ugr Universidad de Granada ETSIIT

DDSBox
Real-time file sharing distributed system

Authors
Víctor Cabezas Lucena
Olmo Jiménez Alaminos

Directors
Juan Manuel López Soler
Juan José Ramos Muñoz

November 4, 2014

Index

Problem

- There are a wide variety of systems used to *share* and *backup* files.
- Most of them are centralized and others lack some properties.

The use of a centralized node implies:

- Less scalable systems (bottleneck).
- Central node failure overrides the full system.
- Lack of confidentiality.

Proposed solution

DDSBox, a real-time file sharing distributed system.

Objectives

- Setting-up a *distributed* and *scalable* system for file sharing.
- Integrate the developed system with OS basic tools.
- Available storage space depending on user's computer disk space,
- Utilize *pub/sub* paradigm through **DDS**

Existing systems I

Dropbox Reference system. Centralized system. Limited storage space. Closed storage system.

Owncloud It is like Dropbox. Centralized system. Setup in own server.

Google Drive Google Docs evolution. Centralized system with online document edition. Limited storage space.

Bittorrent Sync File synchronization through P2P. There is no user role. It's based on a private key system. There are not defined permissions.

Existing systems II

Emule Massive file sharing through P2P. Files are split into parts. Initially centralized servers were used to communicate clients, a de-centralized network was later developed (KAD). Unable to synchronize files or folders.

Rsync File and folder synchronization protocol and application. It uses file compression and incremental versions. One to one communication.

Index

Functional requirements

- Sharing files and folders between users.
- Data transmission must be distributed.
- File sharing is done from a root folder.
- Folder sharing: every user with access to a shared folder must receive all files and folders inside it.
- GUI and console UI.
- Two types of shared folders:
  - Public: Every user in the system has access.
  - Private: Only invited and authorized users.

