Project Description:
Personal Virtual File System

ETH Zurich

March 3, 2014

Hand-out: March 3, 2014
Due: 19 May 2014 (final deadline)

1 Overview

The goal of this project is to develop a single-file Virtual File System (**VFS**) in either Java or C#. A VFS is usually built on top of a host file system, and it enables uniform access to files located in different, possibly heterogeneous, host file systems. A single-file VFS can also be used to implement higher-level functionalities, such as virtual machines, emulators, encryption (storing files in encrypted form), and even Integrated Development Environments (such as IntelliJ[1]). You can find more information about VFSs on Wikipedia[2].

2 Deadlines

All project work must be done in teams of **3 people**. The project itself is divided in 3 parts, each with its own deadline:

- VFS core (due 7 April 2014)
- VFS browser (due 29 April 2014)
- VFS synchronization server (due 19 May 2014)

By each deadline you must hand in, together with the code of your implementation, a written report describing the design and implementation of the corresponding part. The project is structured so as to allow incremental development.

3 Code Repository

Use the SVN repository at `https://code.vis.ethz.ch/`. Create a project using as name the group ID that you have chosen (or was provided to you). Then, add course assistants usernames as project member (see Section 8 for the usernames of assistants).

4 Project Grading

The project awards a total of 100 points, split as follows:

- Requirements coverage, max 40 points

[1] `http://confluence.jetbrains.com/display/IDEADEV/IntelliJ+IDEA+Virtual+File+Systemg`
[2] `http://en.wikipedia.org/wiki/Virtual_file_system`

- Code quality, max 20 points
- Unit tests, max 15 points
- Final report, max 15 points
- Bonus features, max 10 points[3]

The project grade will be determined based on the project's final implementation and the final report as submitted by 19 May 2014. However, each team has to provide an implementation and a report summarizing the current design and project status at each intermediate deadline. The implementation delivered at each intermediate deadline must compile and execute.

4.1 Statement coverage of unit tests

Statement coverage[4] is a measure used in software testing to describe the percentage of statements that have been tested: the higher the coverage, the more thorough the tests are. In this project we will use the following tools to collect coverage information about your tests.[5]

- Java: Eclipse with the EclEmma plugin (`http://www.eclemma.org/`)
- C#: JetBrains dotCover (`http://www.jetbrains.com/dotcover/`)[6]

4.2 Code smells

Code smells[7] are indications of potential problems in the source code. We will use the following tools to detect smells in your code:

- Java: PMD (`http://pmd.sourceforge.net/`)
- C#: VS Code Analysis[8] (`http://msdn.microsoft.com/en-us/library/3z0aeatx.aspx`)

For both tools the quick start guide is available on the course's website.

5 General notes

1. The VFS core, except for bonus features, should *not* rely on any DBMS system or third-party library.
2. The system handle incorrect user input graciously. For example, when a user tries to import files from a nonexistent directory, the VFS should not crash.
3. Design your application in a modular way to support separation of concerns. For example, VFS core should not depend on the GUI.

[3] There's a total of 51 bonus points available. The points for each bonus feature are the *maximum* number of points that can be achieved for that feature. If you achieve more than 10 bonus points in total, the remaining points can be used to compensate missing points in other areas of the projects (but not the exam).

[4] `http://en.wikipedia.org/wiki/Code_coverage`

[5] All tools will be running under the default settings.

[6] You can use dotCover with the academic license – we'll distribute it soon.

[7] `http://en.wikipedia.org/wiki/Code_smell`

[8] Please note, that this tool is not available under Mono stack. You can still develop your project under Mono, but you have to run VS at least once to get feedback from the tool.

6 Project Details

6.1 Part 1: VFS core (due 7 April)

The goal of the first part is to develop the **VFS core**. The VFS core operates on **virtual disks**, each being a single file in the host file system. The core provides an API to facilitate creating, modifying and deleting virtual disks. Particularly, the core provides means for its client to: 1) create a virtual disk of user-specified size, as well as delete an existing virtual disk; 2) import files and directories from the host file system to a virtual disk, as well as export files and directories from the VFS into the host file system; and 3) navigate through the directories of a virtual disk, as well as rename, copy, remove, and move existing files and directories in the virtual disk. The size of a virtual disk is fixed throughout its lifetime, and errors should be reported when pending operations are infeasible due to this restriction.

6.1.1 Requirements

1. The virtual disk must be stored in a single file in the working directory in the host file system.
2. VFS must support the creation of a new disk with the specified maximum size at the specified location in the host file system.
3. VFS must support several virtual disks in the host file system.
4. VFS must support disposing of the virtual disk.
5. VFS must support creating/deleting/renaming directories and files.
6. VFS must support navigation: listing of files and folders, and going to a location expressed by a concrete path.
7. VFS must support moving/copying directories and files, including hierarchy.
8. VFS must support importing files and directories from the host file system.
9. VFS must support exporting files and directories to the host file system.
10. VFS must support querying of free/occupied space in the virtual disk.

