Project Report
Group Tortoise

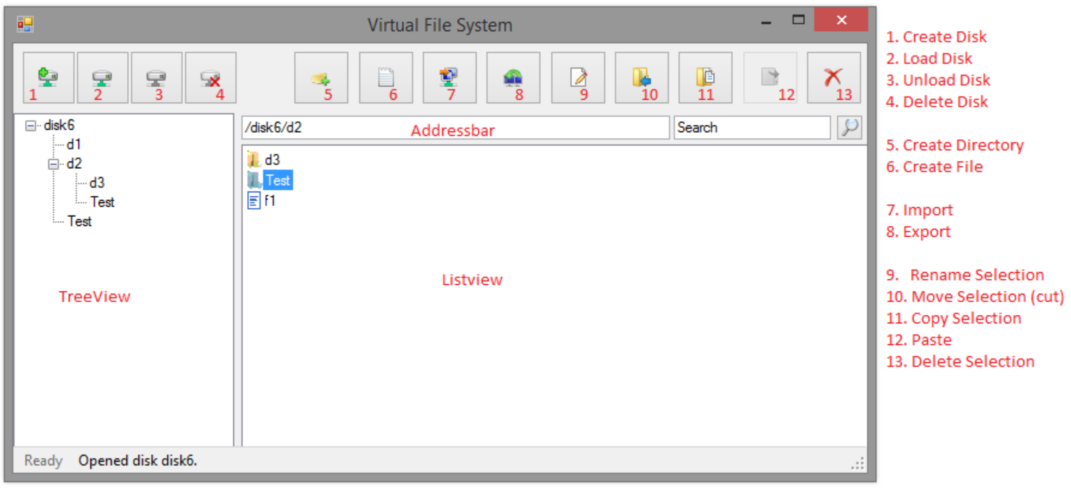
Java and C# in depth, Spring 2014

Lukas Häfliger
Fabian Meier
Andrea Canonica

April 28, 2014

1 VFS Browser

Our Browser is a Desktop Application. It provides a nice GUI that resembles a bit the Windows Explorer and therefore should help to easily use it. We decided to use buttons to represent the core requirements of part one and tried to use self-explaining icons for each of those functions. But just in case, hovering over a button will display a little text that explains its function.



Ready Opened disk disk6.

- A click on the *Create Disk* button opens a little window in which the parameters *name, size, blocksize, location* and *password* can be entered. The size excluded, all of them are optional.
- Clicking on *Load Disk* opens a file dialog to browse and select and open an already created disk.
- The import button will pop up a little window which asks the user to tell whether he wants to import a file or directory. An according dialog is then displayed. The current working directory is used as destination of the import.
- Similarly, the export button provides a folder dialog to select where the selcted entries should be exported to.
- As one can see, the paste button is gray, meaning not clickable. The buttons' states are updated after certain events; and paste is only clickable if either the move or copy button were clicked.

1.1 Design

As hinted before, we implemented the core requirements. The main classes involved are the following:
VfsExplorer.cs, RemoteConsole.cs, LocalConsole.cs, VfsManager.cs

1.1.1 VfsExplorer.cs

This class basically catches every event that happens on the GUI and forwards a reaction to the *remote console.* Those events are mainly clicks on buttons, items in the *treeview* or the *listview.* Those actions trigger the creation of a command which in the end will be executed by the *VfsManager.*

1.1.2 RemoteConsole.cs

This class connects the *LocalConsole.cs* with the *VfsExplorer.cs.* It forwards the commands received from the *VfsExplorer* to the *local console* and returns results from the *local console* back to the *VfsExplorer.* It also forwards error messages and handles queries that the user needs to answer for certain commands to be executed.

1.1.3 LocalConsole.cs

This class connects the *RemoteConsole.cs* with the *VfsManager.cs*. On creation it creates a thread that is responsible for executing each task in a taskqueue. We do this in order to prevent stalling when giving many or long tasks (a bit more on this later).

1.1.4 Vfsmanager.cs

As in part one, this class still implements the core functionality like *copy, delete, rename, import, export* and so on. The main difference now is that all those functions are called by the new function:

```
public static void ExecuteCommand(string command)
{
            var input = new
               AntlrInputStream(command);
            var lexer = new ShellLexer(input);
            var tokens = new
               CommonTokenStream(lexer);
            var parser = new ShellParser(tokens);
            var entry = parser.compileUnit();
            var walker = new ParseTreeWalker();
            var exec = new Executor();
            walker.Walk(exec, entry);
}
```

Listing 1: Command Execution

As mentioned, the parameter *command* is created in the *VfsExplorer.cs*. To give a little example, this code snippet is executed when the *createDirectoryButton* is clicked.

```
                string command = "mkdir " + Address
                   + "/" + window.Result;
                Console.Command(command);
```

Listing 2: Command Creation

1.1.5 Why sending over two consoles?

This design choice is preliminary work for the synchronization server in part three. There, using a synchronization server, we will propagate changes on disks between machines. We want to use the *remote console* as that propagation instrument and add just the necessary functionality to be able to work as such. The server, as currently the *remote console*, will be connected with the *local console.*

2 Requirements

Our VFS Browser provides the following features:

- **Execute core commands without writing to a console.**
  We simply provided a GUI with buttons which got rid of the necessity of having the user write the commands himself.

- **Single and multiple selections of files and folders.**
  The *Explorer* contains a field called *selectedNames* in which the names of the selected entries are stored (as strings). To select an entry, one can click on it with the mouse and by holding the Ctrl-Key multiple entries can be selected.

- **Support for keyboard navigation.**
  We allow keyboard navigation by typing an address in the *addressbar*, when a directory is selected or when an item in the treeview is selected. After each of these cases, *Enter* has to be pressed. Navigation into the parent folder is achieved with the Backspace-Key.

- **Support for mouse navigation.**
  Mouse navigation is done by either clicking on an item in the *treeview* or by double clicking a directory in the *listview.*

- **(Advanced) Search. (Bonus feature)**
  Our search allows to chose between case insensitive and sensitive search. To restrict the search to a folder is just as possible as to include its subfolders. Also supported are wildcards for which we use the *** symbol.

- **Responsive UI. (Bonus feature)** As shortly mentioned before, the *LocalConsole* has a queue of tasks and a thread which constantly dequeues the tasks and executes them. Therefore the user can keep sending commands which then simply will be added to the queue. We implemented two more methods methods in the *VfsManager* to keep the navigation and listing of the entries correct. In simple words, whenever we should list the entries we check if the entry is loaded and only then it is displayed. Otherwise we enqueue the task. The procedure executed by the Worker-Thread can be found in listing 3 (line-compressed).

- **Drag and Drop. (Partial) (Bonus feature)** So far, we allow copying and importing when dragging files into the listview. **TODO: Export and use of treeview not yet supported.**

```
        private void workerThreadProcedure()
        {
            VfsTask task;
            while (true)
            {
                if (tasks.TryDequeue(out task))
                {
                    if (task.Command == "quit")
                        return;
                    remote.SetBusy();
                    lastCommand = task.Command;
                    VfsManager.ExecuteCommand(task.Command);
                    remote.SetReady();
         }   }   }
```

Listing 3: Worker-Thread's Procedure