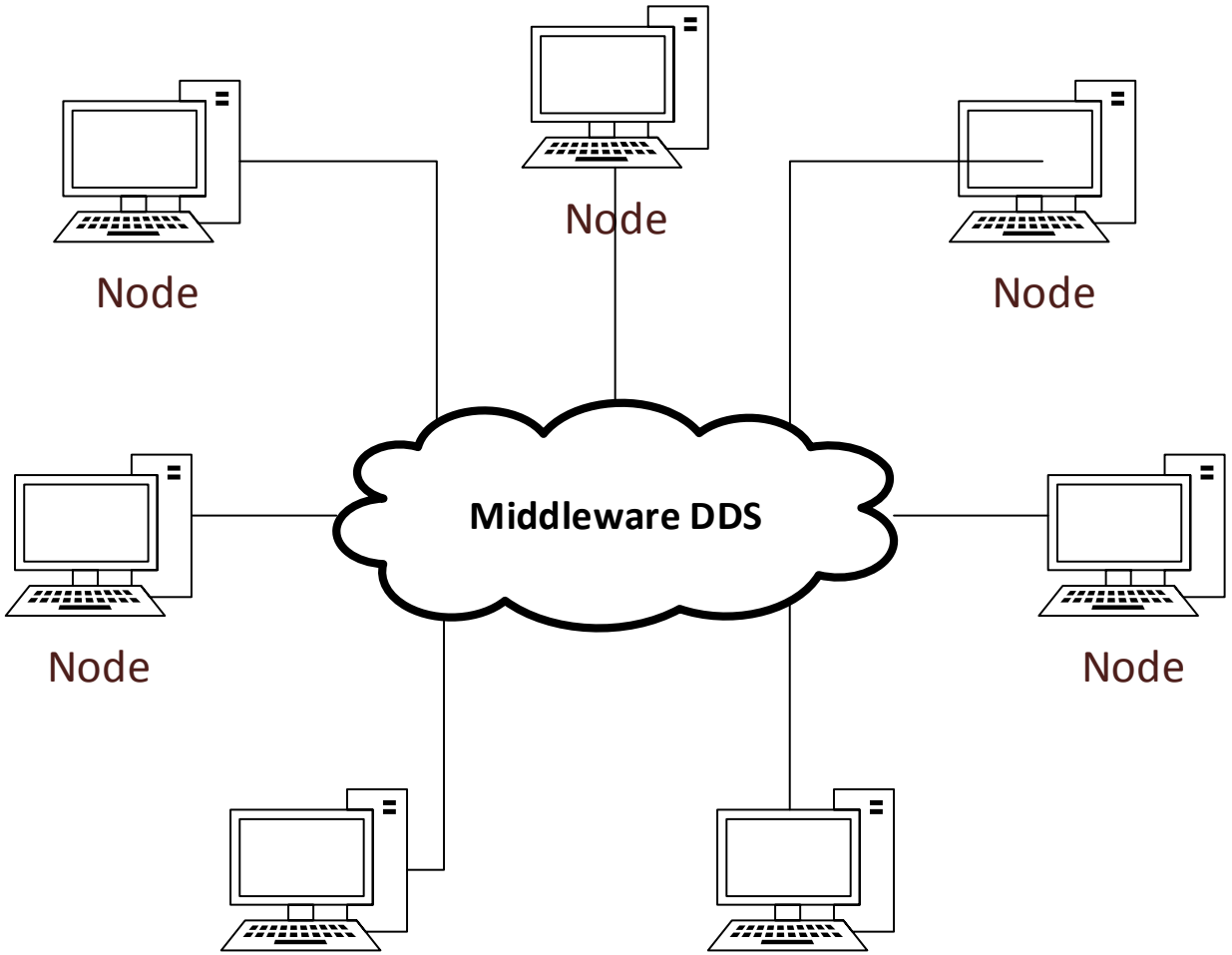
Functional requirements II

Users permissions in shared folders

