12-2012

Filesystems: Addressing the Last-mile "problem" in Services-Oriented/Cloud Computing

George K. Thiruvathukal
Loyola University Chicago, gkt@cs.luc.edu

Joseph P. Kaylor

Konstantin Läufer
Loyola University Chicago, klaeufer@gmail.com

Userland Filesystems: Addressing the Last-mile "problem" in Services-Oriented/Cloud Computing

George K. Thiruvathukal (speaker), Joe Kaylor, and Konstantin Läufer

Presented at the 6th Scientific Software Days, Austin, TX (17 December 2012)

History

This is a quick summary of my efforts to raise the abstraction level in systems programming, especially in the parallel/distributed area:

- task coordination systems: Memo, Enhanced Actors (Ph.D. IIT 1995)
- process parallelism/grid computing: MPICH Globus
- cluster computing: Java NOW, Computational Neighborhood
- Java interfaces to Nexus communication system in Globus
- concurrent/parallel + Java: HPJPC class library and book
- web programming + Python
- present talk: *userland filesystems: Hydra P2P, OLFS, NOFS, RestFS*
- MPI-IO scalability atomicity (scalable distributed byte-range lockserver in Java)

. . . and some other non-systems efforts:

- digital humanities/XML and electronic editing cloud tools
- digital humanities/platform studies: MIT Press

Filesystems

- Still used in most computational science applications
- much ad hoc transfer and processing of data sets
- great need to leverage distributed principles to provide transparent access

Why Userland Filesystems?

- Really an extension of the separation of concerns that exists in micro-kernel and modular operating systems.
- In addition to separating virtual memory, device drivers, and other components, filesystems are typically a separate process or service in such operating systems
- With a separated process, the tasks of debugging, testing, stubbing out components in unit tests, and other development tasks are greatly simplified

The Filesystem as a Connector Abstraction

- In 2010, we built RestFS, a filesystem using the NOFS framework to expose web services as filesystems.
  - the "last mile" of RESTful services is being able to interact with them
  - a task often left to a browser or proprietary desktop client (e.g. Dropbox, Box, etc.)
  - our view: the command-line + a mounted filesystem is another option
- Our prototype implementation is able to:
  - Perform Google searches by creating files with the name containing the search terms with RestFS filling the file's contents with JSON search results
  - Perform Yahoo! Placefinder lookups
  - Connect to Twitter and list a user's Tweets
  - Connect to services using OAuth authentication
- Since then, we have been developing more of our filesystems using NOFS (this talk)
  - SketchFabFS: a filesystem that allows for transparent access to SketchFab.com for rendering 3D models
  - Users upload model files (e.g. model.ply) and can check for the status (model.uploading), completion (model.done), and results (model.ply.html).
  - No need to wait for the result to appear in the browser (by staying on the sketchfab.com page)

Why are Userland Filesystems Good For FOSS?

- As with other userland programs, user-mode OS services can be versioned and released separately from the privileged OS kernel (and without

tainting the kernel).

- User-mode OS services are more friendly to FOSS. In disconnected, distributed development communities it can be simpler to get people to use a separate package and service than to convince an existing project to accept a new implementation.
- Well-known example of SSHFS allows a remote collection of folders/files to be "mounted" over SSH (as opposed to NFS)

Userland Filesystems Development APIs / Frameworks

- FUSE - Filesystems in User Space
  - Available on Linux, BSD, Solaris, Mac OSX
  - Most well known and popular framework.
  - Many language bindings exist - C#, C++, Python, Java, etc..
- Dokan
  - Available on Windows XP, Vista, 7, 2003, 2008, 2008r2
  - C, C++, C# language bindings exist
- NOFS - Naked Objects Filesystem
  - Developed at Loyola University Chicago
  - Available on Windows and Linux (OS X possible but untested)
  - Java and C# editions exist
- RESTFS - RESTful File System - Built atop our NOFS framework - Specifically designed for interaction with RESTful services - Addresses web services specific issues (e.g. authentication, authorization, etc.)

Publications

NOFS and RestFS

- First Tech Report, http://works.bepress.com/gkthiruvathukal/1/
- Cloud Computing Book Chapter, http://works.bepress.com/gkthiruvathukal/48/
- WS-REST 2011 Paper, http://works.bepress.com/laufer/6/
- IEEE EIT 2012, http://ecommons.luc.edu/cs_facpubs/62/

OLFS (an early work leading to NOFS and RestFS)

- http://ecommons.luc.edu/cs_facpubs/40/

Hydra (a peer-to-peer filesystem written in Python)

