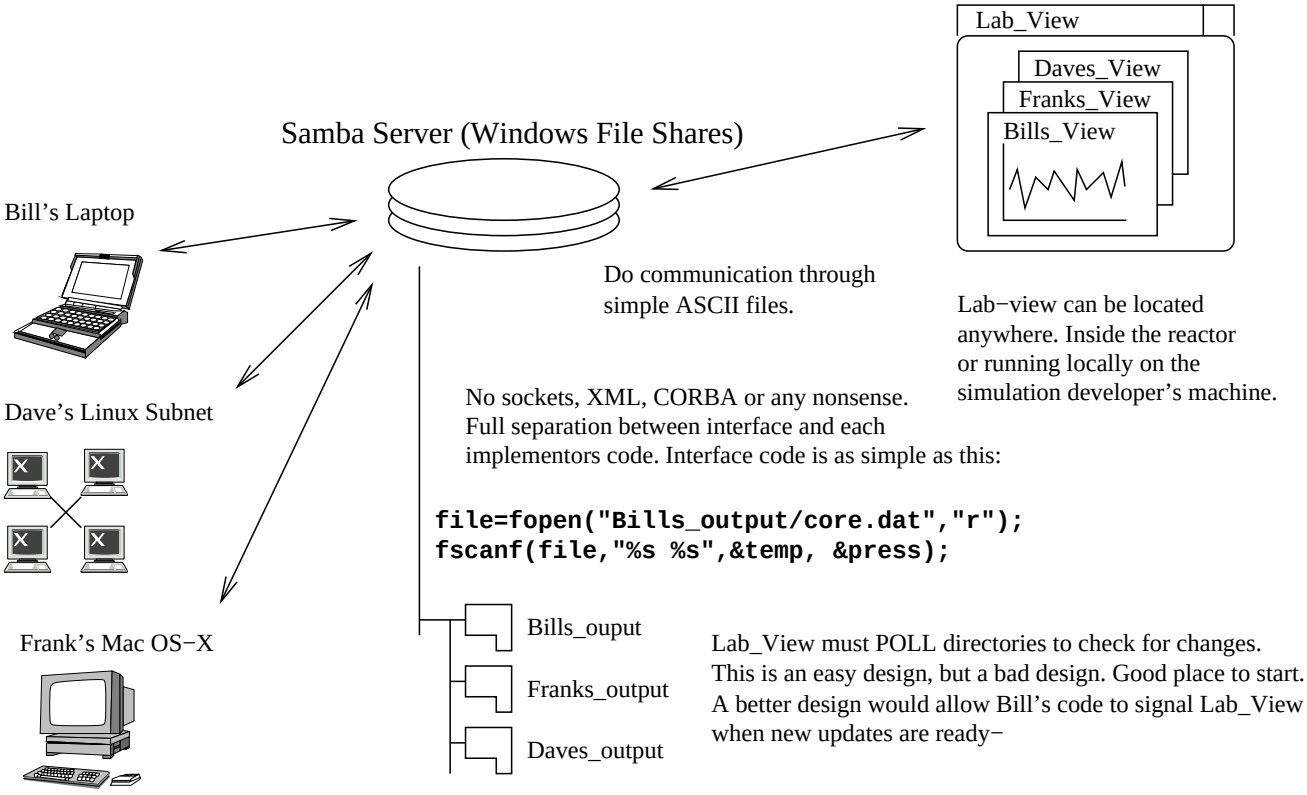
Hi Bill,

I had a few thoughts about the LabWindows tool you showed to me. Here's a picture that shows Labview clients as a unifying user interface communicating with graduate student codes via a file server. This is in line with a lot of the things you've said about blackboards ect., but I think you'd do yourself a favor by sticking to more familiar implementation and terminology of a file server. Choosing a simple and robust founding design is important if you want the research project to fly, if it was up to me, this is how I would do it.



Reading and writing little ASCII files as a communication mechanism between machines which share a common communication directory is very simple to implement. I'd recommend sending even floating point numbers between codes by converting them to ASCII in one program and saving them in a file to be read by another. Although its a little slower this way it solves a lot of binary compatibility problems that machines tend to have, AND it makes debugging a breeze. When communications fail- you can just use a text editor to examine the files in the communications directories. If its a windows NT file server I think you and your students could work with it easily.

I think that you as the project director need to keep a tight watch over exactly how the LabView window display code communicates with the file server. You could define a basic standard for your students to follow (something simple, like use ASCII files, use such and such a naming convention for communication directories etc.), and the LabView client would be a small enough piece of

code that you wouldn't be overburdened by its maintenance.

You really don't want to be locked into a situation where you are maintaining the root node in a giant code tree of every program you and your students ever wrote. Trying to keep something like that functional and coherent is what killed Dr. Parnas's research software. Students just supply you with a small labView stub which implements their display, then they can go away and work on their research modeling software which runs on a separate computer. You will probably need to do a sanity check on your student's display code, but this shouldn't be too much work if all the functionality is in a separate program, and labview code is only for display.

A person who is developing software could simply run the lab view client on their local machine, and read and write files to a local directory while they tested their code, in isolation from the main project.

Decoupling an interface from student implementations will give you a robust system. When one student model hangs, it won't effect the others. Since information is communicated via a file server, you can have multiple clients which all see the same picture. Since your labview client offers several views of a problem, it won't be busy churning away on all of them at the same time.

This kind of thing is necessary for my work. I'd like to run a simulation code which perhaps generates a new graphic representing the core state every 15 minutes. I can save my output as a .gif file, and your labview client will only bother to load that .gif file if my view is being displayed.

You could also put say 3 labview terminals in the reactor room, which only display the well tested modules. Say 7 views are available, reactor workers choose which view they want to see. Perhaps a configuration flag hides the experimental views. Students develop code until you think its ready, then you mark their window as tested, and it becomes available for display in the reactor. Your whole research group on the other hand gets to see all the views, tested and untested, you get better interaction and feedback between your students.

The problem with this implementation is that it requires regular polling, or checking of the communications directory. This is like the global variables of networking, its considered very bad because it can create horrible headaches in certain situations, but its easy to implement, and I think for what you want it can be made to work. It works best with display only type clients, but I think this is a good starting point which we can fix and augment as time goes on. Signals, lockfiles, semaphores, all these things can be added later if needed.

I'm just throwing in my 2 cents here. I think getting us started off on the right foot is important, and I think an idea like using a file server with ASCII files as a communication device is simple and general enough to satisfy just about anybody. Whatever you want me to do for the RA job is up to you though, these are only suggestions.

Dave