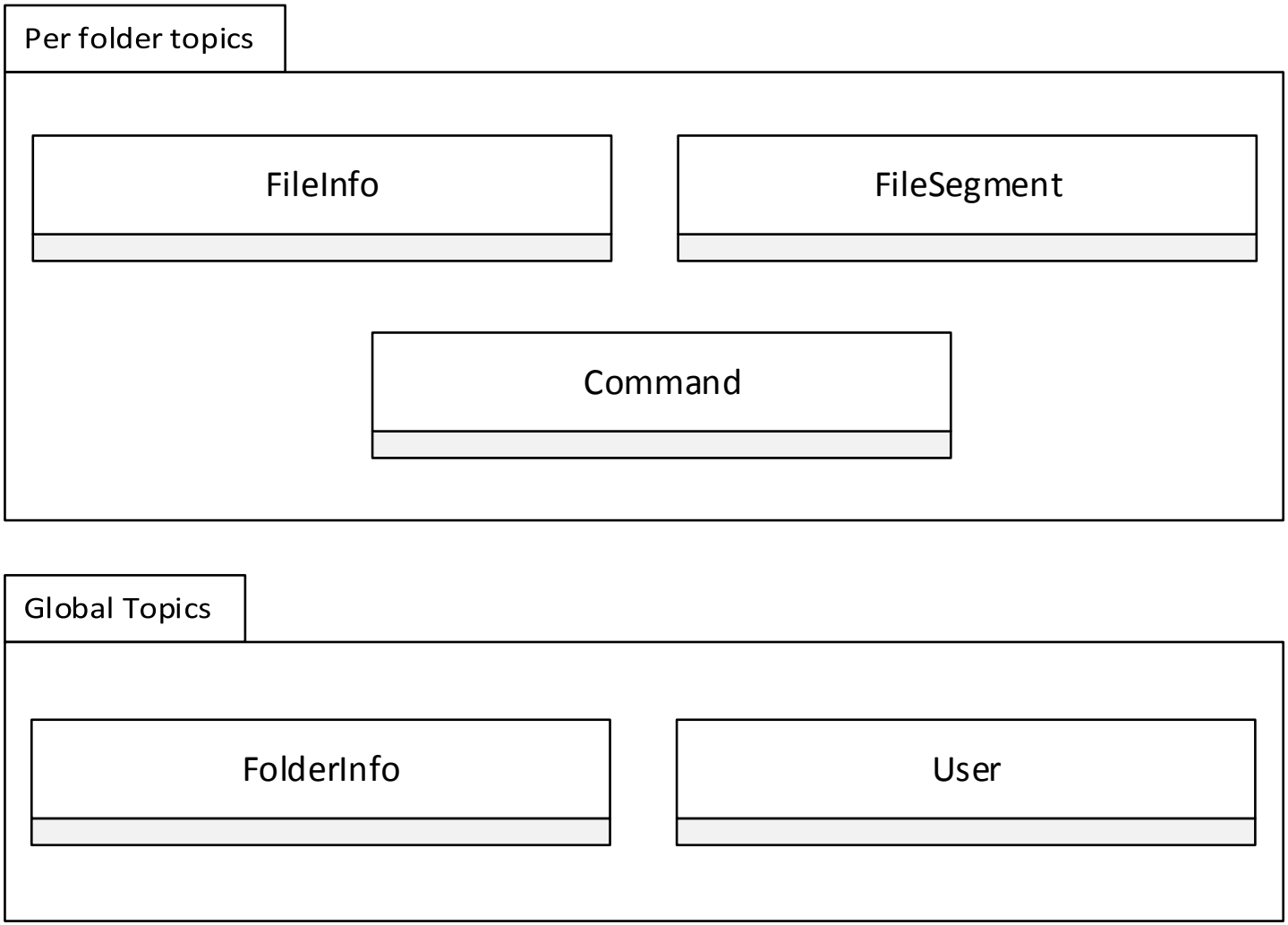
Type	Read	Write	Add user	Add editor
Reader	x			
Collaborator	x	x		
Editor	x	x	x	
Owner	x	x	x	x

