EXIF information. Generally, adaptors do not have to map the full file contents to XML, they may very well only map a subset, such as a file's metadata.

4. PERFORMANCE RESULTS

First experiments show that that our simple implementation is much slower than the Unix `find` utility. Especially the initializing of the program needs a lot of computing. As an example we created a filesystem environment with directories that contain each ten text-files and ten such folders. At the end, there were 2320 files and directories in this environment. We wanted to select files with a given user id recursively. `find` needed 30ms, our (naive and unoptimized) library 13s. This is a big difference. It means that our extension should be used mainly combined with file-contents, where `find` fails.

5. CONCLUSIONS

An XPath shell could be a very useful tool for developers because it allows them to select many files in different sub-directories with one path expression. The files can be selected depending on their attributes and their content with an easily understandable syntax.

A sensitive point of the concept is that finding the files by traveling through many directories could need a lot of computing. But like the Unix `find` command or other powerful utilities, it is the user's choice to specify reasonable boundaries for the search.

6. REFERENCES

[1] Ginger Alliance. Sablotron — XSLT, DOM and XPath processor. http://www.gingerall.org/charlie/ga/xml/p_sab.xml.

[2] James Clark and Steven J. DeRose. XML Path Language (XPath) Version 1.0. World Wide Web Consortium, Recommendation REC-xpath-19991116, November 1999.

[3] James Clark. XML Namespaces. http://www.jclark.com/xml/xmlns.htm.

[4] Petr Pajas. XSH — XML Editing Shell. http://xsh.sf.net/.

[5] Oleg A. Paraschenko. find with XPath over File System. http://uucode.com/texts/xfind.

[6] Erik Wilde. Merging Trees: File System and Content Integration. In *Poster Proceedings of the 15th International World Wide Web Conference*, Edinburgh, UK, May 2006. ACM Press.