# Zgae: Proposed file system architecture

*Laboratorio de Sistemas*

*ABSTRACT*

This paper is a description of the present ideas for building a new distributed file system at the Laboratorio de Sistemas. The design described here is subject to change on a daily basis but the main ideas will probably remain the same.

## Problem statement

The problem we are trying to address is distributed file access. In principle there are many distributed file systems that can be used. In practice no one of them works well for different usage scenarios, because scenarios may vary between disconnected operation and be sitting on top of our main file server.

We are copying files by hand despite Plan 9 and the Octopus, on certain occasions. This means that we require a new global file system providing file access so that no file has to be **ever** copied by hand again.

## Plan for Zgae

The overview for the architecture we have in mind is depicted in figure 1. This architecture has one main purpose:

> **Instead of using a single protocol that works well in all cases, let's integrate different protocols so that different usage scenarios may have the best protocol known while still be part of a single file system.**

We are considering only three different usage scenarios, although there may be other ones:

1. Local usage. Similar to using a Plan 9 file server from within a local area network. A strong coherency model may be supplied by the protocol.
2. Remote usage. Crossing links typical of ADSL lines. Here we have in mind using a protocol similar to Nop (Octopus, next generation). It is a protocol based on leases, already implemented but not yet operational.
3. Overseas usage. A very weak coherency model has to be employed for usability. Perhaps even disconnected operation.

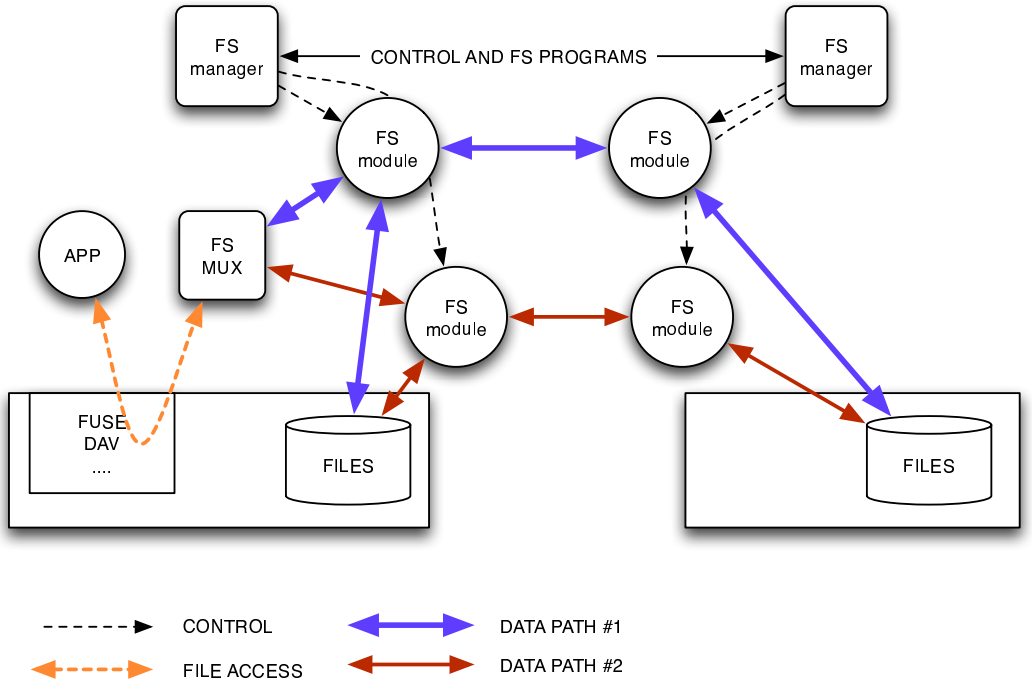


**Figure 1: The proposed architecture for the Zgae file system. All the components are dedicated to put in place different file system modules. Each FS module implements a particular distributed file system protocol intended for particular network conditions. To do so it relies on local storage and on its own FS protocol spoken with other peers.**

### Zgae architecture

The architecture we are thinking on is depicted on figure 1. Its purpose is to keep a set of, so called, **file volumes** accessible for the user. A file volume is a file tree along with a global user-given name. For example, *Teddy's music*.

To make the tree highly available it will be replicated on many occasions (depending on what Zgae decides to do with the volume). We call each of these replicas a **volume replica**. Therefore volume replicas are mostly file trees.

The top-level component of the architecture is the **file system manager**, or FS manager for short. It is a component of the file system that makes choices regarding which actual protocol should be used to reach a volume (one of its replicas) and how to do it. Its main purpose is to instantiate the component described next.

