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

Recommended Citation


This work is licensed under a Creative Commons Attribution-Noncommercial-No Derivative Works 3.0 License.
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**
  - 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/](http://works.bepress.com/gkthiruvathukal/1/)
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.

```c
struct fuse_operations examplefs_oper;

int main(int argc, char *argv[]) {
    int i, fuse_stat;

    examplefs_oper.getattr = wrap_getattr;
    examplefs_oper.readlink = wrap_readlink;
    examplefs_oper.getdir = NULL;
    examplefs_oper.mknod = wrap_mknod;
    examplefs_oper.mkdir = wrap_mkdir;
    examplefs_oper.unlink = wrap_unlink;
    examplefs_oper.rmdir = wrap_rmdir;
    examplefs_oper.symlink = wrap_symlink;
    examplefs_oper.rename = wrap_rename;
    examplefs_oper.link = wrap_link;
    examplefs_oper.chmod = wrap_chmod;
    examplefs_oper.chown = wrap_chown;
    examplefs_oper.truncate = wrap_truncate;
    examplefs_oper.utime = wrap_utime;
    examplefs_oper.open = wrap_open;
    examplefs_oper.read = wrap_read;
    examplefs_oper.write = wrap_write;
    examplefs_oper.statfs = wrap_statfs;
    examplefs_oper.flush = wrap_flush;
    examplefs_oper.release = wrap_release;
```
examplefs_oper.fsync = wrap_fsync;
examplefs_oper.setxattr = wrap_setxattr;
examplefs_oper.getxattr = wrap_getxattr;
examplefs_oper.listxattr = wrap_listxattr;
examplefs_oper.removexattr = wrap_removexattr;
examplefs_oper.opendir = wrap_opendir;
examplefs_oper.readdir = wrap.readdir;
examplefs_oper.readdir = wrap_readdir;
examplefs_oper.releasedir = wrap_releasedir;
examplefs_oper.fsyncdir = wrap_fsyncdir;
examplefs_oper.init = wrap_init;

printf("mounting file system...\n");

for(i = 1; i < argc && (argv[i][0] == ’-’); i++) {
    if(i == argc) {
        return (-1);
    }
}

//realpath(...) returns the canonicalized absolute pathname
set_rootdir(realpath(argv[i], NULL));

for(; i < argc; i++) {
    argv[i] = argv[i+1];
} argc--;

fuse_stat = fuse_main(argc, argv, &examplefs_oper, NULL);
printf("fuse_main returned %d\n", fuse_stat);
return fuse_stat;

void set_rootdir(const char *path) {
    ExampleFS::Instance()->setRootDir(path);
}

int wrap_getattr(const char *path, struct stat *statbuf) {
    return ExampleFS::Instance()->Getattr(path, statbuf);
}

int wrap_readlink(const char *path, char *link, size_t size) {
    return ExampleFS::Instance()->Readlink(path, link, size);
}
int wrap_mknod(const char *path, mode_t mode, dev_t dev) {
    return ExampleFS::Instance()->Mknod(path, mode, dev);
}

int wrap_mkdir(const char *path, mode_t mode) {
    return ExampleFS::Instance()->Mkdir(path, mode);
}

int wrap_unlink(const char *path) {
    return ExampleFS::Instance()->Unlink(path);
}

int wrap_rmdir(const char *path) {
    return ExampleFS::Instance()->Rmdir(path);
}

int wrap_symlink(const char *path, const char *link) {
    return ExampleFS::Instance()->Symlink(path, link);
}

int wrap_rename(const char *path, const char *newpath) {
    return ExampleFS::Instance()->Rename(path, newpath);
}

int wrap_link(const char *path, const char *newpath) {
    return ExampleFS::Instance()->Link(path, newpath);
}

int wrap_chmod(const char *path, mode_t mode) {
    return ExampleFS::Instance()->Chmod(path, mode);
}

int wrap_chown(const char *path, uid_t uid, gid_t gid) {
    return ExampleFS::Instance()->Chown(path, uid, gid);
}

int wrap_truncate(const char *path, off_t newSize) {
    return ExampleFS::Instance()->Truncate(path, newSize);
}

int wrap_utime(const char *path, struct utimbuf *ubuf) {
    return ExampleFS::Instance()->Utime(path, ubuf);
}

int wrap_open(const char *path, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Open(path, fileInfo);
}

int wrap_read(const char *path, char *buf, size_t size, off_t offset, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Read(path, buf, size, offset, fileInfo);
}

int wrap_write(const char *path, const char *buf, size_t size, off_t offset, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Write(path, buf, size, offset, fileInfo);
}