- http://ecommons.luc.edu/cs_facpubs/7/

Example Pass-Through FUSE C++ Filesystem

- This example shows how to write a null (pass through) filesystem in FUSE using C/C++.
- A great deal of detail must be mastered, even to write the simplest filesystem.
- Other filesystems are much more complicated.
- Enter NOFS.

```
1
2  struct fuse_operations examplefs_oper;
3
4  int main(int argc, char *argv[]) {
5          int i, fuse_stat;
6
7          examplefs_oper.getattr = wrap_getattr;
8          examplefs_oper.readlink = wrap_readlink;
9          examplefs_oper.getdir = NULL;
10         examplefs_oper.mknod = wrap_mknod;
11         examplefs_oper.mkdir = wrap_mkdir;
12         examplefs_oper.unlink = wrap_unlink;
13         examplefs_oper.rmdir = wrap_rmdir;
14         examplefs_oper.symlink = wrap_symlink;
15         examplefs_oper.rename = wrap_rename;
16         examplefs_oper.link = wrap_link;
17         examplefs_oper.chmod = wrap_chmod;
18         examplefs_oper.chown = wrap_chown;
19         examplefs_oper.truncate = wrap_truncate;
20         examplefs_oper.utime = wrap_utime;
21         examplefs_oper.open = wrap_open;
22         examplefs_oper.read = wrap_read;
23         examplefs_oper.write = wrap_write;
24         examplefs_oper.statfs = wrap_statfs;
25         examplefs_oper.flush = wrap_flush;
26         examplefs_oper.release = wrap_release;
```

```
27          examplefs_oper.fsync = wrap_fsync;
28          examplefs_oper.setxattr = wrap_setxattr;
29          examplefs_oper.getxattr = wrap_getxattr;
30          examplefs_oper.listxattr = wrap_listxattr;
31          examplefs_oper.removexattr = wrap_removexattr;
32          examplefs_oper.opendir = wrap_opendir;
33          examplefs_oper.readdir = wrap_readdir;
34          examplefs_oper.releasedir = wrap_releasedir;
35          examplefs_oper.fsyncdir = wrap_fsyncdir;
36          examplefs_oper.init = wrap_init;
37
38          printf("mounting file system...\n");
39
40          for(i = 1; i < argc && (argv[i][0] == '-'); i++) {
41                  if(i == argc) {
42                          return (-1);
43                  }
44          }
45
46          //realpath(...) returns the canonicalized absolute pathname
47          set_rootdir(realpath(argv[i], NULL));
48
49          for(; i < argc; i++) {
50                  argv[i] = argv[i+1];
51          }
52          argc--;
53
54          fuse_stat = fuse_main(argc, argv, &examplefs_oper, NULL);
55
56          printf("fuse_main returned %d\n", fuse_stat);
57
58          return fuse_stat;
59  }
60
61
62  void set_rootdir(const char *path) {
63          ExampleFS::Instance()->setRootDir(path);
64  }
65
66  int wrap_getattr(const char *path, struct stat *statbuf) {
67          return ExampleFS::Instance()->Getattr(path, statbuf);
68  }
69
70  int wrap_readlink(const char *path, char *link, size_t size) {
71          return ExampleFS::Instance()->Readlink(path, link, size);
72  }
```

```
73
74  int wrap_mknod(const char *path, mode_t mode, dev_t dev) {
75          return ExampleFS::Instance()->Mknod(path, mode, dev);
76  }
77  int wrap_mkdir(const char *path, mode_t mode) {
78          return ExampleFS::Instance()->Mkdir(path, mode);
79  }
80  int wrap_unlink(const char *path) {
81          return ExampleFS::Instance()->Unlink(path);
82  }
83  int wrap_rmdir(const char *path) {
84          return ExampleFS::Instance()->Rmdir(path);
85  }
86  int wrap_symlink(const char *path, const char *link) {
87          return ExampleFS::Instance()->Symlink(path, link);
88  }
89  int wrap_rename(const char *path, const char *newpath) {
90          return ExampleFS::Instance()->Rename(path, newpath);
91  }
92  int wrap_link(const char *path, const char *newpath) {
93          return ExampleFS::Instance()->Link(path, newpath);
94  }
95  int wrap_chmod(const char *path, mode_t mode) {
96          return ExampleFS::Instance()->Chmod(path, mode);
97  }
98  int wrap_chown(const char *path, uid_t uid, gid_t gid) {
99          return ExampleFS::Instance()->Chown(path, uid, gid);
100 }
101 int wrap_truncate(const char *path, off_t newSize) {
102         return ExampleFS::Instance()->Truncate(path, newSize);
103 }
104 int wrap_utime(const char *path, struct utimbuf *ubuf) {
105         return ExampleFS::Instance()->Utime(path, ubuf);
106 }
107 int wrap_open(const char *path, struct fuse_file_info *fileInfo) {
108         return ExampleFS::Instance()->Open(path, fileInfo);
109 }
110 int wrap_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info
111         return ExampleFS::Instance()->Read(path, buf, size, offset, fileInfo);
112 }
113 int wrap_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_fil
114         return ExampleFS::Instance()->Write(path, buf, size, offset, fileInfo);
115 }
116 int wrap_statfs(const char *path, struct statvfs *statInfo) {
117         return ExampleFS::Instance()->Statfs(path, statInfo);
118 }
```

```
119 int wrap_flush(const char *path, struct fuse_file_info *fileInfo) {
120         return ExampleFS::Instance()->Flush(path, fileInfo);
121 }
122 int wrap_release(const char *path, struct fuse_file_info *fileInfo) {
123         return ExampleFS::Instance()->Release(path, fileInfo);
124 }
125 int wrap_fsync(const char *path, int datasync, struct fuse_file_info *fi) {
126         return ExampleFS::Instance()->Fsync(path, datasync, fi);
127 }
128 int wrap_setxattr(const char *path, const char *name, const char *value, size_t size, int fl
129         return ExampleFS::Instance()->Setxattr(path, name, value, size, flags);
130 }
131 int wrap_getxattr(const char *path, const char *name, char *value, size_t size) {
132         return ExampleFS::Instance()->Getxattr(path, name, value, size);
133 }
134 int wrap_listxattr(const char *path, char *list, size_t size) {
135         return ExampleFS::Instance()->Listxattr(path, list, size);
136 }
137 int wrap_removexattr(const char *path, const char *name) {
138         return ExampleFS::Instance()->Removexattr(path, name);
139 }
140 int wrap_opendir(const char *path, struct fuse_file_info *fileInfo) {
141         return ExampleFS::Instance()->Opendir(path, fileInfo);
142 }
143 int wrap_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct f
144         return ExampleFS::Instance()->Readdir(path, buf, filler, offset, fileInfo);
145 }
146 int wrap_releasedir(const char *path, struct fuse_file_info *fileInfo) {
147         return ExampleFS::Instance()->Releasedir(path, fileInfo);
148 }
149 int wrap_fsyncdir(const char *path, int datasync, struct fuse_file_info *fileInfo) {
150         return ExampleFS::Instance()->Fsyncdir(path, datasync, fileInfo);
151 }
152 int wrap_init(struct fuse_conn_info *conn) {
153         return ExampleFS::Instance()->Init(conn);
154 }
155
156 ExampleFS* ExampleFS::_instance = NULL;
157
158 #define RETURN_ERRNO(x) (x) == 0 ? 0 : -errno
159
160 ExampleFS* ExampleFS::Instance() {
161         if(_instance == NULL) {
162                 _instance = new ExampleFS();
163         }
164         return _instance;
```

```
165 }
166
167 ExampleFS::ExampleFS() {
168
169 }
170
171 ExampleFS::~ExampleFS() {
172
173 }
174
175 void ExampleFS::AbsPath(char dest[PATH_MAX], const char *path) {
176         strcpy(dest, _root);
177         strncat(dest, path, PATH_MAX);
178         //printf("translated path: %s to %s\n", path, dest);
179 }
180
181 void ExampleFS::setRootDir(const char *path) {
182         printf("setting FS root to: %s\n", path);
183         _root = path;
184 }
185
186 int ExampleFS::Getattr(const char *path, struct stat *statbuf) {
187         char fullPath[PATH_MAX];
188         AbsPath(fullPath, path);
189         printf("getattr(%s)\n", fullPath);
190         return RETURN_ERRNO(lstat(fullPath, statbuf));
191 }
192
193 int ExampleFS::Readlink(const char *path, char *link, size_t size) {
194         printf("readlink(path=%s, link=%s, size=%d)\n", path, link, (int)size);
195         char fullPath[PATH_MAX];
196         AbsPath(fullPath, path);
197         return RETURN_ERRNO(readlink(fullPath, link, size));
198 }
199
200 int ExampleFS::Mknod(const char *path, mode_t mode, dev_t dev) {
201         printf("mknod(path=%s, mode=%d)\n", path, mode);
202         char fullPath[PATH_MAX];
203         AbsPath(fullPath, path);
204
205         //handles creating FIFOs, regular files, etc...
206         return RETURN_ERRNO(mknod(fullPath, mode, dev));
207 }
208
209 int ExampleFS::Mkdir(const char *path, mode_t mode) {
210         printf("**mkdir(path=%s, mode=%d)\n", path, (int)mode);
```

```
211         char fullPath[PATH_MAX];
212         AbsPath(fullPath, path);
213         return RETURN_ERRNO(mkdir(fullPath, mode));
214 }
215
216 int ExampleFS::Unlink(const char *path) {
217         printf("unlink(path=%s\n)", path);
218         char fullPath[PATH_MAX];
219         AbsPath(fullPath, path);
220         return RETURN_ERRNO(unlink(fullPath));
221 }
222
223 int ExampleFS::Rmdir(const char *path) {
224         printf("rmkdir(path=%s\n)", path);
225         char fullPath[PATH_MAX];
226         AbsPath(fullPath, path);
227         return RETURN_ERRNO(rmdir(fullPath));
228 }
229
230 int ExampleFS::Symlink(const char *path, const char *link) {
231         printf("symlink(path=%s, link=%s)\n", path, link);
232         char fullPath[PATH_MAX];
233         AbsPath(fullPath, path);
234         return RETURN_ERRNO(symlink(fullPath, link));
235 }
236
237 int ExampleFS::Rename(const char *path, const char *newpath) {
238         printf("rename(path=%s, newPath=%s)\n", path, newpath);
239         char fullPath[PATH_MAX];
240         AbsPath(fullPath, path);
241         return RETURN_ERRNO(rename(fullPath, newpath));
242 }
243
244 int ExampleFS::Link(const char *path, const char *newpath) {
245         printf("link(path=%s, newPath=%s)\n", path, newpath);
246         char fullPath[PATH_MAX];
247         char fullNewPath[PATH_MAX];
248         AbsPath(fullPath, path);
249         AbsPath(fullNewPath, newpath);
250         return RETURN_ERRNO(link(fullPath, fullNewPath));
251 }
252
253 int ExampleFS::Chmod(const char *path, mode_t mode) {
254         printf("chmod(path=%s, mode=%d)\n", path, mode);
255         char fullPath[PATH_MAX];
256         AbsPath(fullPath, path);
```

```
257         return RETURN_ERRNO(chmod(fullPath, mode));
258 }
259
260 int ExampleFS::Chown(const char *path, uid_t uid, gid_t gid) {
261         printf("chown(path=%s, uid=%d, gid=%d)\n", path, (int)uid, (int)gid);
262         char fullPath[PATH_MAX];
263         AbsPath(fullPath, path);
264         return RETURN_ERRNO(chown(fullPath, uid, gid));
265 }
266
267 int ExampleFS::Truncate(const char *path, off_t newSize) {
268         printf("truncate(path=%s, newSize=%d\n", path, (int)newSize);
269         char fullPath[PATH_MAX];
270         AbsPath(fullPath, path);
271         return RETURN_ERRNO(truncate(fullPath, newSize));
272 }
273
274 int ExampleFS::Utime(const char *path, struct utimbuf *ubuf) {
275         printf("utime(path=%s)\n", path);
276         char fullPath[PATH_MAX];
277         AbsPath(fullPath, path);
278         return RETURN_ERRNO(utime(fullPath, ubuf));
279 }
280
281 int ExampleFS::Open(const char *path, struct fuse_file_info *fileInfo) {
282         printf("open(path=%s)\n", path);
283         char fullPath[PATH_MAX];
284         AbsPath(fullPath, path);
285         fileInfo->fh = open(fullPath, fileInfo->flags);
286         return 0;
287 }
288
289 int ExampleFS::Read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file
290         printf("read(path=%s, size=%d, offset=%d)\n", path, (int)size, (int)offset);
291         return RETURN_ERRNO(pread(fileInfo->fh, buf, size, offset));
292 }
293
294 int ExampleFS::Write(const char *path, const char *buf, size_t size, off_t offset, struct fu
295         printf("write(path=%s, size=%d, offset=%d)\n", path, (int)size, (int)offset);
296         return RETURN_ERRNO(pwrite(fileInfo->fh, buf, size, offset));
297 }
298
299 int ExampleFS::Statfs(const char *path, struct statvfs *statInfo) {
300         printf("statfs(path=%s)\n", path);
301         char fullPath[PATH_MAX];
302         AbsPath(fullPath, path);
```

```
303         return RETURN_ERRNO(statvfs(fullPath, statInfo));
304 }
305
306 int ExampleFS::Flush(const char *path, struct fuse_file_info *fileInfo) {
307         printf("flush(path=%s)\n", path);
308         //noop because we don't maintain our own buffers
309         return 0;
310 }
311
312 int ExampleFS::Release(const char *path, struct fuse_file_info *fileInfo) {
313         printf("release(path=%s)\n", path);
314         return 0;
315 }
316
317 int ExampleFS::Fsync(const char *path, int datasync, struct fuse_file_info *fi) {
318         printf("fsync(path=%s, datasync=%d\n", path, datasync);
319         if(datasync) {
320                 //sync data only
321                 return RETURN_ERRNO(fdatasync(fi->fh));
322         } else {
323                 //sync data + file metadata
324                 return RETURN_ERRNO(fsync(fi->fh));
325         }
326 }
327
328 int ExampleFS::Setxattr(const char *path, const char *name, const char *value, size_t size,
329         printf("setxattr(path=%s, name=%s, value=%s, size=%d, flags=%d\n",
330                 path, name, value, (int)size, flags);
331         char fullPath[PATH_MAX];
332         AbsPath(fullPath, path);
333         return RETURN_ERRNO(lsetxattr(fullPath, name, value, size, flags));
334 }
335
336 int ExampleFS::Getxattr(const char *path, const char *name, char *value, size_t size) {
337         printf("getxattr(path=%s, name=%s, size=%d\n", path, name, (int)size);
338         char fullPath[PATH_MAX];
339         AbsPath(fullPath, path);
340         return RETURN_ERRNO(getxattr(fullPath, name, value, size));
341 }
342
343 int ExampleFS::Listxattr(const char *path, char *list, size_t size) {
344         printf("listxattr(path=%s, size=%d)\n", path, (int)size);
345         char fullPath[PATH_MAX];
346         AbsPath(fullPath, path);
347         return RETURN_ERRNO(llistxattr(fullPath, list, size));
348 }
```

```
349
350 int ExampleFS::Removexattr(const char *path, const char *name) {
351         printf("removexattry(path=%s, name=%s)\n", path, name);
352         char fullPath[PATH_MAX];
353         AbsPath(fullPath, path);
354         return RETURN_ERRNO(lremovexattr(fullPath, name));
355 }
356
357 int ExampleFS::Opendir(const char *path, struct fuse_file_info *fileInfo) {
358         printf("opendir(path=%s)\n", path);
359         char fullPath[PATH_MAX];
360         AbsPath(fullPath, path);
361         DIR *dir = opendir(fullPath);
362         fileInfo->fh = (uint64_t)dir;
363         return NULL == dir ? -errno : 0;
364 }
365
366 int ExampleFS::Readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, st
367         printf("readdir(path=%s, offset=%d)\n", path, (int)offset);
368         DIR *dir = (DIR*)fileInfo->fh;
369         struct dirent *de = readdir(dir);
370         if(NULL == de) {
371                 return -errno;
372         } else {
373                 do {
374                         if(filler(buf, de->d_name, NULL, 0) != 0) {
375                                 return -ENOMEM;
376                         }
377                 } while(NULL != (de = readdir(dir)));
378         }
379         return 0;
380 }
381
382 int ExampleFS::Releasedir(const char *path, struct fuse_file_info *fileInfo) {
383         printf("releasedir(path=%s)\n", path);
384         closedir((DIR*)fileInfo->fh);
385         return 0;
386 }
387
388 int ExampleFS::Fsyncdir(const char *path, int datasync, struct fuse_file_info *fileInfo) {
389         return 0;
390 }
391
392 int ExampleFS::Init(struct fuse_conn_info *conn) {
393         return 0;
394 }
```

```
395
396 int ExampleFS::Truncate(const char *path, off_t offset, struct fuse_file_info *fileInfo) {
397         printf("truncate(path=%s, offset=%d)\n", path, (int)offset);
398         char fullPath[PATH_MAX];
399         AbsPath(fullPath, path);
400         return RETURN_ERRNO(ftruncate(fileInfo->fh, offset));
401 }
```

