Merging Trees: File System and Content Integration

Erik Wilde
ETH Zürich

ABSTRACT

XML is the predominant format for representing structured information inside documents, but it stops at the level of files. This makes it hard to use XML-oriented tools to process information which is scattered over multiple documents within a file system. *File System XML (FSX)* and its content integration provides a unified view of file system structure and content. FSX's *adaptors* map file contents to XML, which means that any file format can be integrated with an XML view in the integrated view of the file system.

Categories and Subject Descriptors: E.5 [**Files**]: Organization/Structure

General Terms: Design, Languages

1. INTRODUCTION

File systems are hierarchically structured, and mapping this hierarchy to an XML structure therefore can easily be done. We present a schema and a framework for representing file system information in an XML document. It uses a modular structure for representing generic file system concepts as well as concepts specific for particular types of file systems (Section 2). Additionally, we describe a framework for integrating file contents with the file system information, so that the resulting document provides an integrated view of the file system structure as well as file contents (Section 3). File content integration can be done directly (for XML and text files) or through adaptors, which provide translators for mapping arbitrary file formats onto XML structures.

The framework presented here is mainly intended for small scale environments where developers work with a variety of files and would like to have a unified view of all these resources. It is not designed for large scale scenarios such as data mining over a large set of resources or large resources.

The approach presented here shares some common ideas with IBM's *Virtual XML Garden*, which is a Java framework for implementing XML processing over diverse structured data sources. The framework presented here is open and extensible (Section 4 describes a naïve implementation), while IBM's approach is tightly bound to the implementation.

2. FILE SYSTEM XML

File system structures are based on some concepts which are shared among a large number of file system types, and other concepts which are specific to a certain type of file system. *File System XML (FSX)* supports these relationships of file system types by providing three simple core concepts of file system contents, and the ability to extend this basic structure with additional node types and/or attributes.

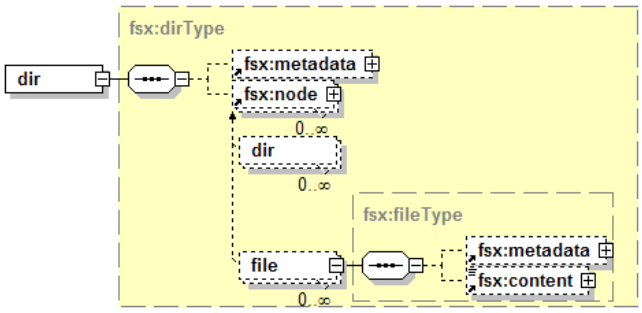

Figure 1: FSX Core Schema

FSX defines the generic `node`, which is an abstract concept and represents any node within a file system structure. A node may be named or unnamed and contains metadata and/or other nodes or contents. The two most common concepts derived from this abstract concept are `dir` and `file` nodes for representing *directories* and *files*. Directories contain metadata and other nodes, while files contain metadata and/or content. This structure is shown in Figure 1 (dashed arrows depict XML Schema substitution groups).

The core schema is used by extensions which add node types and/or attributes, for example the `fsx-unix` schema, which adds the `symlink` node type for *symbolic links* (which are treated neither as directories nor as files), and extends the `dir` and `file` node types with Unix-specific attributes (such as owner and permission information). Other file system types can be represented by creating other schemas, resulting in a file system hierarchy as shown in Figure 2. The figure only shows the additional node types that FSX schemas add, it does not show the addition of attributes.

```
<dir name="test" xmlns=".....">
  <dir name="subdir">
    <file name="README"/>
  </dir>
  <file name="test.txt" mime="text/plain"/>
  <file name="text.xml" mime="application/xml"/>
</dir>
```

This short FSX document omits most of the attributes available for node description, but demonstrates the overall structure of an FSX document. The MIME types may have been part of the file system's internal information, or they

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

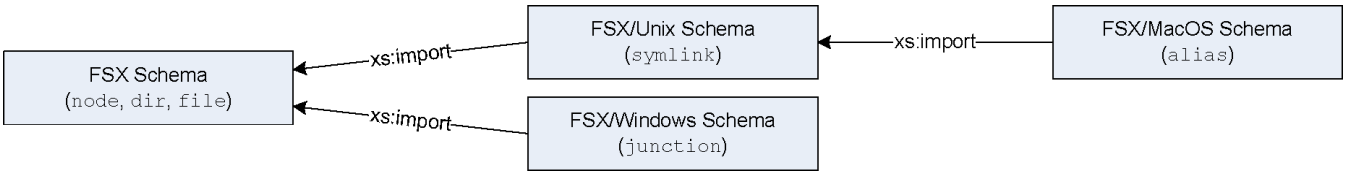

Figure 2: Import Structure of FSX Schemas

may have been derived from the file names or something similar to the Unix `file` command.

```
1. //dir[*[@uid != ../@uid]]
2. //dir[sum(file[starts-with(@mime,"image/")]
                                /@size) > 1048576]
```