Índice

Global architecture



System topics



Per-folder topics

FileInfo

-String userId
-Integer fileId
-binary content

FileSegment

-String userId
-Integer fileId
-Integer idSegment
-binary content

Command

-String userId
-binary content

Global topics

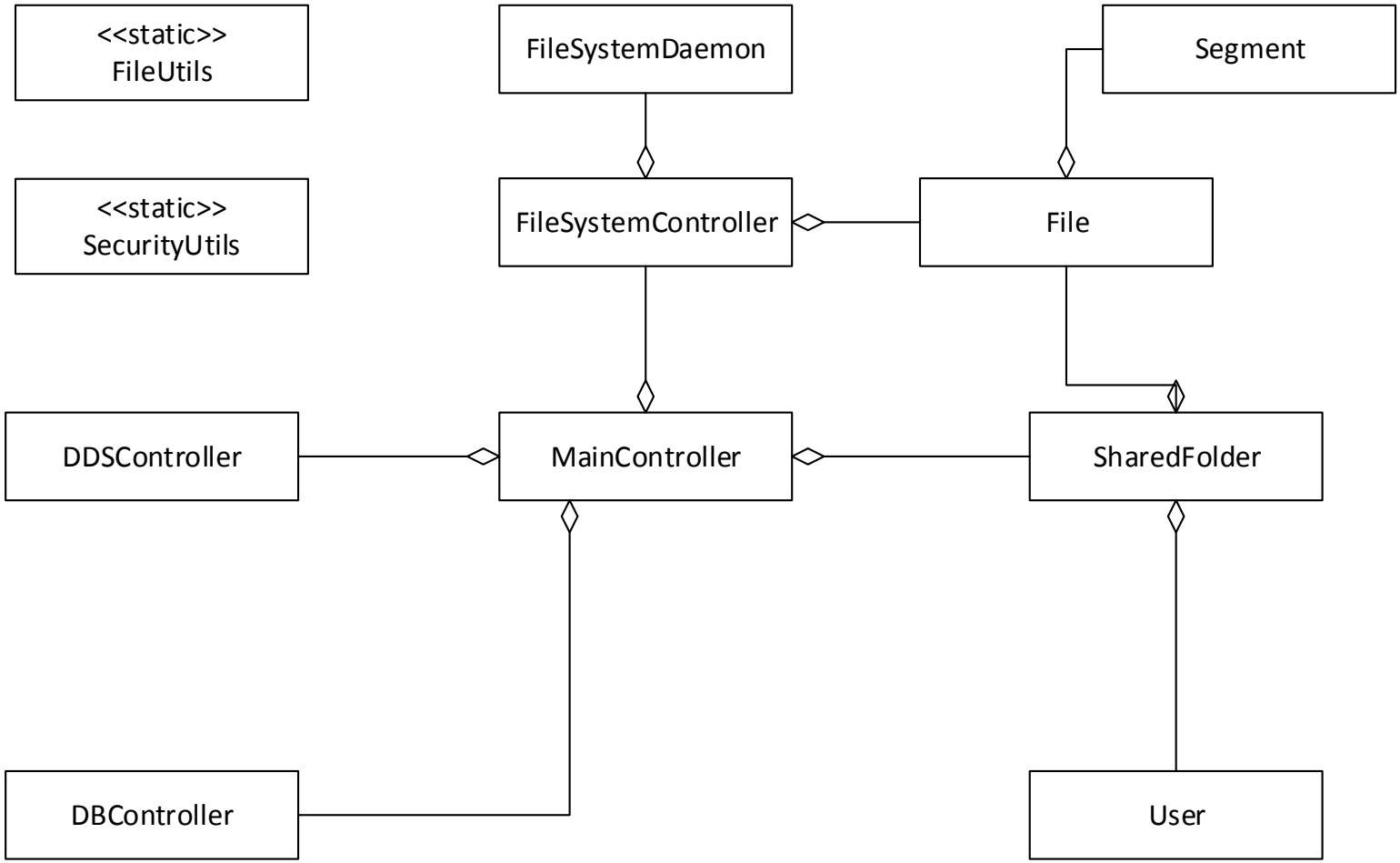
FolderInfo

-String userId
-String destUserId
-binary content
-binary encryptedKey

UserInfo

-String id
-String username
-String realname
-String email
-String publicRSA

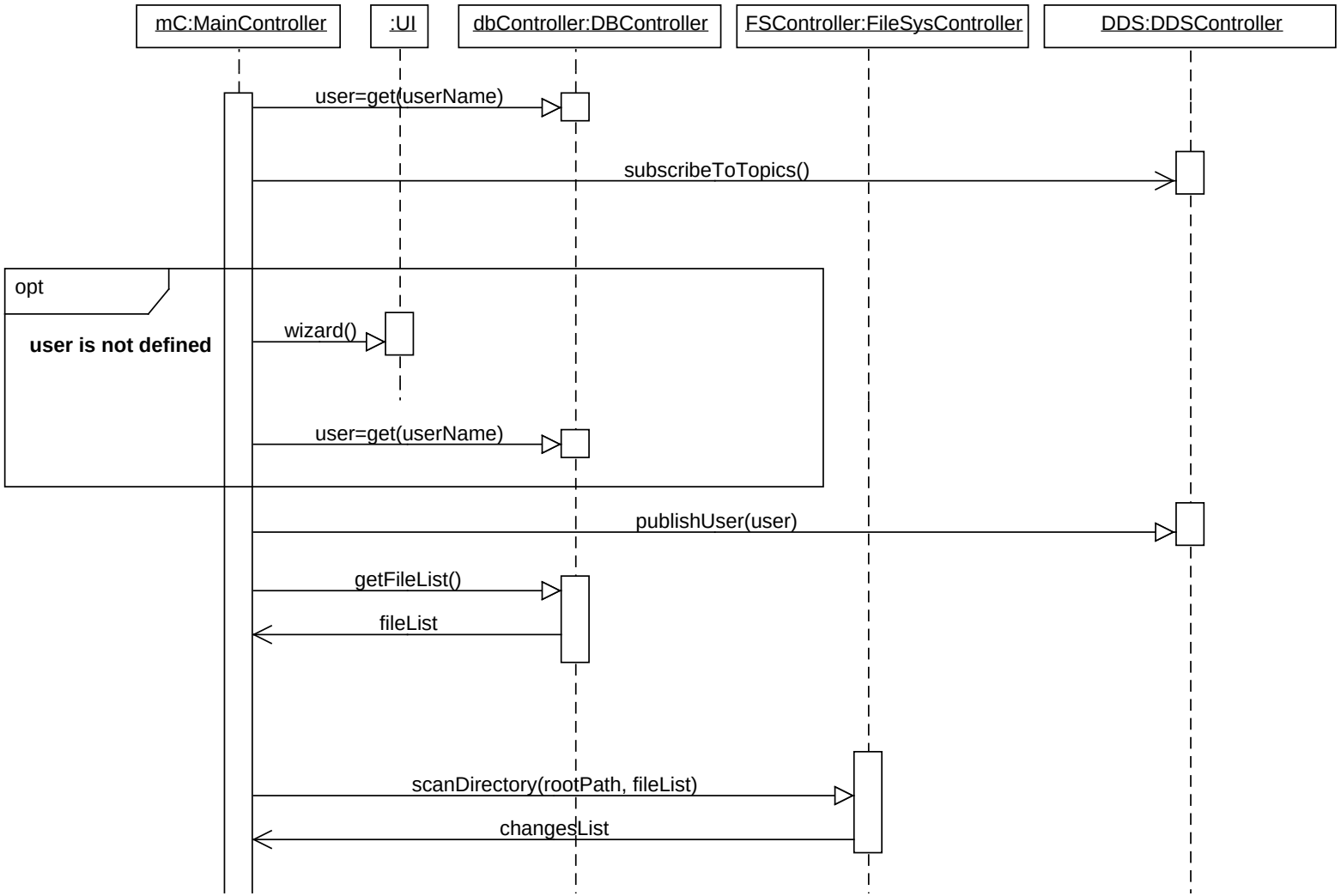
Class diagram