Example Pass-Through NOFS C# Filesystem

- NOFS simplifies the immediately preceding code by focusing on folders/files and the actual *domain* you are trying to interface to NOFS.
- Instead of requiring the user to think about all the innards of Posix interfaces, users instead focus on mapping objects to/from the filesystem via *annotations*.
- The classes shown here map naturally to the abstractions that are most visible in a filesystem instead of the low-level operations performed by Posix interfaces.
- Disclosure: We don't support all of the interfaces yet. Early work tells us that most application-oriented FUSE filesystems don't need all of the FUSE interfaces. This is a subject of further research.

```
1
2  using System;
3  using nofs.attributes;
4  using nofs.containers.interfaces;
5
6  namespace storagefs
7  {
8      [RootFolder]
9      public class FsRoot : FsFolder
10     {
11         public FsRoot() : base("", Guid.NewGuid().ToString()){}
12         public FsRoot(string name, string id) : base(name,id){}
13     }
14
15     [DomainObject]
16     public abstract class FsFolderOrFile : IObjectWithID
17     {
18         private string _name;
19         [NeedsContainer] public IDomainObjectContainer Container { get; set; }
20         public string Id { get; set; }
```

```
21
22          protected FsFolderOrFile(){
23              Id = Guid.NewGuid().ToString();
24          }
25
26          protected FsFolderOrFile(string name, string id){
27              _name = name;
28              Id = id;
29          }
30
31          [ProvidesName]
32          public string Name {
33              get { return _name; }
34              set {
35                  _name = value;
36                  if (Container != null) {
37                      Container.ObjectChanged(this);
38                  }
39              }
40          }
41      }
42
43      [FolderObject(ChildTypeFilterMethod = "Filter")]
44      [ProvidesMappingDetails(typeof(FsFolderMapper))]
45      public class FsFolder : FsFolderOrFile, IWeakReferenceList<FsFolderOrFile>
46      {
47          private readonly WeakReferenceList<FsFolderOrFile> _list;
48
49          public FsFolder() : base(){
50              _list = new WeakReferenceList<FsFolderOrFile>(this);
51          }
52
53          public FsFolder(string name, string id) : base(name, id){
54              _list = new WeakReferenceList<FsFolderOrFile>(this);
55          }
56
57          public bool Filter(Type possibleChildtype){
58              return
59                  possibleChildtype == typeof(FsFolder) ||
60                  possibleChildtype == typeof(FsFile);
61          }
62
63          [NeedsContainerManager]
64          public void SetContainer(IDomainObjectContainerManager container){
65              _list.setContainer(container);
66          }
```

```
67
68          public IEnumerator<IWeakReference> GetEnumerator(){
69              return _list.GetEnumerator();
70          }
71
72          IEnumerator IEnumerable.GetEnumerator(){
73              return GetEnumerator();
74          }
75
76          public int Count{
77              get { return _list.Count; }
78          }
79
80          public void Add(IWeakReference item){
81              _list.Add(item);
82          }
83
84          public void Add(object item){
85              _list.Add(item);
86          }
87
88          public void Remove(IWeakReference item){
89              _list.Remove(item);
90          }
91
92          public void Remove(object item){
93              _list.Remove(item);
94          }
95
96          public void Add(FsFolderOrFile item){
97              _list.Add(item);
98          }
99
100         public void Remove(FsFolderOrFile item){
101             _list.Remove(item);
102         }
103
104         public IEnumerable<FsFolderOrFile> GetAll(){
105             return _list.GetAll();
106         }
107     }
108
109     [ProvidesMappingDetails(typeof(FsFileMapper))]
110     public class FsFile : FsFolderOrFile, IProvidesUnstructuredData
111     {
112         public FsFile() : base(){}
```

```
        public FsFile(string name, string id) : base(name, id){}

        private IDomainObjectRawDataStore _data;
        [NeedsRawDataStore]
        public void SetDataStore(IDomainObjectRawDataStore data){
            _data = data;
        }

        public long DataSize(){
            return _data.DataSize();
        }

        public bool Cacheable(){
            return false;
        }

        public int Read(byte[] buffer, long offset, long length){
            return _data.Read(buffer, offset, length);
        }

        public int Write(byte[] buffer, long offset, long length){
            return _data.Write(buffer, offset, length);
        }

        public void Truncate(long length){
            _data.Truncate(length);
        }
    }

    public sealed class FsFileMapper : IProvidesMappingDetails
    {
        public string GetXMLRepresentation(object obj){
            var file = (FsFile) obj;
            var xml = string.Format("<File name=\"{0}\" id=\"{1}\"></File>", file.Name, file
            return xml;
        }

        public object ConstructNew(string xml){
            var doc = new XmlDocument();
            doc.LoadXml(xml);
            var root = doc.ChildNodes[0];
            var name = root.Attributes["name"].Value;
            var id = root.Attributes["id"].Value;
            return new FsFile(name, id);
        }
    }
```

```
159
160     public sealed class FsFolderMapper : IProvidesMappingDetails
161     {
162         private IDomainObjectContainerManager _manager;
163
164         [NeedsContainerManager]
165         public void SetContainerManger(IDomainObjectContainerManager manager){
166             _manager = manager;
167         }
168
169         public string GetXMLRepresentation(object obj){
170             var folder = (FsFolder) obj;
171             StringBuilder xml = new StringBuilder();
172             xml.AppendFormat("<Folder name=\"{0}\" id=\"{1}\" root=\"{2}\">",
173                              folder.Name, folder.Id, (folder is FsRoot));
174             foreach (var item in folder){
175                 xml.AppendFormat("<Child id=\"{0}\" type=\"{1}\"></Child>", item.Id, item.Ur
176             }
177             xml.Append("</Folder>");
178             var xmlText = xml.ToString();
179             return xmlText;
180         }
181
182         public object ConstructNew(string xml){
183             var doc = new XmlDocument();
184             doc.LoadXml(xml);
185             var root = doc.ChildNodes[0];
186             var name = root.Attributes["name"].Value;
187             var id = root.Attributes["id"].Value;
188             var isRoot = bool.Parse(root.Attributes["root"].Value);
189             FsFolder folder;
190             if (isRoot){
191                 folder = new FsRoot(name, id);
192             } else{
193                 folder = new FsFolder(name, id);
194             }
195
196             foreach (var childNode in root.ChildNodes.OfType<XmlElement>()){
197                 id = childNode.Attributes["id"].Value;
198                 var typeName = childNode.Attributes["type"].Value;
199                 IDomainObjectContainer container;
200                 if(typeName == typeof(FsFile).Name){
201                     container = _manager.GetContainer(typeof (FsFile));
202                 } else if (typeName == typeof(FsFolder).Name){
203                     container = _manager.GetContainer(typeof (FsFolder));
204                 } else{
```

```
205                     throw new Exception("unknown child type name: " + typeName);
206                 }
207                 var weakRef = container.GetWeakReference(id);
208                 folder.Add(weakRef);
209             }
210             return folder;
211         }
212     }
213
214 }
```

Wiring Objects to FUSE with NOFS

- NOFS exposes a set of interfaces and attributes that a domain model can use to expose itself to the filesystem
- Filesystem root objects are marked with the [RootFolder] attribute
- Folders can be either:
  - methods with return type ICollection and marked with [FolderObject] attribute
  - classes that implement ICollection and are marked with [FolderObject] attribute
- Files are any class marked with [DomainObject] attribute but not marked with any other attribute
- Files by default are serialized / deserialized to / from XML automatically by NOFS
  - The user can provide their own custom serialization code to export to other formats like CSV, JSON, etc. . .
- If files implement IProvidesUnstructuredData, they can handle their own representation

FUSE vs. NOFS

- NOFS filesystems on average can be expressed more concisely than FUSE filesystems
- NOFS glues an object model to the FUSE contract. The object model can handle as many or as few details as desired.
- In the first example, the FS was storage oriented and managed read, write, truncate calls.
- In the second example, the greater concern was managing and keeping external data sources up to date. How IO is handled is managed by NOFS. In this case the objects are translated to/from XML.