Using XPaths on an FSX document, example 1 selects all directories containing nodes owned by somebody else than the directory owner. Example 2 selects all directories which contain image files with a total size of more than 1MB.

3. CONTENT INTEGRATION

The basic FSX tree contains only file system structure and other data available from the file system, it does not contain any file contents. In a second step, the FSX tree is traversed and file metadata and/or contents are appended to `file` nodes. For XML files, for example, this can be done by directly including the XML found in the file in the FSX tree. For text files, it is possible to simply include the file's content as one text node in the FSX tree. Generally, FSX adopts an approach which enables the easy integration of any content type into the FSX tree.

FSX defines *adaptors*, which are selectively applied to file nodes in the FSX tree. If an adaptor matches a file during the traversal process, the associated translator is executed, translating the file's contents into XML which is then integrated into the FSX tree as metadata and/or content.

Matching a file can be done by applying different criteria, based on the file system information available on the `file` node, such as the file name, the MIME type, or the size. If a `file` node matches the criteria specified by the adaptor, the adaptor is executed and generates XML with is used as contributions to the file's metadata or content children.

It is important to point out that nothing in this approach requires an adaptor to provide a complete mapping of a file. For example, media files such as images or audio probably should not be completely translated into XML, but the metadata in these files could be very valuable information to be available through an XML view. Whether such information is considered metadata or content and where it thus is added is decided by the adaptor's designer.

```
1. //id3v2:TIT2[contains(text(),"XML")]
                          /ancestor::dir[1]/@name
2. //file[@mime="application/zip"]
              [not(.//zip:file[@name="README"])]
```

Example 1 returns the names of all directories which contain MP3 files containing some track with "XML" in its title (using a hypothetical `id3v2` namespace for ID3 metadata). Example 2 selects all files which are ZIP archives but do not contain a `README` file somewhere inside the archive (using a hypothetical `zip` namespace for ZIP metadata).

4. IMPLEMENTATIONS

There are currently two implementations of FSX. The prototype implementation uses Java's `java.io.File` class to produce the initial FSX document, and an XSLT 2.0 stylesheet for subsequent content integration. Only XSLT 2.0 adaptors are supported, which means that the adaptors can only work on XML and text-based files. For large file system structures and/or large files, this implementation results in poor performance and the danger of memory overflows. Because the prototype implementation uses XSLT, content integration of XML documents is Infoset-based and not purely XML-based. In practice, this means that entities are resolved, which may not be appropriate in all cases.

The second implementation is *XPath Shell (XPsh)* [1], which is a command line utility based on FSX. XPsh not only supports FSX, but also provides a special syntax for querying FSX, inspired by XPath, but also designed to be as intuitive and compatible with Unix shell file selection mechanisms as possible. XPsh is only available under Unix and has a flexible plug-in concept for adding new adaptors.

5. CONCLUSIONS AND FUTURE PLANS

The prototype implementation is based on a processing model separating the construction of the FSX tree, the content integration, and the XPath evaluation. For large file structures or documents, this implies significant performance problems. The XPsh implementation partly avoids these problems, because it executes the adaptors on demand, only if the XPath evaluation accesses the children of a `file`.

FSX is useful for providing integrated views over files systems and file contents. If this data is used as input for subsequent processing steps, both the prototype and the XPsh implementation can be used. If FSX is used more in the spirit of the Unix `find` and `egrep` commands, then XPsh should be used because of its better optimization of FSX construction and evaluation of XPaths.

Using FSX as an integrated view over file systems provides a useful tool for making file system access less operation-system dependent, and the adaptor concept adds the utility of accessing file metadata and content. Thus, FSX turns a file system into a (rather slow) XML database.

6. REFERENCE

[1] Kaspar Giger and Erik Wilde. XPath Filename Expansion in a Unix Shell. In *Poster Proceedings of the 15th International World Wide Web Conference*, Edinburgh, UK, May 2006. ACM Press.