int wrap_statfs(const char *path, struct statvfs *statInfo) {
    return ExampleFS::Instance()->Statfs(path, statInfo);
}
int wrap_flush(const char *path, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Flush(path, fileInfo);
}

int wrap_release(const char *path, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Release(path, fileInfo);
}

int wrap_fsync(const char *path, int datasync, struct fuse_file_info *fi) {
    return ExampleFS::Instance()->Fsync(path, datasync, fi);
}

int wrap_setxattr(const char *path, const char *name, const char *value, size_t size, int flags) {
    return ExampleFS::Instance()->Setxattr(path, name, value, size, flags);
}

int wrap_getxattr(const char *path, const char *name, char *value, size_t size) {
    return ExampleFS::Instance()->Getxattr(path, name, value, size);
}

int wrap_listxattr(const char *path, char *list, size_t size) {
    return ExampleFS::Instance()->Listxattr(path, list, size);
}

int wrap_removexattr(const char *path, const char *name) {
    return ExampleFS::Instance()->Removexattr(path, name);
}

int wrap_opendir(const char *path, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Opendir(path, fileInfo);
}

int wrap_readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Readdir(path, buf, filler, offset, fileInfo);
}

int wrap_releasedir(const char *path, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Releasedir(path, fileInfo);
}

int wrap_fsyncdir(const char *path, int datasync, struct fuse_file_info *fileInfo) {
    return ExampleFS::Instance()->Fsyncdir(path, datasync, fileInfo);
}

int wrap_init(struct fuse_conn_info *conn) {
    return ExampleFS::Instance()->Init(conn);
}

ExampleFS* ExampleFS::_instance = NULL;

#define RETURN_ERRNO(x) (x) == 0 ? 0 : -errno

ExampleFS* ExampleFS::Instance() {
    if (_instance == NULL) {
        _instance = new ExampleFS();
    }
    return _instance;
ExampleFS::ExampleFS() {
}
ExampleFS::~ExampleFS() {
}
void ExampleFS::AbsPath(char dest[PATH_MAX], const char *path) {
    strcpy(dest, _root);
    strncat(dest, path, PATH_MAX);
    //printf("translated path: %s to %s\n", path, dest);
}
void ExampleFS::setRootDir(const char *path) {
    printf("setting FS root to: %s\n", path);
    _root = path;
}
int ExampleFS::Getattr(const char *path, struct stat *statbuf) {
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    printf("getattr(%s)\n", fullPath);
    return RETURN_ERRNO(lstat(fullPath, statbuf));
}
int ExampleFS::Readlink(const char *path, char *link, size_t size) {
    printf("readlink(path=%s, link=%s, size=%d)\n", path, link, (int)size);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(readlink(fullPath, link, size));
}
int ExampleFS::Mknod(const char *path, mode_t mode, dev_t dev) {
    printf("mknod(path=%s, mode=%d)\n", path, mode);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    //handles creating FIFOs, regular files, etc...
    return RETURN_ERRNO(mknod(fullPath, mode, dev));
}
int ExampleFS::Mkdir(const char *path, mode_t mode) {
    printf("*mkdir(path=%s, mode=%d)\n", path, (int)mode);
```c
char fullPath[PATH_MAX];
AbsPath(fullPath, path);
return RETURN_ERRNO(mkdir(fullPath, mode));
}

int ExampleFS::Unlink(const char *path) {
    printf("unlink(path=%s\n)", path);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(unlink(fullPath));
}

int ExampleFS::Rmdir(const char *path) {
    printf("rmdir(path=%s\n)", path);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(rmdir(fullPath));
}

int ExampleFS::Symlink(const char *path, const char *link) {
    printf("symlink(path=%s, link=%s)\n", path, link);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(symlink(fullPath, link));
}

int ExampleFS::Rename(const char *path, const char *newpath) {
    printf("rename(path=%s, newPath=%s)\n", path, newpath);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(rename(fullPath, newpath));
}

int ExampleFS::Link(const char *path, const char *newpath) {
    printf("link(path=%s, newPath=%s)\n", path, newpath);
    char fullPath[PATH_MAX];
    char fullNewPath[PATH_MAX];
    AbsPath(fullPath, path);
    AbsPath(fullNewPath, newpath);
    return RETURN_ERRNO(link(fullPath, fullNewPath));
}

int ExampleFS::Chmod(const char *path, mode_t mode) {
    printf("chmod(path=%s, mode=%d)\n", path, mode);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(chmod(fullPath, mode));
}
```
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_info *fileInfo) {
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 fuse_file_info *fileInfo) {
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);
return RETURN_ERRNO(statvfs(fullPath, statInfo));