- A great part of performance is managed by NOFS.
  - NOFS manages caching
  - NOFS manages object life-cycles
  - These are extra details that need to be managed by any direct use of FUSE.

Naked Objects

- The typical enterprise application pattern separates presentation, task/controller, domain model, and persistence into four separate layers. The developer is responsible for providing code or markup for each layer.
- In Naked Objects frameworks, The developer provides code for the controller + domain model (a.k.a. behaviorally complete domain model) and the framework provides the presentation and persistence layers.

Naked Objects - User Interfaces

- Naked Objects frameworks expose object oriented domain models as object oriented user interfaces
- OO-UIs are:
  - Not process oriented
  - Are principally concerned with object creation, management, and associations between/among objects
  - Treat user as a "problem solver"
- OO-UIs work very well when:
  - User is very familiar with business domain model and can be treated as a problem solver
  - Business requirements and associations are subject to frequent change
  - Or when the natural metaphor is itself object oriented
- OO-UI vs. Model-View-Controller
  - MVC can be said to encourage dilution of business logic to the controller or even the view.
  - Think about validation in the GUI as an example.
  - Pawson: "MVC was an outgrowth of the original direct-manipulation metaphor popularized in early OO practice .... where you want the objects on the screen to be the objects in the program. MVC actually works against that metaphor but evolved as a necessary evil. Why? Because the user object maintains multiple simultaneous views of the model at once; the factoring into user, model, view, and controller allows one to support that."

  - Pawson: “close coupling of views and controllers to a model. Both view and controller components make direct calls to the model. This implies that changes to the model’s interface are likely to break the code for both view and controller. This problem is magnified if the system uses a multitude of views and controllers.”

Naked Objects - Why?

- Naked Objects frameworks discourages the separation of business logic and data
- Behaviorally complete objects
- Faster development cycle
- Common language between developers and users (thanks the the OO UI)
- A more empowering user interface (debatable)

Naked Objects - Why Not?

- Service Oriented Architectures
  - Encourage the separation of data and business logic
  - Encourage stateless code and services
- OO-UIs aren’t a good match for some applications
- Strictly process oriented applications like “grep” or “find” don’t make as much sense with Naked Objects frameworks.

Naked Objects - Filesystems

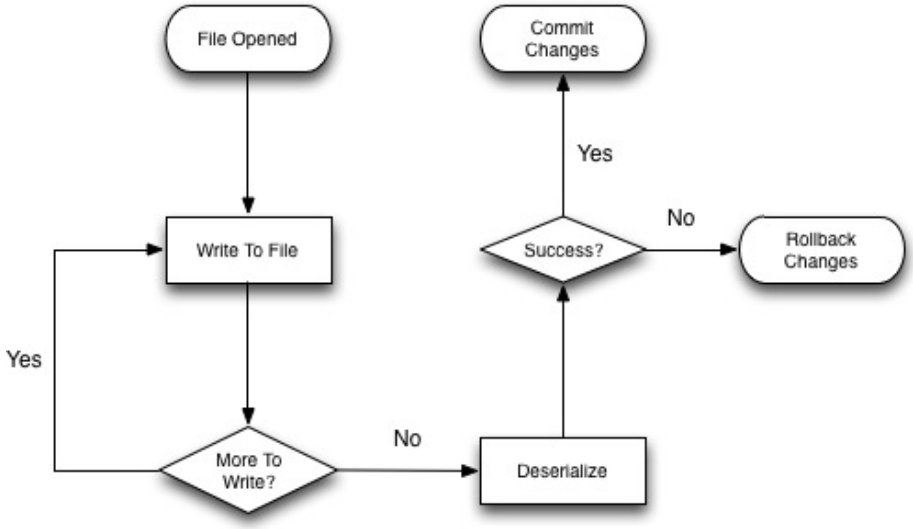
- The filesystem itself is very much object oriented.
- Files and folders are objects with associations to each other.
- The user is treated as a problem solver - decides when / where to create, move, delete files.
- There are no processes to follow in terms of what a filesystem service provides.
- In these senses, the FS is itself an OO-UI.

Naked Objects - NOFS

- NOFS is a framework that takes behaviorally complete domain models and translates them into filesystems.
- NOFS provides the glue code between FUSE and the domain model.

- Where other Naked-Objects frameworks would present an OO-UI on a monitor or as a web-page, NOFS presents a filesystem as the user-interface component.

NOFS Object Serialization



A Simple Application Filesystem in NOFS

```
1
2   using System;
3   using System.Runtime.Serialization;
4   using System.Reflection;
5   using System.Net;
6   using System.IO;
7   using System.Text;
8   using nofs.attributes;
9   using nofs.application;
10  using nofs.containers.interfaces;
11
12  namespace stocks
13  {
14      class Program
15      {
16          static void Main(string[] args) {
```

```
            if(args.Length != 2){
                Console.WriteLine("usage: ");
                Console.WriteLine("stocks.exe [mount point] [db folder]");
            } else{
                var application = NofsApplicationFactory.CreateApplication();
                var persistenceFactory = new PersistenceFactoryFactory(
                    Assembly.Load("nofs.db4o"), "nofs.db4o.factories.PersistenceFactory"
                    );
                var mountPoint = args[0];
                var dbFolder = args[1];
                application.StartFileSystem(
                    persistenceFactory, typeof(Program).Assembly,
                    mountPoint, dbFolder, "Stock Portfolio");
            }
        }
    }


    [RootFolder]
    [DomainObject]
    [FolderObject]
    public class Portfolio
    {
        private IDomainObjectContainerManager _containerManager;
        private List<Stock> _stocks;

        public Portfolio() {
            _stocks = new List<Stock>();
        }

        [NeedsContainerManager]
        public IDomainObjectContainerManager ContainerManager {
            set {
                _containerManager = value;
            }
        }

        [FolderObject]
        public IEnumerable<Stock> Stocks() {
            UpdateStockData();
            return _stocks;
        }

        public void AddAStockForTesting(Stock stock) {
            _stocks.Add(stock);
        }
```

```
63
64          private void UpdateStockData() {
65              String url = BuildURL();
66              List<String> dataLines = getDataFromURL(url);
67              foreach (Stock stock in _stocks) {
68                  String dataLine = null;
69                  foreach (String line in dataLines) {
70                      if (line.StartsWith("\"" + stock.Ticker)) {
71                          dataLine = line;
72                          break;
73                      }
74                  }
75                  if (dataLine != null) {
76                      stock.UpdateData(dataLine);
77                  }
78              }
79          }
80
81          private String BuildURL() {
82              List<string> tickers = new List<string>();
83              foreach (Stock stock in _stocks) {
84                  tickers.Add(stock.Ticker);
85              }
86              StringBuilder url = new StringBuilder();
87              url.Append(@"http://download.finance.yahoo.com/d/quotes.csv?s=");
88              url.Append(string.Join(",", tickers.ToArray()));
89              url.Append("&f=sl1d1t1c1ohgv&e=.csv");
90              return url.ToString();
91          }
92
93          private List<String> getDataFromURL(String url) {
94              Uri uri = new Uri(url);
95              HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
96              request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethoc
97              request.KeepAlive = true;
98              request.Timeout = 10 * 60 * 1000;
99
100             byte[] array = null;
101
102             using (HttpWebResponse response = (HttpWebResponse)request.GetResponse()) {
103                 using (Stream os = response.GetResponseStream()) {
104                     using (MemoryStream ms = new MemoryStream()) {
105                         const int bufferLength = 1024;
106                         byte[] buf = new byte[bufferLength];
107                         int read = 0;
108                         while ((read = os.Read(buf, 0, bufferLength)) > 0) {
```

```
                            ms.Write(buf, 0, read);
                        }
                        array = ms.ToArray();
                    }
                }
            }

            string s = new UTF8Encoding(true, true).GetString(array);

            List<String> lines = new List<String>();
            foreach (String line in s.Split("\r\n".ToCharArray())) {
                lines.Add(line);
            }
            return lines;
        }

        [Executable]
        public void AddAStock(String ticker) {
            var stockContainer = _containerManager.GetContainer(typeof(Stock));
            Stock stock = stockContainer.NewPersistentInstance() as Stock;
            stock.Ticker = ticker;
            _stocks.Add(stock);
            stockContainer.ObjectChanged(stock);
            _containerManager.GetContainer(typeof(Portfolio)).ObjectChanged(this);
        }
    }


    [DomainObject]
    [DataContract]
    public class Stock
    {
        public Stock() {
        }

        public Stock(String ticker) {
            Ticker = ticker;
        }

        [ProvidesName]
        public String Ticker {
            get;
            set;
        }

        public void UpdateData(String data) {
```