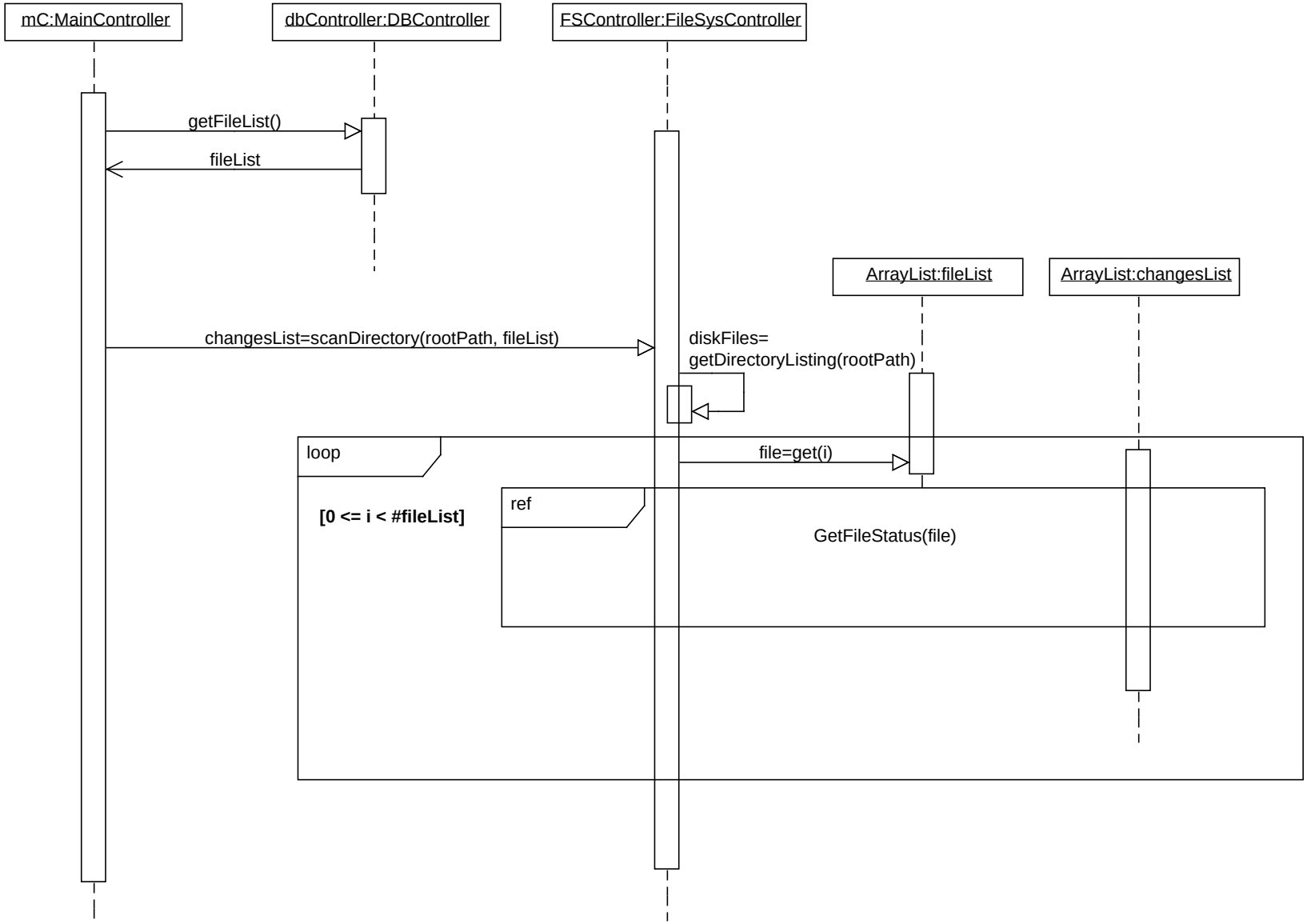
Sequence diagrams

- Application start up.
- Initial folder scan.
- Application shutdown.
- Shared folder creation
- Add user to shared folder
- Shared folder subscription
- Make modification
- Receive modification

