System Y: portfolio

Jannes Van Goeye

Robbe Pauwels

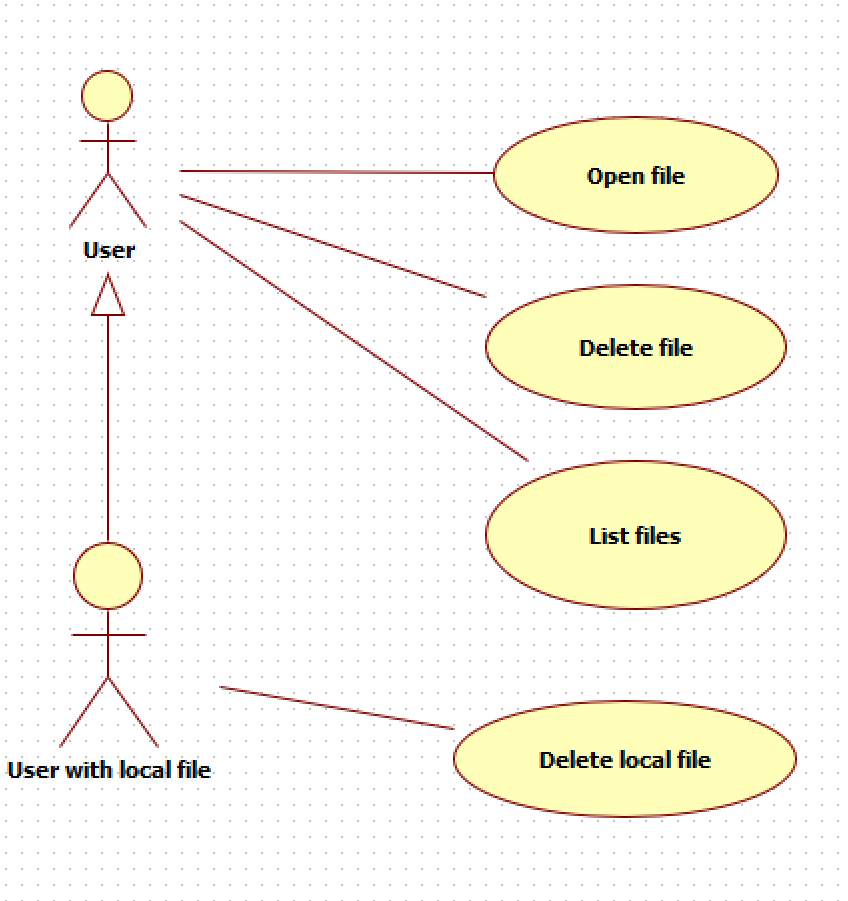
Imre Liessens

# Introduction

System Y is a distributed filesystem: it’s task is to store files in a network.

It’s important that the system is robust and fail-safe, otherwise important files could be lost. User’s local files are automatically sent to at least one other nodes in de network. Every user can access the files that are stored anywhere in the network, in an easy and transparent (the user doesn’t need to know the inner workings) way.

Minimal configuration should be necessary for new nodes, only a selection of the network interface to use (via it’s associated IP) is necessary.



# Assignment 1: Naming Server

The naming server’s task is to keep an overview of all the nodes in the network. It has a supporting role to the nodes and doesn’t store any files. In our design of the system we assume the nameserver is always on and reachable.

When a change in the node-map occurs, it is also immediately saved to disk. This happens in a comma separated list (CSV). In this manner, the list is maintained when the namingserver shuts down, or experiences a failure. When we restart the program the CSV-file is read and the data map pre-populated.

When in memory, the list of nodes is saved in a TreeMap from the Java Collections framework. This collection has some useful properties. It can’t contain duplicate keys (hashes from the node name), which is exactly what we want. The keys are sorted in ascending order, which makes searching for the next and previous node when given a hash a breeze.