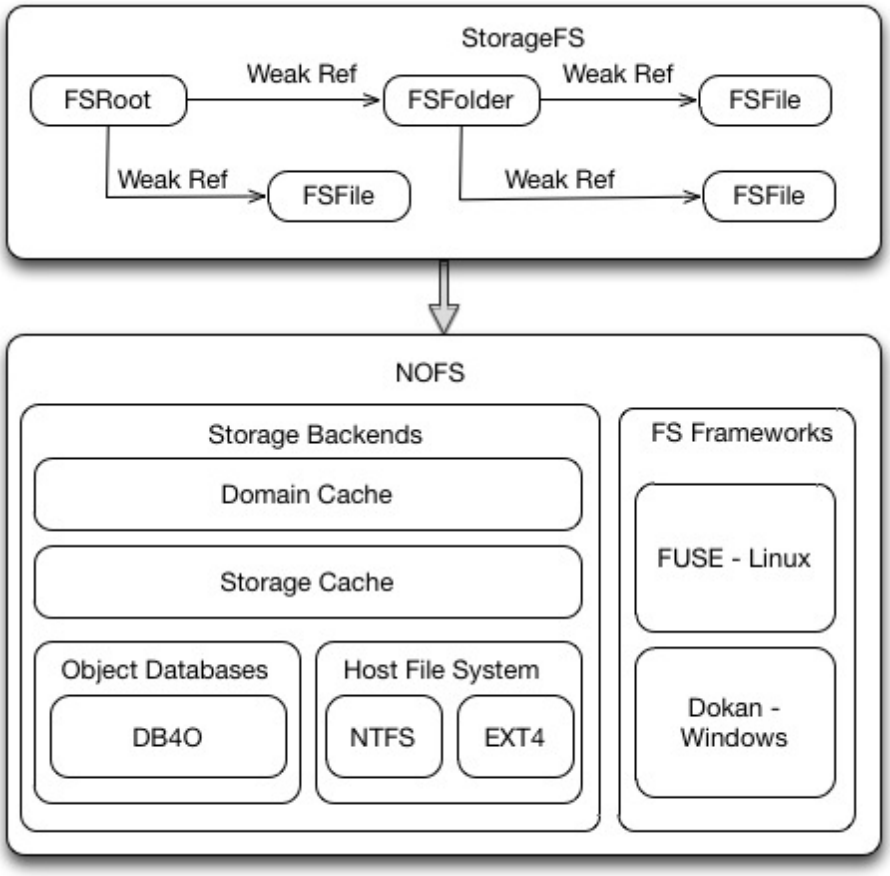
```
            string[] array = ("" + data).Split(',');
            Price = (array.Length < 2 ? "unknown" : array[1].Trim());
            Date = (array.Length < 3 ? "unknown" : array[2].Replace("\"", "").Trim());
            Time = (array.Length < 4 ? "unknown" : array[3].Replace("\"", "").Trim());
            Diff = (array.Length < 5 ? "unknown" : array[4].Trim());
        }

        [DataMember]
        public string Price { get; set; }

        [DataMember]
        public string Date { get; set; }

        [DataMember]
        public string Time { get; set; }

        [DataMember]
        public string Diff { get; set; }
    }
}
```

NOFS Architecture

- NOFS manages object persistence -DB4O and host filesystem supported -Only 400-500 lines of code needed to write a new provider
- NOFS manages connection to FUSE for Linux filesystems and Dokan for Windows filesystems
- NOFS manages object lifetime and caching

Userland Filesystems and Performance

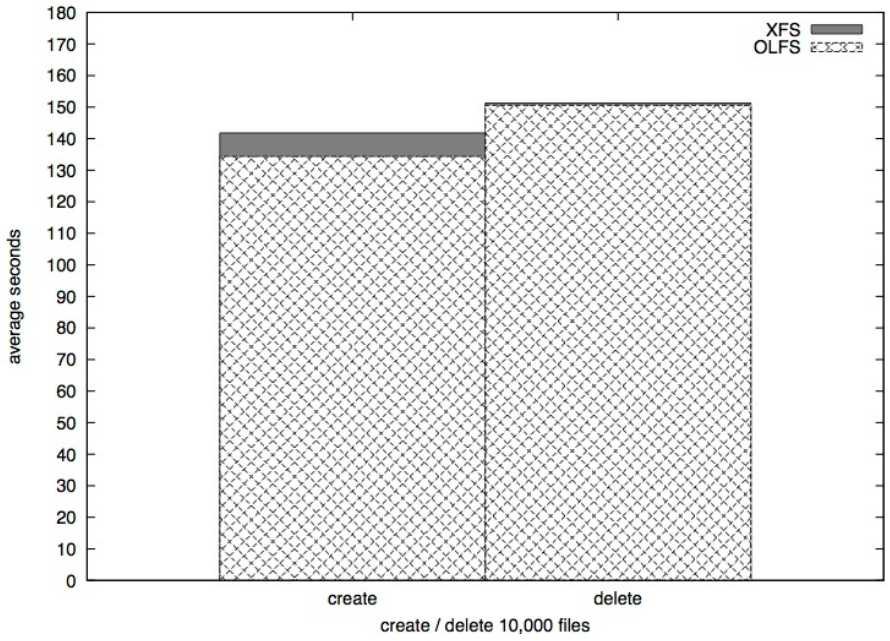
- An important concern with any user-mode operating system service is the overhead introduced by copying and context switching
- Care must be taken to return from any OS call quickly
- Care must be taken to favor larger grained operations over smaller grained operations
  - read 128K of a file 10 times instead of 8k 160 times
- User-mode filesystems make use of OS services too
  - introduce additional latency and context switching
  - again, care must be taken to favor larger grained operations and favor asynchronous operations where possible

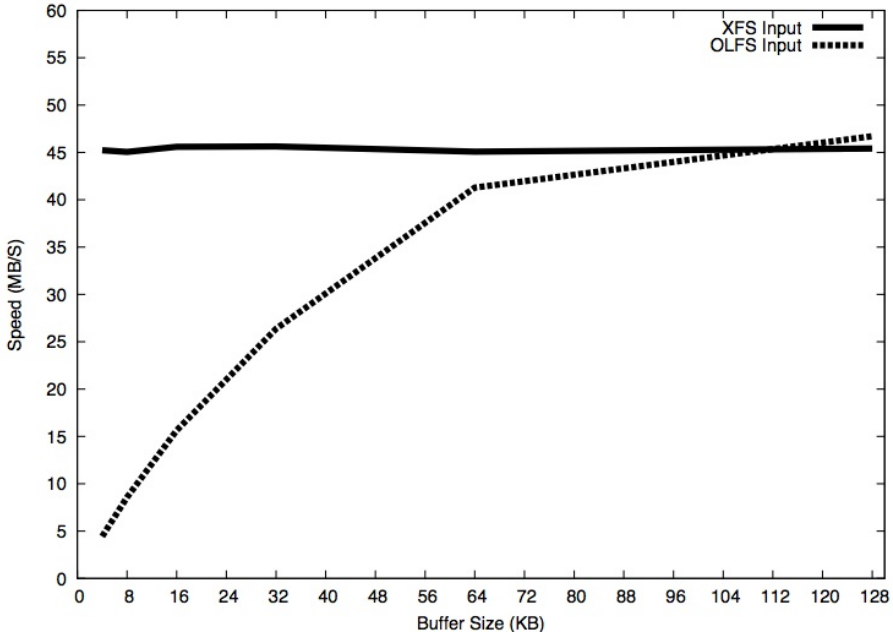

StorageFS
FSRoot
Weak Ref
FSFolder
Weak Ref
FSFile
Weak Ref
FSFile
Weak Ref
FSFile
NOFS
Storage Backends
Domain Cache
Storage Cache
Object Databases
DB4O
Host File System
NTFS
EXT4
FS Frameworks
FUSE - Linux
Dokan - Windows

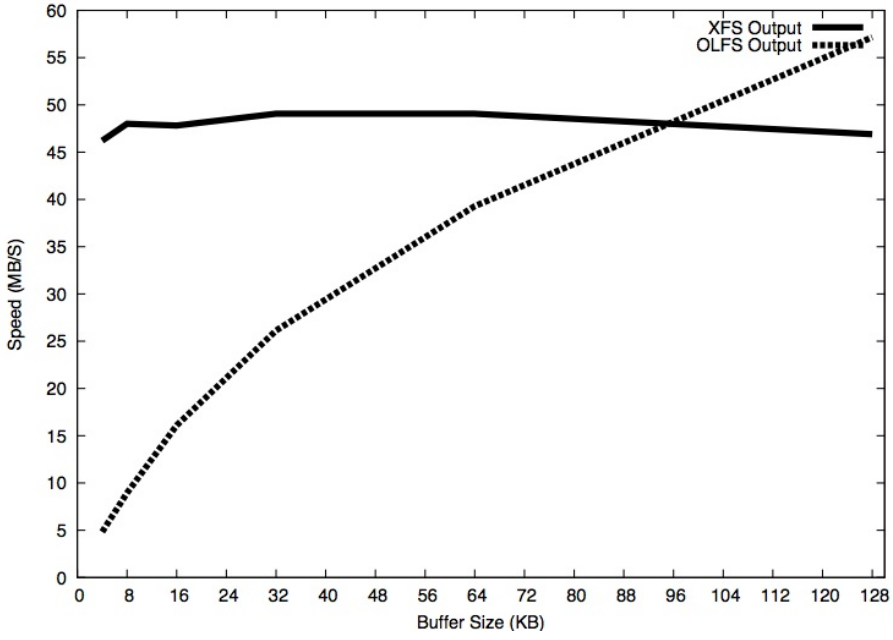
OLFS and Our History with Performance

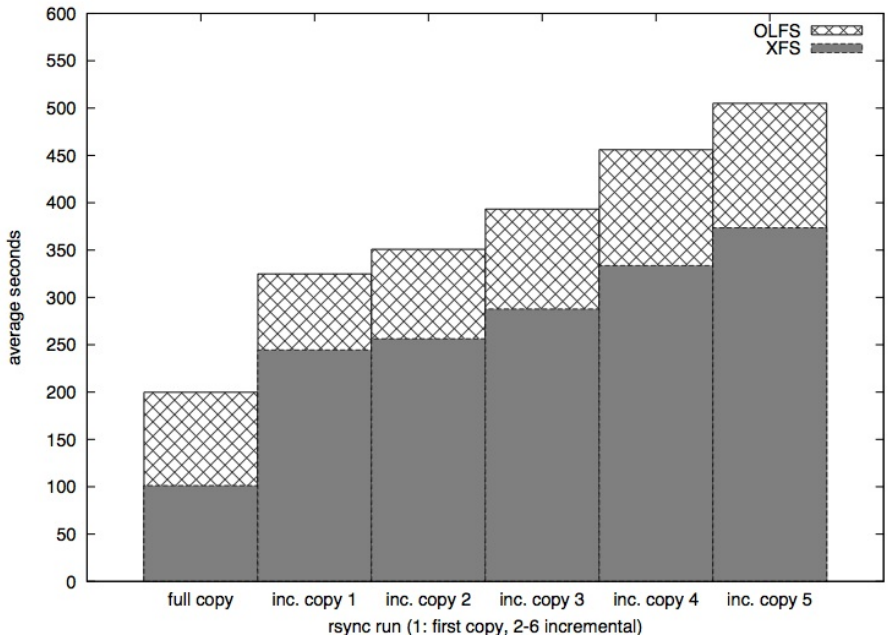
- Our first attempt at user-mode filesystems was with the OLFS project.
- The OLFS project served as a great exploration of filesystems best-practices and performance issues
- Lessons learned from the OLFS project helped to make NOFS what it is today.