int ExampleFS::Flush(const char *path, struct fuse_file_info *fileInfo) {
    printf("flush(path=%s)\n", path);
    //noop because we don't maintain our own buffers
    return 0;
}

int ExampleFS::Release(const char *path, struct fuse_file_info *fileInfo) {
    printf("release(path=%s)\n", path);
    return 0;
}

int ExampleFS::Fsync(const char *path, int datasync, struct fuse_file_info *fi) {
    printf("fsync(path=%s, datasync=%d\n", path, datasync);
    if(datasync) {
        //sync data only
        return RETURN_ERRNO(fdatasync(fi->fh));
    } else {
        //sync data + file metadata
        return RETURN_ERRNO(fsync(fi->fh));
    }
}

int ExampleFS::Setxattr(const char *path, const char *name, const char *value, size_t size, int flags) {
    printf("setxattr(path=%s, name=%s, value=%s, size=%d, flags=%d\n",
            path, name, value, (int)size, flags);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(lsetxattr(fullPath, name, value, size, flags));
}

int ExampleFS::Getxattr(const char *path, const char *name, char *value, size_t size) {
    printf("getxattr(path=%s, name=%s, size=%d\n", path, name, (int)size);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(getxattr(fullPath, name, value, size));
}

int ExampleFS::Listxattr(const char *path, char *list, size_t size) {
    printf("listxattr(path=%s, size=%d)\n", path, (int)size);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(llistxattr(fullPath, list, size));
}
int ExampleFS::Removexattr(const char *path, const char *name) {
    printf("removexattr(path=%s, name=%s)\n", path, name);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    return RETURN_ERRNO(lremovexattr(fullPath, name));
}

int ExampleFS::Opendir(const char *path, struct fuse_file_info *fileInfo) {
    printf("opendir(path=%s)\n", path);
    char fullPath[PATH_MAX];
    AbsPath(fullPath, path);
    DIR *dir = opendir(fullPath);
    fileInfo->fh = (uint64_t)dir;
    return NULL == dir ? -errno : 0;
}

int ExampleFS::Readdir(const char *path, void *buf, fuse_fill_dir_t filler, off_t offset, struct fuse_file_info *fileInfo) {
    printf("readdir(path=%s, offset=%d)\n", path, (int)offset);
    DIR *dir = (DIR*)fileInfo->fh;
    struct dirent *de = readdir(dir);
    if(NULL == de) {
        return -errno;
    } else {
        do {
            if(filler(buf, de->d_name, NULL, 0) != 0) {
                return -ENOMEM;
            }
        } while(NULL != (de = readdir(dir)));
    }
    return 0;
}

int ExampleFS::Releasedir(const char *path, struct fuse_file_info *fileInfo) {
    printf("releasedir(path=%s)\n", path);
    closedir((DIR*)fileInfo->fh);
    return 0;
}

int ExampleFS::Fsyncdir(const char *path, int datasync, struct fuse_file_info *fileInfo) {
    return 0;
}

int ExampleFS::Init(struct fuse_conn_info *conn) {
    return 0;
}
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.
protected FsFolderOrFile()
{
    Id = Guid.NewGuid().ToString();
}

protected FsFolderOrFile(string name, string id)
{
    _name = name;
    Id = id;
}

[ProvidesName]
public string Name {
    get { return _name; }
    set {
        _name = value;
        if (Container != null) {
            Container.ObjectChanged(this);
        }
    }
}

[FolderObject(ChildTypeFilterMethod = "Filter")]
[ProvidesMappingDetails(typeof(FsFolderMapper))]
public class FsFolder : FsFolderOrFile, IWeakReferenceList<FsFolderOrFile>
{
    private readonly WeakReferenceList<FsFolderOrFile> _list;

    public FsFolder() : base()
    {
        _list = new WeakReferenceList<FsFolderOrFile>(this);
    }

    public FsFolder(string name, string id) : base(name, id)
    {
        _list = new WeakReferenceList<FsFolderOrFile>(this);
    }

    public bool Filter(Type possibleChildtype)
    {
        return
        possibleChildtype == typeof(FsFolder) ||
        possibleChildtype == typeof(FsFile);
    }

    [NeedsContainerManager]
    public void SetContainer(IDomainObjectContainerManager container)
    {
        _list.setContainer(container);
    }
}
public IEnumerator<IWeakReference> GetEnumerator(){
    return _list.GetEnumerator();
}
IEnumerator IEnumerable.GetEnumerator(){
    return GetEnumerator();
}
public int Count{
    get { return _list.Count; }
}
public void Add(IWeakReference item){
    _list.Add(item);
}
public void Add(object item){
    _list.Add(item);
}
public void Remove(IWeakReference item){
    _list.Remove(item);
}
public void Remove(object item){
    _list.Remove(item);
}
public void Add(FsFolderOrFile item){
    _list.Add(item);
}
public void Remove(FsFolderOrFile item){
    _list.Remove(item);
}
public IEnumerator<FsFolderOrFile> GetAll(){
    return _list.GetAll();
}