6.1.2 Bonus Features

Basic

1. Compression implemented with third-party library. (1 point)
2. Encryption implemented with third-party library. (1 point)
3. Elastic disk: the virtual disk can dynamically grow or shrink, depending on its occupied space. (2 points)

Advanced

1. Compression implemented from scratch (you may use arithmetic compression[9]). (2 points)
2. Encryption implemented from scratch. (3 points)
3. Large data: the VFS core can store and operate with an amount of data that doesn't a standard PC RAM (about 4 Gb). (3 points)

[9] `http://en.wikipedia.org/wiki/Arithmetic_coding`

6.1.3 Hints

1. Develop a set of unit tests prior to development, try to work using the Test-Driven Development (TDD) approach.
2. The key challenge is developing an efficient storage structure at the physical layer. You can take an advantage of specialized data structures like B-trees[10].
3. Keep in mind that, in the next part, you will implement file search.
4. At this point, it is helpful to develop a command-line shell client application to test your VFS (using operations like `ls`, `cd`, `copy`, etc.)

6.2 Part 2: VFS browser (due 29 April)

The goal of this part is to provide a Graphical User Interface (GUI) for the VFS core. The target platform is up to you: you can choose between a desktop application (Windows or Linux), a web application, and mobile clients. The GUI must support all operations implemented in Part 1. Navigation should be possible using both mouse (or touch for mobiles) and keyboard. For this part of the project you only have to support a single user; multi-user support is not required. You are encouraged to design a nice and user-friendly interface. Bonus points are given for a responsive GUI that does not stall during long-running operations, like import or search. Programming two different clients is another option to earn bonus points.

6.2.1 Requirements

1. The file browser should be implemented for one of the following platforms: desktop, web, and mobile.
   (a) Web applications written in C# must use Silverlight or ASP.NET.
   (b) Web applications written in Java must either use Google Web Toolkit or JavaFX.
   (c) Mobile applications must target Android or Windows Phone.
2. The browser should support all operations from Part 1 (VFS core). For example, users should be able to select a file/folder and copy it to another location without using console commands.
3. The browser should support both single and multiple selections of files/folders.
4. The browser should support keyboard navigation. The mandatory set of operations includes folder navigation and going to the parent and child folders. Keyboard navigation is, however, optional for mobile platforms (due to limited keyboard functionality).
5. The browser should support mouse navigation (or touch navigation in case of mobile platforms). The required operations are the same as in requirement 4.
6. The browser should support file-name search based on user-given keywords. The search should provide options for: case sensitive/insensitive search; restricting search to a folder; restricting search to a folder and its subfolders.

6.2.2 Bonus Points

Basic

1. Responsive UI: the browser does not stall during long-running operations (file search or import). (3 points)

[10] `http://en.wikipedia.org/wiki/B-tree#In_filesystems`

2. Advanced search: for example, search with wildcards or regular expressions, and approximate search based on some metric such as *edit distance*. (2 points)

3. Nice-to-have features like operation progress report (e.g., the number of files processed during export) or drag-and-drop for move, copy, and import operations. (2 points)

Advanced

1. The browser is implemented for two different platforms. (7 points)

2. Efficient full-text search (using some sort of indexing). (4 points)

Other additional features should be approved by the project assistants on a case-by-case basis.

6.2.3 Hints

1. The web interface of Dropbox is an example of good interface.

6.3 Part 3: Synchronization Server (due 19 May)

The goal of this part is to develop a synchronization server which propagates changes in a virtual disk from one machine to another using a network connection. The management of the distributed VFSs is made on an account basis. That is, in order to use the server, a user must have an account and link a virtual disk to this account. The synchronization should allow a user to work with a single disk on multiple machines. It does not need to support synchronization between multiple users of the same disk. Note that unlinked disks should not be synchronized; also, you do not have to implement a conflict resolution strategy (e.g., for the case where a document is changed on two different machines at the same time). You should extend the browser from the previous part to be a client of the synchronization server. Additional points are given for implementing a concurrent server (where changes on the same account are done simultaneously on different machines).

6.3.1 Requirements for the browser

1. The browser should allow the user to create a new account or to log in to an existing account.

2. The browser should offer to switch to an offline mode, and be able to operate without a connection to the server.

3. The browser should support linking an existing virtual disk to an active account.

6.3.2 Requirements for the server

1. The server should support registration of unique accounts. Each account is identified by name and password.

2. The server should track changes to linked virtual disks of registered accounts and synchronize changes across machines hosting linked disks.

6.3.3 Bonus Points

Basic

1. Provide a set of unit tests using mocking[11] for your implemenation that test client/server interaction. (1 point)

2. Basic Conflict resolution: saving conflicting files as separate versions. (1 points)

Advanced

[11] `http://en.wikipedia.org/wiki/Mock_object`

1. A concurrent client/server system:
   - The browser updates automatically when changes to the disk occur. (3 points)
   - The server is able to synchronize changes introduced concurrently on different machines on the same account. (3 points)
2. Incremental changes: minimize communication between browser and server by only transferring the parts of a file that changed. (4 points)
3. Advanced conflict resolution: implement a conflict resolution scheme, so that concurrent changes to the same file are not lost (e.g., merging text file, when possible, keeping file history[12]). (3 points)
4. File history: provide a history for files and an interface to restore a previous version. (4 points)
5. File sharing: provide a way to share a file/folder with several users. (3 points)

7 Questions

Got a question? Ask it at `piazza.com/ethz.ch/spring2014/2520284001/home`

8 Contacts

Alexey Kolesnichenko (alexeyk)	alexey.kolesnichenko@inf.ethz.ch
Nadia Polikarpova (polikarn)	nadia.polikarpova@inf.ethz.ch
Nikolic Durica (dunikoli)	durica.nikolic@inf.ethz.ch

[12] which also counts as File history bonus feature