OLFS vs XFS - Metadata Operations




XFS Input
OLFS Input
Speed (MB/S)
Buffer Size (KB)
0
5
10
15
20
25
30
35
40
45
50
55
60
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128


XFS Output
OLFS Output
Speed (MB/S)
Buffer Size (KB)
0
5
10
15
20
25
30
35
40
45
50
55
60
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128



OLFS vs XFS - Sequential Read Performance

OLFS vs XFS - Sequential Write Performance

OLFS vs XFS - Rsync copy of Linux Kernel Source Code Across 6 Minor Kernel Releases

OLFS - Performance - Conclusions Reached

- Realizing good performance in user-mode filesystems is possible.
- Metadata operations are the most difficult to perform well on because they represent the smallest chunk of data and require the greatest attention to locking and consistency.
- User-land caching, asynchronous operations, and using large buffers wherever possible is key.
- The use of a higher level language like Java (OLFS) versus C/C++ (XFS) was not a real contributor to the I/O performance difference between the two systems.
- The behavior of applications using the filesystem can greatly determine the differences between kernel-mode and user-mode FS performance.

- Applications that make use of smaller buffers and small reads/writes will perform poorly with user-mode filesystems.

NOFS - Performance - Current Progress

- Our latest published work concerns managing domain models that exceed the size of physical memory
- This is most important for storage filesystems and less important for application oriented filesystems
- We demonstrated how to move the caching concern into the NOFS framework and out of the domain model
- Demonstrated how to make use of the weak-reference pattern in filesystem domain models and similarities between this patterns and the i-node structure.

NOFS - Caching

NOFS - Caching - Context Switches

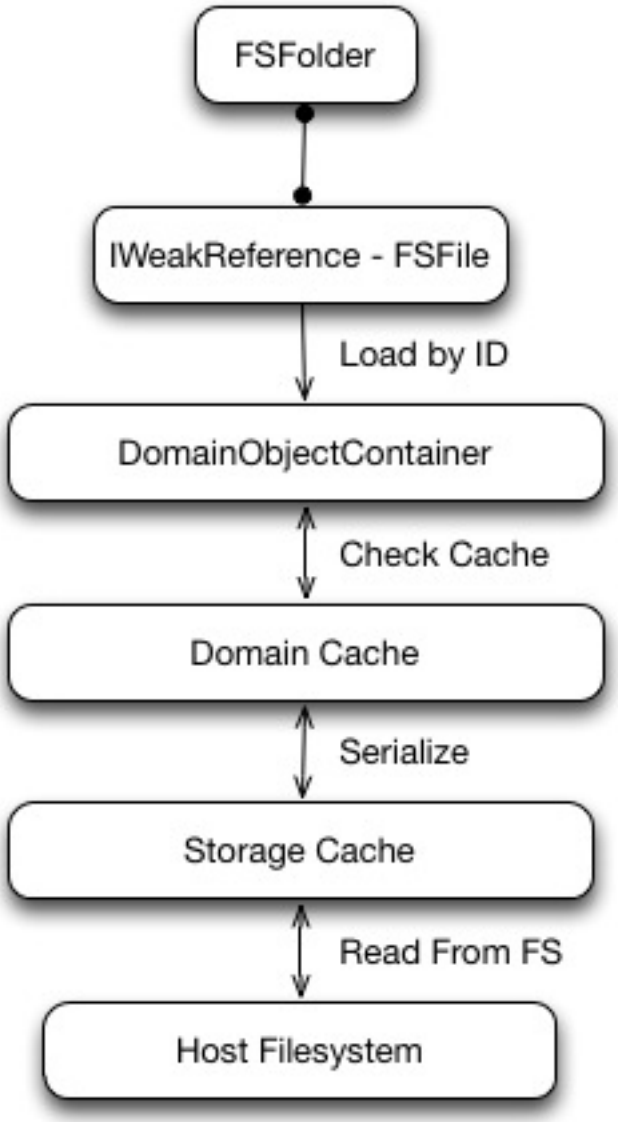
- The NOFS cache helps reduce the number of context switches in the FUSE model:

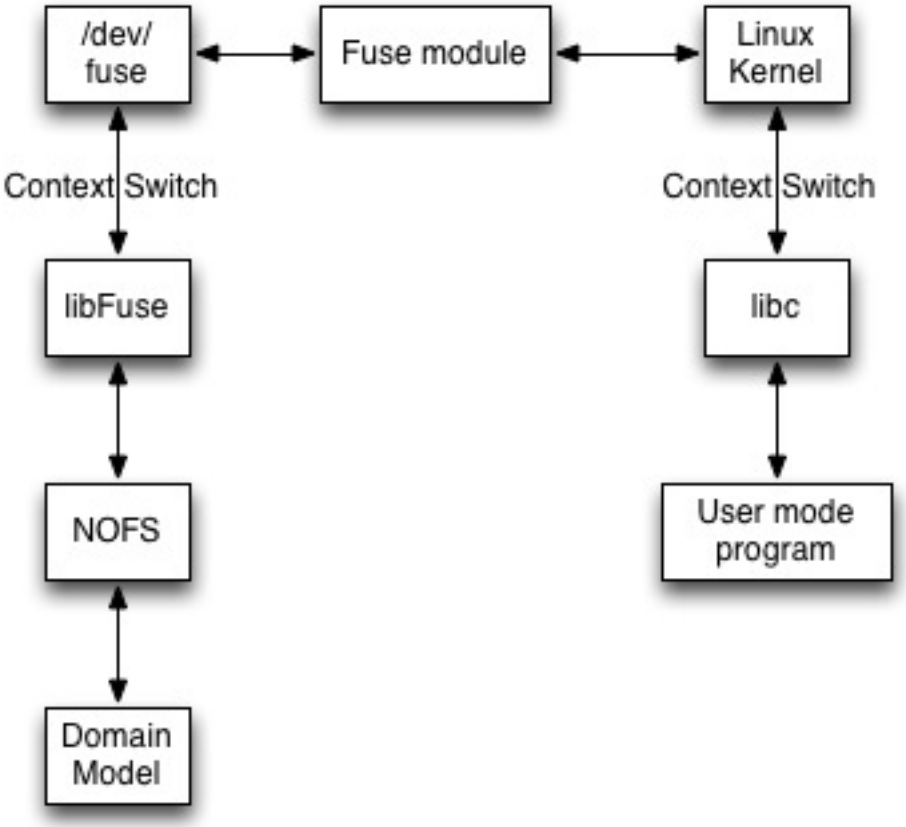
NOFS - Performance - Current Picture

- I/O Performance is acceptable, but metadata operations are not very fast.
- Currently we are working on making asynchronous metadata and I/O operations transparent to the domain model.
- With our experience with OLFS, we believe that we'll close the gap with native FS performance.

NOFS - Application Filesystems

- In addition to modeling storage based filesystems, an important aspect to NOFS is its ability to model application oriented filesystems
- These filesystems play an important role in re-exposing and filtering external services and data-sets.
- The choice of using a Naked-Objects architecture makes the construction of application oriented filesystems in NOFS much simpler than would be in FUSE.


FSFolder
IWeakReference - FSFile
Load by ID
DomainObjectContainer
Check Cache
Domain Cache
Serialize
Storage Cache
Read From FS
Host Filesystem


/dev/
fuse
Fuse module
Linux
Kernel
Context Switch
Context Switch
libFuse
libc
NOFS
User mode
program
Domain
Model

- Many application oriented filesystems can be built in NOFS with just 2-3 classes and less than 400 lines of code.
- There is very little coding overhead imposed by the NOFS framework itself.