A **file system module**, or FS module for short, is an actual implementation for a particular distributed file system protocol. For example, one module could implement Nop; another could implement disconnected operation; yet another could simply speak 9P over the network.

The file interface as seen by the client (the host OS at any terminal machine, or an application) is that of UNIX. The **file system multiplexor** module, or FS mux for short, provides this interface and delegates *all* of its implementation to a particular FS module.

**Protocols**

The interface provided by the FS mux must be available to native applications in the host OS. This means that the implementation must include either a FUSE module, or a WebDAV server, or any other protocol implementation known to the host system. This is what we call the **file access protocol**. A direct protocol (e.g., based on FUSE) is preferred although others may be simpler to implement and maintain.

The protocol spoken between FS managers is responsible for keeping the system integrated. However, it does **not** include anything related to how to transfer files between parties involved. This would prevent FS modules from being able to use whatever protocol they might prefer.

In general, the messages exchanged between managers are used to report known volumes (and replicas), the FS modules available for them, performing global (per-volume) authorization, etc.

The protocol spoken between two FS modules is defined by the particular FS modules considered. Therefore this is outside of the architecture. As we said before we are considering three alternative modules: 9p, Nop, and one supported disconnected operation.

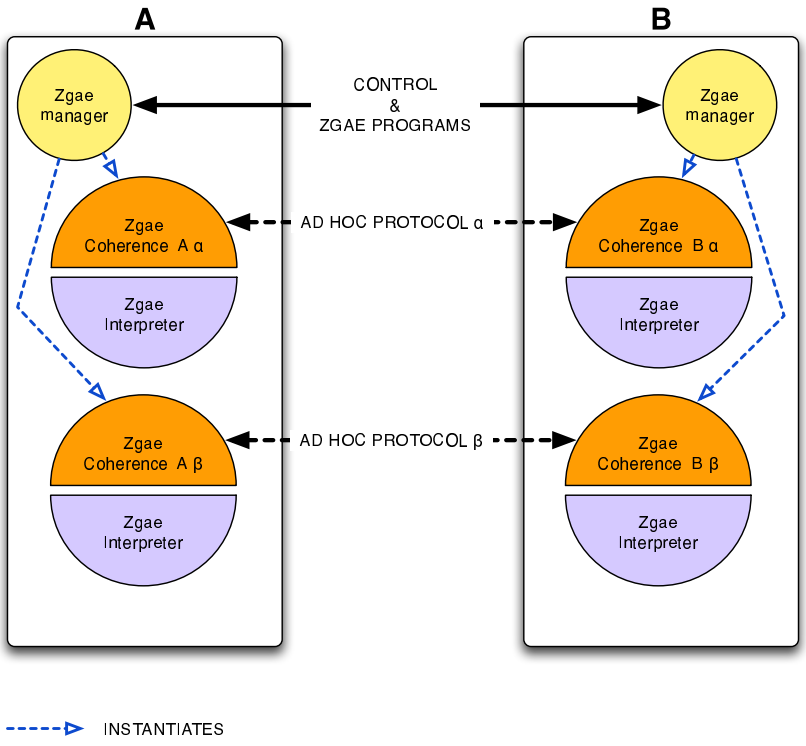


**Figure 2: Interpreted FS modules. The implementation for different FS modules can be an interpreted program evaluated by a single interpreter. This may coexist with native implementations.**

**Deployment**

We think that the implementation for all the FS modules is likely to be quite similar. In general, they will implement the FS access protocol (that exported by the FS mux) by relying on the local storage and a protocol spoken with a peer module.

Our plan is to define and implement a language to specify these modules. See figure 2. By doing so, we believe that a single implementation for the FS module may be enough to support all different modules. This would reduce the number of implementations required to one per host OS (Windows, Linux, Plan 9, MacOS X) and may provide the benefit of a hosted implementation with the speed of a native one.

Using an interpreted FS module is highly desirable because this permits FS managers to deploy new coherency models (new FS modules) at run time, without disruption of existing services.

The local storage used as a cache or replica of a volume kept locally must provide a well-known API so that different FS modules could operate on the same local copy. We are considering an abstract API based on UNIX. Other per-module local persistent storage may be used to keep module specific data (e.g., replication data bases or logs).

Although there are some early ideas regarding this interpreted FS module implementation, they are probably best discussed separately because it does not affect the overall architecture (ignoring the changes in the manager protocol required to download the interpreted code for new FS modules).