[ProvidesMappingDetails(typeof(FsFileMapper))]
public class FsFile : FsFolderOrFile, IProvidesUnstructuredData
{
    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.Id);  
    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);  
  }  
}
public sealed class FsFolderMapper : IProvidesMappingDetails
{
    private IDomainObjectContainerManager _manager;

    [NeedsContainerManager]
    public void SetContainerManager(IDomainObjectContainerManager manager)
    {
        _manager = manager;
    }

    public string GetXMLRepresentation(object obj)
    {
        var folder = (FsFolder) obj;
        StringBuilder xml = new StringBuilder();
        xml.AppendFormat("<Folder name="{0}" id="{1}" root="{2}" >", 
            folder.Name, folder.Id, (folder is FsRoot));
        foreach (var item in folder)
        {
            xml.AppendFormat("<Child id="{0}" type="{1}" ></Child>", item.Id, item.UnderlyingType.Name);
        }
        xml.Append("</Folder>");
        var xmlText = xml.ToString();
        return xmlText;
    }

    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;
        var isRoot = bool.Parse(root.Attributes["root"].Value);
        FsFolder folder;
        if (isRoot)
        {
            folder = new FsRoot(name, id);
        }
        else
        {
            folder = new FsFolder(name, id);
        }

        foreach (var childNode in root.ChildNodes.OfType<XmlElement>()){ 
            id = childNode.Attributes["id"].Value;
            var typeName = childNode.Attributes["type"].Value;
            IDomainObjectContainer container;
            if (typeName == typeof(FsFile).Name)
            {
                container = _manager.GetContainer(typeof(FsFile));
            }
            else if (typeName == typeof(FsFolder).Name)
            {
                container = _manager.GetContainer(typeof(FsFolder));
            }
            else{
                }}}}
throw new Exception("unknown child type name: " + typeName);
}

var weakRef = container.GetWeakReference(id);
folder.Add(weakRef);

return folder;
}

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**

![Flowchart diagram](image)

**A Simple Application Filesystem in NOFS**

```csharp
using System;
using System.Runtime.Serialization;
using System.Reflection;
using System.Net;
using System.IO;
using System.Text;
using nofs.attributes;
using nofs.application;
using nofs.containers.interfaces;

namespace stocks
{
    class Program
    {
        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");
}

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);
    }
}
private void UpdateStockData() {
    String url = BuildURL();
    List<String> dataLines = getDataFromURL(url);
    foreach (Stock stock in _stocks) {
        String dataLine = null;
        foreach (String line in dataLines) {
            if (line.StartsWith("" + stock.Ticker)) {
                dataLine = line;
                break;
            }
        }
        if (dataLine != null) {
            stock.UpdateData(dataLine);
        }
    }
}

private String BuildURL() {
    List<string> tickers = new List<string>();
    foreach (Stock stock in _stocks) {
        tickers.Add(stock.Ticker);
    }
    StringBuilder url = new StringBuilder();
    url.Append(string.Join(",", tickers.ToArray()));
    url.Append("&f=sl1d1t1c1ohgv&e=.csv")
    return url.ToString();
}

private List<String> getDataFromURL(String url) {
    Uri uri = new Uri(url);
    HttpWebRequest request = (HttpWebRequest)HttpWebRequest.Create(uri);
    request.AutomaticDecompression = DecompressionMethods.GZip | DecompressionMethods.Deflate;
    request.KeepAlive = true;
    request.Timeout = 10 * 60 * 1000;
    byte[] array = null;

    using (HttpResponseMessage response = (HttpResponseMessage)request.GetResponse()) {
        using (Stream os = response.GetResponseStream()) {
            using (MemoryStream ms = new MemoryStream()) {
                const int bufferLength = 1024;
                byte[] buf = new byte[bufferLength];
                int read = 0;
                while ((read = os.Read(buf, 0, bufferLength)) > 0) {
                    ms.Write(buf, 0, read);
                }
                return ms.ToArray();
            }
        }
    }
}
```csharp
ms.WriteByte(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("\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
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
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.
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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<RestfulSetting>
  <FsMethod>utime</FsMethod>
  <WebMethod>get</WebMethod>
</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
<?xml version="1.0" encoding="UTF-8"?>
<OAuthConfigFile>
  <Key>afhasdfkaljs34</Key>
</OAuthConfigFile>
```
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
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