RestFS - The Filesystem as a Connector Abstraction

- In 2010, we built RestFS, a filesystem using the NOFS framework to expose web services as filesystems.
- Our prototype implementation is able to:
  - Perform Google searches by creating files with the name containing the search terms with RestFS filling the file's contents with JSON search results
  - Perform Yahoo! Placefinder lookups
  - Connect to Twitter and list a user's Tweets
  - Connect to services using OAuth authentication

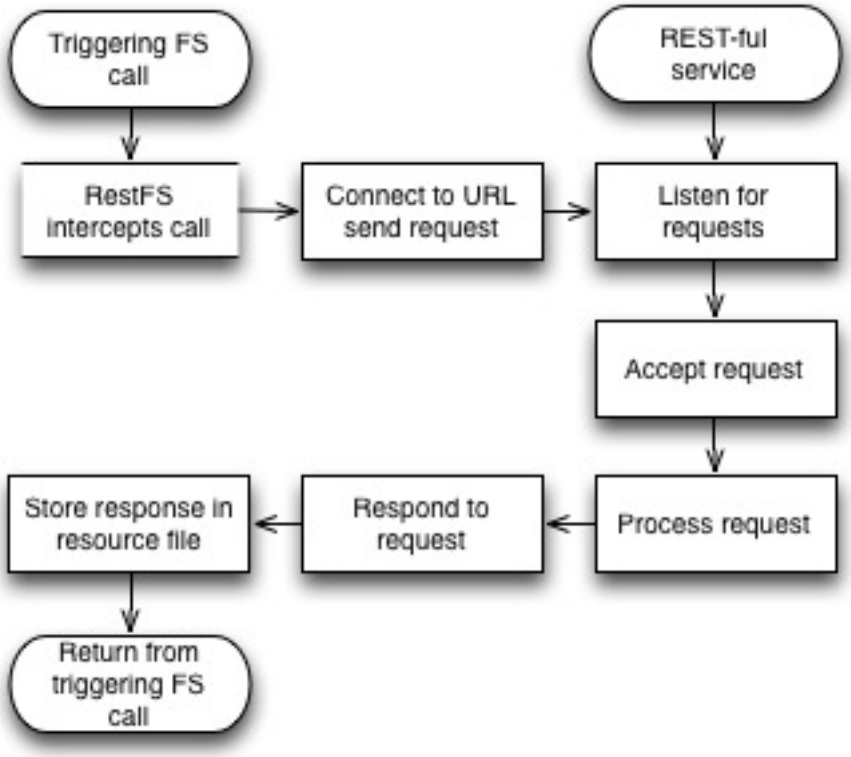
RestFS - Connecting Filesystem Calls to Web Service Methods

RestFS - Connecting Filesystem Calls to Web Service Methods

- Through the use of special configuration files, a RestFS user can map a filesystem action (create, delete, read, write, updating the timestamp), to a HTTP verb (GET, PUT, POST, DELETE) for a single file in the filesystem.
- For restful web services, it is possible to map the resources in such a service onto a filesystem and use filesystem operations to map the the HTTP verbs implemented by the restful service.
- Since restful services are modeled in some sense after FS-like calls, this is a very natural mapping in many cases

RestFS - Configuration Files - Sample

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <RestfulSetting>
3   <FsMethod>utime</FsMethod>
4   <WebMethod>get</WebMethod>
```


Triggering FS call
REST-ful service
RestFS intercepts call
Connect to URL send request
Listen for requests
Accept request
Store response in resource file
Respond to request
Process request
Return from triggering FS call

