Metaphor

Concept

Metadata is a hot topic in computer science. Apple recently introduced a metadata search engine in Mac OS X 10.4 and Microsoft has been working on metadata tagging and search for Vista. An open source project named Beagle brings such features to Linux. These metadata systems allow a user to tag files with arbitrary metadata. Pictures, for example, can be given a tag indicating where they were taken, and this information can later be used in searches. These systems also automatically extract metadata when possible (using ID3 and EXIF tags for instance).

Metaphor is a bridge between these metadata systems and the traditional file system. It creates a directory structure containing a hierarchical representation of metadata. Figure 1 illustrates this concept with a possible representation of two Bob Dylan albums.

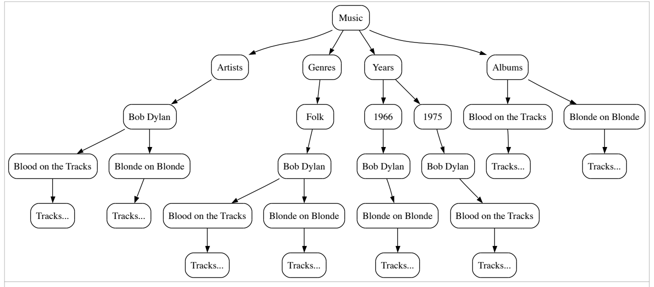

Figure 1

Tracks are referenced from several locations in the file system. If a track's metadata is altered the directory structure is updated as well. The concept works for any file that has metadata associated with it, including PDFs, Word Documents, movies, etc.

Applications

1- File Servers

Traditional file servers either have to organize their data into one fixed hierarchy or their administrators have to spend a great deal of time organizing links so that a file can be referenced from multiple locations. Maintaining links is more difficult than it seems. If a file is deleted then all of it's links need to be deleted as well. Likewise for metadata updates. Metaphor cleanly solves this problem by creating a dynamic directory structure based on metadata. Because it is mounted as a traditional file system it can be exported via Samba, NFS or any other file server.

2- Browsing

In current systems the primary interface for accessing metadata is the search field, which is not always the best solution for finding a piece of data. In some circumstances browsing is more efficient. Assume a user wants to find all the rock music on his system. This could be accomplished by doing a search, but the search results would be presented as a flat list. Even with several levels of groupings, one artist could take up several pages of results, preventing a quick glance at what rock artists are available (or what rock albums are available, etc). An interface to browse metadata, on the other hand, would present the available rock music in an easily navigatable hierarchy.

Implementation

Metaphor is implemented as a plugin for a small, high performance web server named lighttpd. This plugin creates a WebDAV (a ubiquitous network filesystem based on HTTP) server. When a file is requested it is streamed over the loopback device. All of this is totally invisible to the user. Metaphor launches and mounts itself at startup so that all the user ever sees is a new volume.

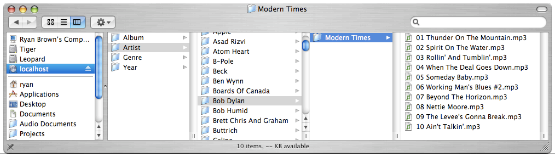

Figure 2

The current version of Metaphor is available only for Mac OS X, so much of its implementation is specific to that operating system. However the parts that are dependent on OS X either have analogues on other systems or can be re-implemented, and a Linux port is planned. On OS X, Metaphor makes extensive use of Spotlight, OS X's metadata archive and search system.

Internally Metaphor maintains two main data structures: a metadata tree and a meta-metadata tree. As should be clear from the name, the meta-metadata tree describes the metadata hierarchy. It is configurable via an easy to read XML file. While XML may not be applicable to every situation, it works well here because of how suited XML is to describing tree structures. Every node of the meta-metadata tree contains three items: a name formatter, a type and a predicate. Name formatters describe how the name of a folder or file is constructed. In the XML file a special format is used to describe name formatters. Keys are surrounded in @ symbols and all other text is considered static. For example, "[@kMDItemRecordingYear@] @kMDItemAlbum@" will generate folders with the recording year in square brackets, followed by the albums name (i.e "[1975] Blood on the Tracks"). The space and the square brackets are considered static text in this case.

Predicates are the second element in meta-metadata nodes. They use a small C-like query language which is conveniently similar to Spotlight's own query language. "kMDItemAlbum != nil", for example, means that an item must have an album tag in order to be filed into a given node. BLOBing for strings is possible as are compound predicates built using ORs and ANDs. Predicates are ANDed at each level of the tree so that a leaf node's predicate is the product of it's ancestors. A node can have one of three types: static folder (meaning its name does not change), dynamic folder (folders with keys in the name) or file (which are always dynamic). See figures 3 and 4 for a sample XML file and a graph representation of the data structure it is parsed into.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Note that things have been simplified for example's sake -->

<tree>
 <node name="Music" predicate="kMDItemContentTypeTree CONTAINS 'public.audio'">
  <node name="Artists" predicate="kMDItemAuthors != nil">
   <node name="@kMDItemAuthors@">
    <node name="[@kMDItemRecordingYear@] @kMDItemAlbum@" predicate="kMDItemAlbum != nil">
     <leaf name="@kMDItemDisplayName@" />
    </node>
   </node>
  </node>