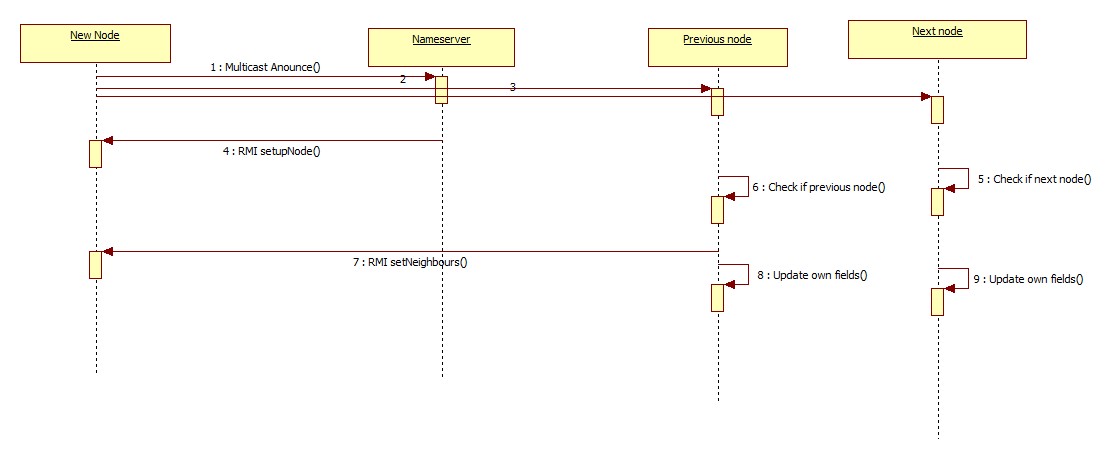
The code is split in two packages: be.dist.common and be.dist.name. The first package contains all classes that are needed in the nameserver as well as the nodes. This makes the distinction between the different classes easier.

The common-package also contains a subpackage “exceptions” (not shown) which contains our custom exceptions. These might also come in handy on the node.



# Assignment 2: Discovery & Bootstrap

The discovery and bootstrap process is executed when a new node is started. Its tasks are manifold, lots of things need to be set up. To accomplish this many classes need to work together, even between different hosts. The function calls are send over the network trough Remote Method Invocation (RMI).



The first announcement is sent via multicast. As internet routers do not forward this type of packet, the discovery process will only work in a LAN. Once the first announcement is sent, the rest of the communication happens trough RMI or TCP (for the file transfers). This has the advantage of a more robust and reliable communication.

It’s quite important that the multicast announcement arrives without problems, as there is no built-in way to check it’s transmission. In a later stadium we could always add a timeout timer to resend this packet when no setup from the neighbours is received. This however yields the new problem of duplicate announcements.

# Assignment 3: Replication

The purpose of System Y is to store files in a distributed way, so the replication is a core part of the functionality. When a node joins, where should the files be stored? How should they be transferred?

Each file should also have some metadata associated with it. This data is always stored on the “owner” node and should be transferred if ownership changes.

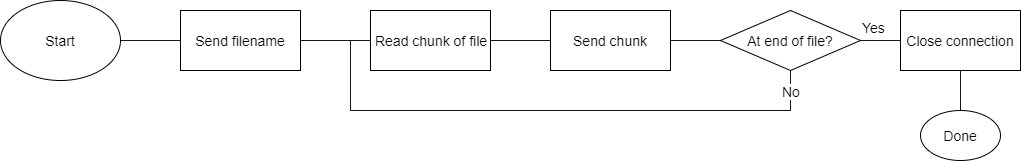
We should also take care to transfer the file name and extension, so that the replicated files can have the correct name. The solution for this is to have some bytes at the beginning of the TCP stream allocated for this purpose. As the maximum file name length in most common platforms (Windows with NTFS, MacOS wit HFS+ and Linux with ext4) is 255 characters, this seems like a good choice.

|  |  |  |
| --- | --- | --- |
| TCP header | 255 bytes | [0 - …] bytes |
| Filename + extension | File data |

Most of the bytes will remain unused, but this is a small overhead to pay for the added simplicity. Especially because this happens only once per file transfer.

While transferring files we should take care to do the process in chunks. We read a piece of the file and send it on its way. Then we do this for the following parts, until the full file is send.

The flowchart illustrates this process:



TCP automatically provides the splitting of the file in appropriately sized packets and also protection against failures.