```
  <FormName></FormName>
  <Resource>ajax/services/search/web?v=1.0&amp;q=Brett%Favre
  </Resource>
  <Host>ajax.googleapis.com</Host>
  <Port>80</Port>
  <OAuthTokenPath></OAuthTokenPath>
</RestfulSetting>
```

RestFS - Software Composition

- Since the FS service is itself a very stable contract, it offers a great point at which software components can be composed.
- Specifically, we've shown several examples where RestFS could be used to compose applications and application filesystems with NOFS (multi-filesystem composition)

RestFS - Picture Album Composition

RestFS - Investment Alerts

RestFS - Blog Example

RestFS - OAuth Authentication

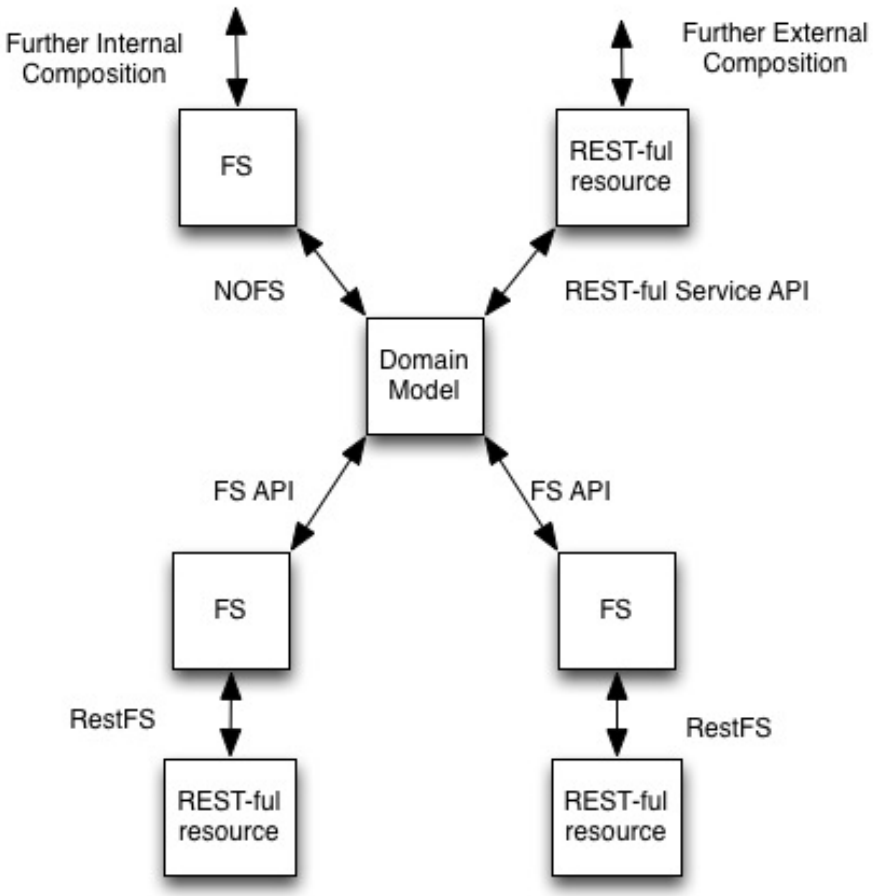
- Users of RestFS can authenticate with web services using OAuth by reading from and writing to files in a RestFS instance

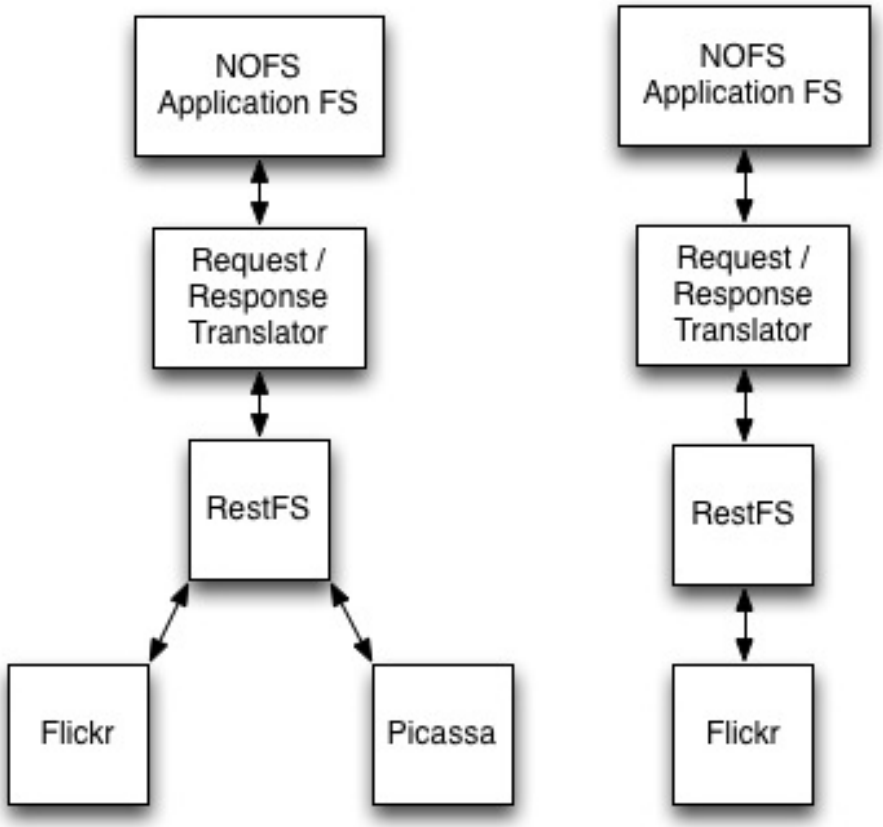
RestFS - OAuth Authentication

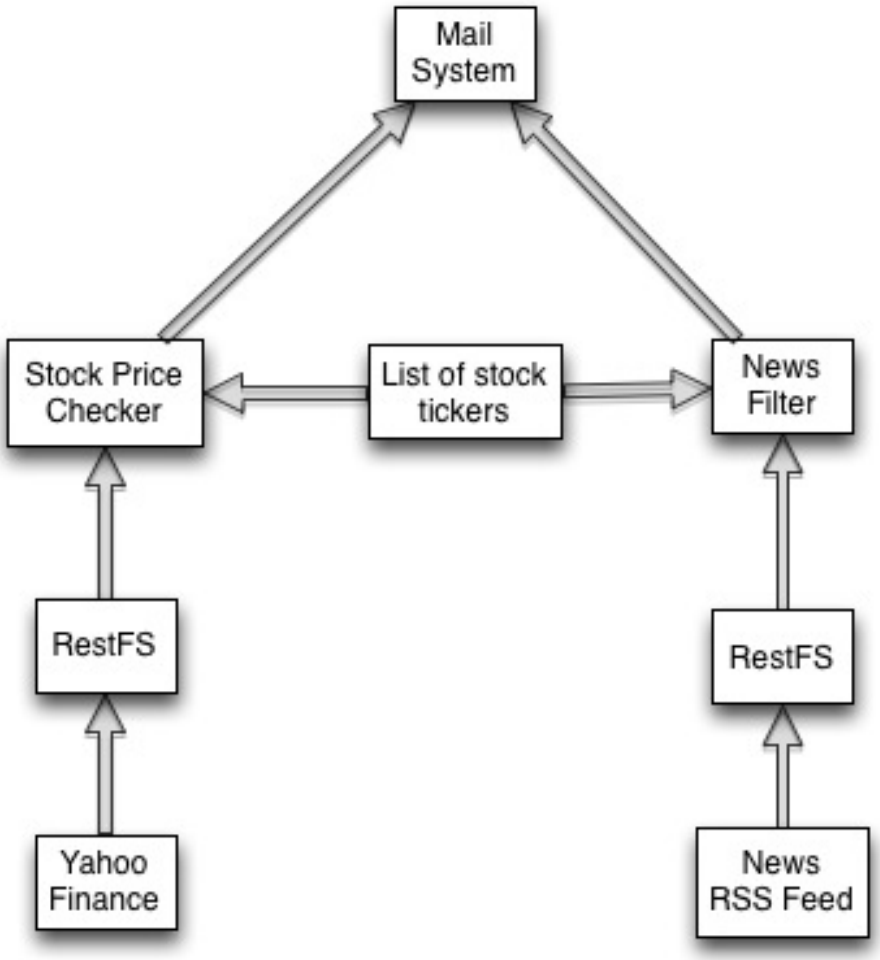
RestFS - OAuth Authentication Configuration in RestFS

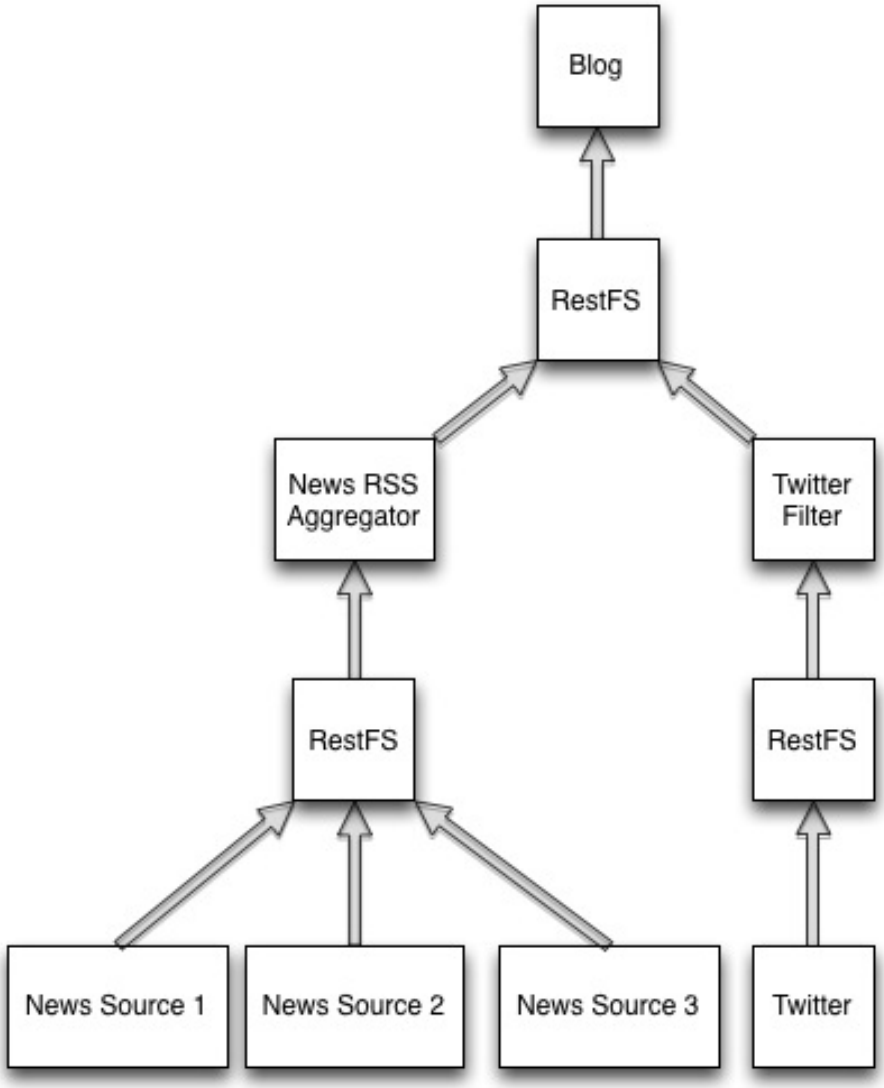
- /auth/twitter/config

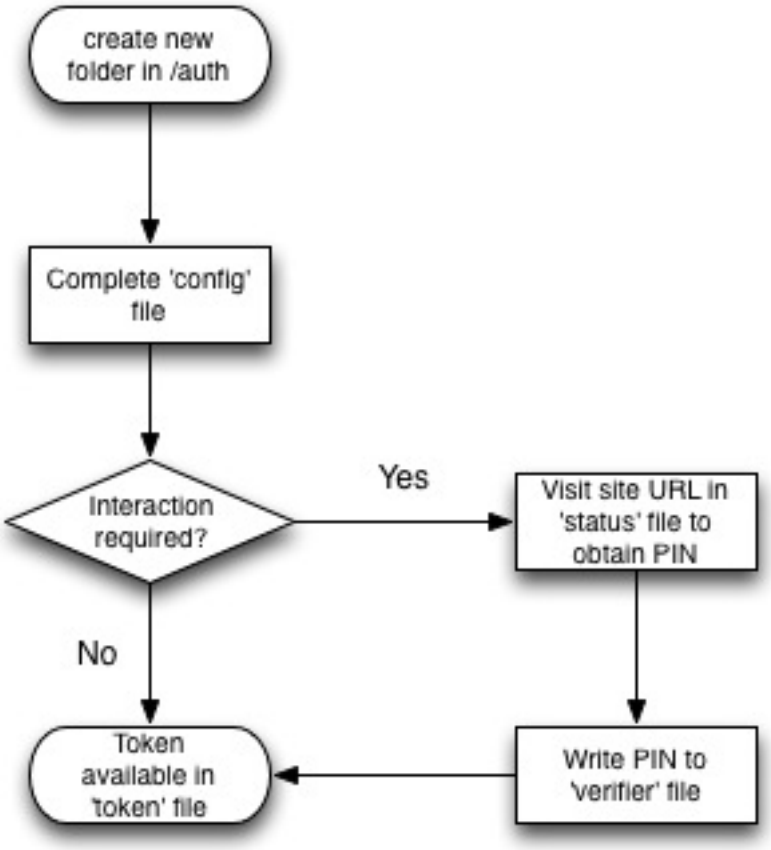
```
<?xml version="1.0" encoding="UTF-8"?>
<OAuthConfigFile>
  <Key>afhasdfkaljs34</Key>
```

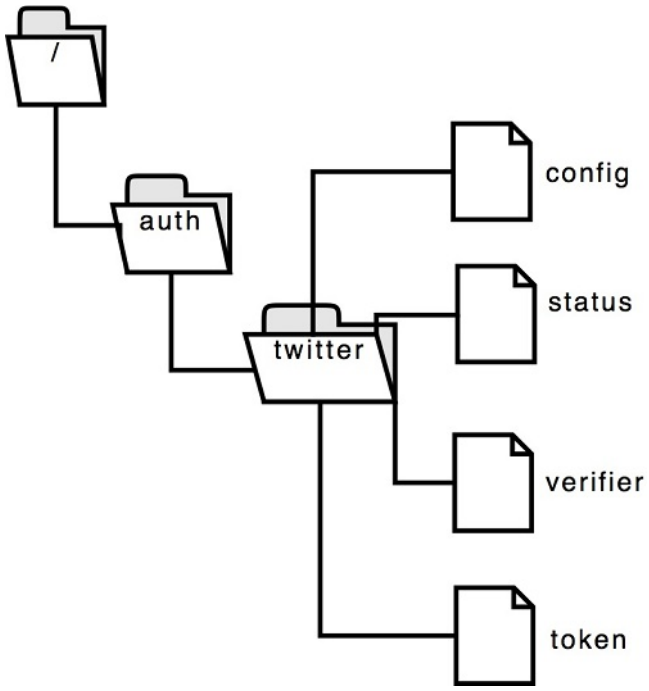

Further Internal Composition
Further External Composition
FS
REST-ful resource
NOFS
REST-ful Service API
Domain Model
FS API
FS API
FS
FS
RestFS
RestFS
REST-ful resource
REST-ful resource


NOFS
Application FS
Request /
Response
Translator
RestFS
Flickr
Picassa
NOFS
Application FS
Request /
Response
Translator
RestFS
Flickr


Mail System
Stock Price Checker
List of stock tickers
News Filter
RestFS
RestFS
Yahoo Finance
News RSS Feed


Blog
RestFS
News RSS Aggregator
Twitter Filter
RestFS
RestFS
News Source 1
News Source 2
News Source 3
Twitter


create new folder in /auth
Complete 'config' file
Interaction required?
Yes
Visit site URL in 'status' file to obtain PIN
No
Token available in 'token' file
Write PIN to 'verifier' file




```
  <AccessTokenURL>https://api.twitter.com/oauth/access_token</AccessTokenURL>
  <UserAuthURL>https://api.twitter.com/auth/authorize</UserAuthURL>
  <RequestTokenURL>https://api.twitter.com/oauth/request_token</RequestTokenURL>
  <Secret>adfhasl234</Secret>
</OAuthConfigFile>
```

- /auth/twitter/token

```
<?xml version="1.0" encoding="UTF-8"?>
<OAuthTokenFile>
  <AccessToken>2457u20283094230</AccessToken>
  <RequestToken>dfaslh34</RequestToken>
  <TokenSecret>dfho234t03hfsdf</TokenSecret>
</OAuthTokenFile>
```

Separation Between Application and Connector Filesystems

- We believe that application filesystems are best at:
  - user interaction through the FS browser

  - local composition
  - local validation
  - local data cleansing
  - making use of local software and resources
  - local file conversion - decompression / compression / filetype conversions like jpeg-png
  - exposing a single large data source as a structured filesystem
- Connector filesystems are best at:
  - providing a stable FS contract for remote web-services
  - re-exposing local software compositions as web-services
  - combining one or more web services as a single filesystem

Combining Application and Connector Filesystems

- The application filesystem should be viewed as the stable implementation
- The connector filesystem should be viewed as the unstable component
- The connector filesystem provides glue between a web service and the application filesystem
- The connector filesystem is the point of configuration
  - Adding new web services
  - Removing old web services
  - Changing mappings with web services