  <node name="Genres" predicate="kMDItemMusicalGenre != nil">
   <node name="@kMDItemMusicalGenre@">
    <node name="@kMDItemAuthors@" predicate="kMDItemAuthors != nil">
     <node name="[@kMDItemRecordingYear@] @kMDItemAlbum@" predicate="kMDItemAlbum != nil">
      <leaf name="@kMDItemDisplayName@" />
     </node>
    </node>
   </node>
  </node>
 </node>
</tree>
```

Figure 3

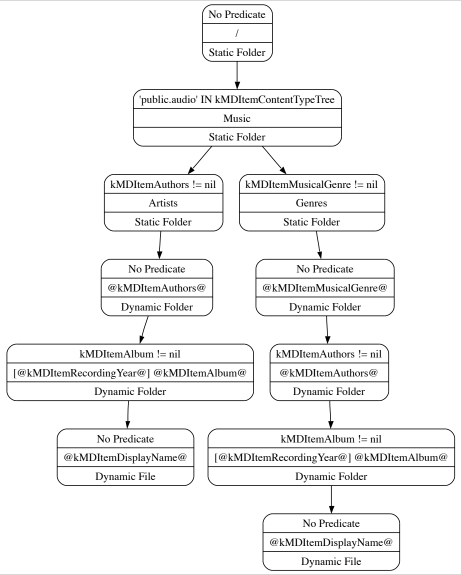

Figure 4

The first implementation of Metaphor attempted to query Spotlight in real time as WebDAV requests were made. This approach turned out to be far too slow to provide a good user experience. The solution was to run Metaphor as a daemon and create a local copy of relevant metadata. When files are added, deleted, or updated, Metaphor is notified and it's copy is changed. At shutdown the metadata tree is serialized into a file and at startup it is loaded into memory.

The actual metadata structure is fairly obvious; it is essentially a copy of the directory tree. Every node contains two items: a pointer to a node in the meta-metadata tree and a hash table for metadata keys and values. When a request is made the tree is traversed until the requested path is found. This entails asking every node encountered in the traversal to construct a name using it's name formatter. All the keys referenced in the name formatter must be requested from the hash table. A string comparison is then done between the constructed name and the current path component. If a match is made either another traversal takes place or the current node is returned to the WebDAV client.

The first step in initially propagating the metadata tree (which should only take place once) is to analyze the meta-metadata tree and identify a common predicate. This predicate must be applicable to all nodes in the tree and is often something like, "find all movies, music or documents". Once this minimum query is established Spotlight is queried and it's results are iterated through. Propagating an item into the tree entails traversing through each node in the tree and testing the item against a recursively built predicate. If a match is not made for a given node then it is not traversed further. If a match is made then the item is added to the current node.

Problems

1- Cataloguing Applications

Applications like iTunes and Direct Connect that keep references to paths are fundamentally incompatible with Metaphor. If a file's metadata changes then Metaphor's directory hierarchy changes in response, breaking these references. In addition to this, most cataloging applications will import each reference to a file separately, which can quickly fill up a catalog with duplicates.

2- The Loopback Device

Streaming files over the loopback device has a negative effect on the transfer rate. Several solutions to this problem exist, all of which involve ditching WebDAV. The best (although most difficult) solution is to develop a kernel level VFS plugin. Files in the VFS would actually be links to files on other volumes, causing the system to read directly from files instead of through a proxy. A similar solution could be developed by modifying a light weight, cross-volume link supporting NFS server.

3- Dealing with the Finder

When the metadata tree is updated, the Finder (OS X's file browser) needs to be updated as well. Because of the nature of HTTP it isn't possible for a WebDAV server to notify a client when updates are made. In addition to this, the Finder implements an erratic caching scheme which can prevent a user from seeing updates at all (even if a folder is reloaded). The current solution is a hack: the Finder is asked to refresh directories by making AppleScript (a scripting language implemented in many OS X's applications) calls to the Finder. Unfortunately this solution is slow and unreliable.

4- Duplicates

Duplicates can occur if two files or directories have the same due to a match-up in metadata attributes. Metaphor was designed with this in mind and will eventually support appending numbers to the end of names to allow access to duplicates when they occur.

Conclusion and Future Directions

"Plan to throw one away; you will, anyhow" -Fred Brooks, The Mythical Man Month
"If you plan to throw one away, you will throw away two." -Craig Zerouni

As is often the case with software, a significantly better approach became apparent only after attempting an initial implementation. If a rewrite was attempted a real VFS plugin would most likely be developed. Doing so would allow the Finder and the rest of the system to handle live updates using native kernel notification system. A VFS plugin would also improve both browsing speed and transfer rate. By using cross-volume links the catalog problem could be eliminated, assuming that the application followed links.

Metaphor in it's current state is however, usable. The data structures and core concepts are sound and it is a good solution for file servers or individuals that want a clean way to organize a diverse set of data. As operating systems continue to develop their use of metadata these ideas will be increasingly relevant.

wabdo.com/metaphor

Project By Ryan Brown
Sponsored By Richard Weiss

EVERGREEN
THE EVERGREEN STATE COLLEGE
OLYMPIA, WASHINGTON