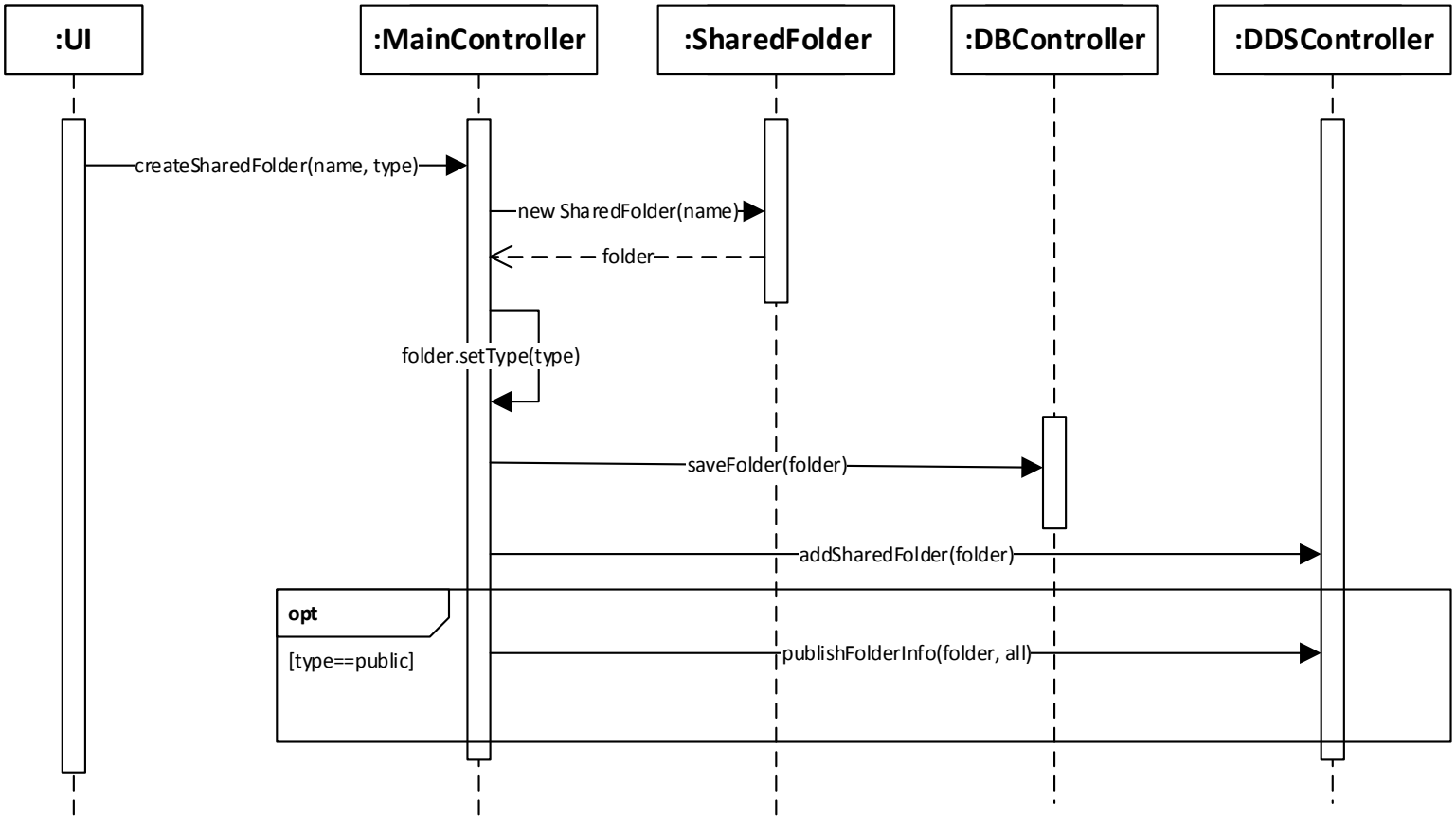
Application start up



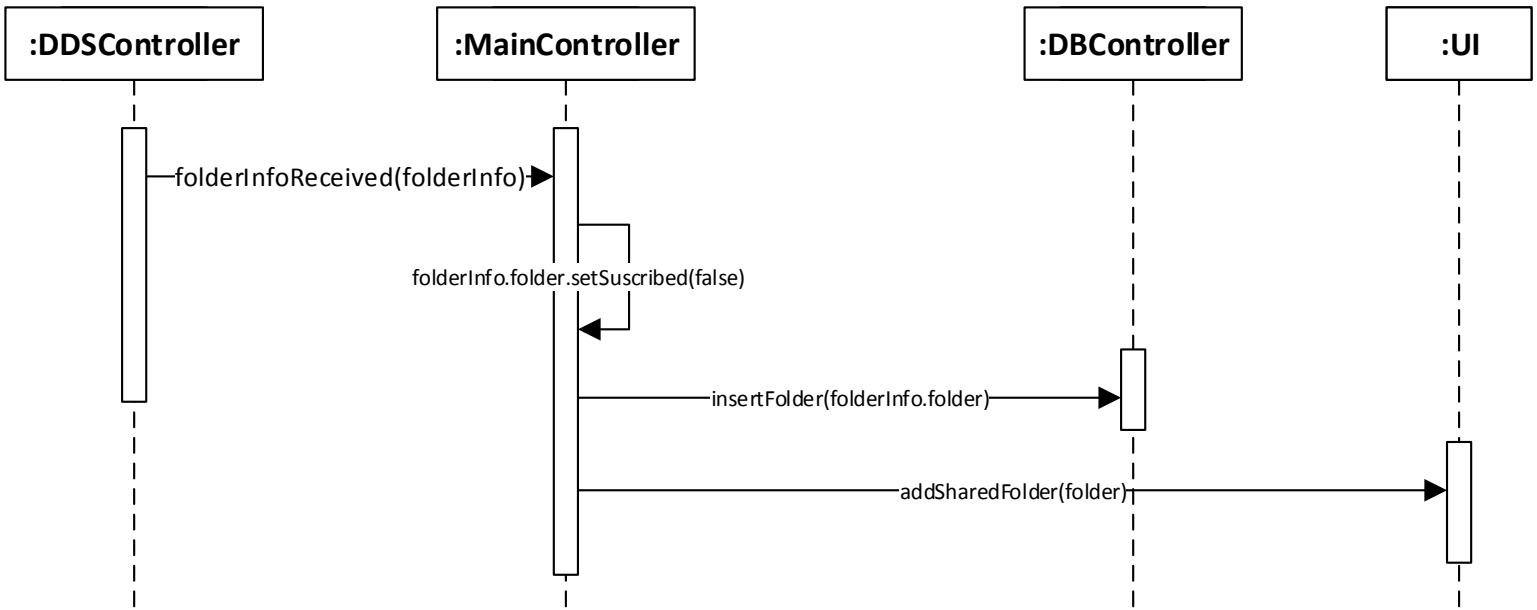
Initial folder scan



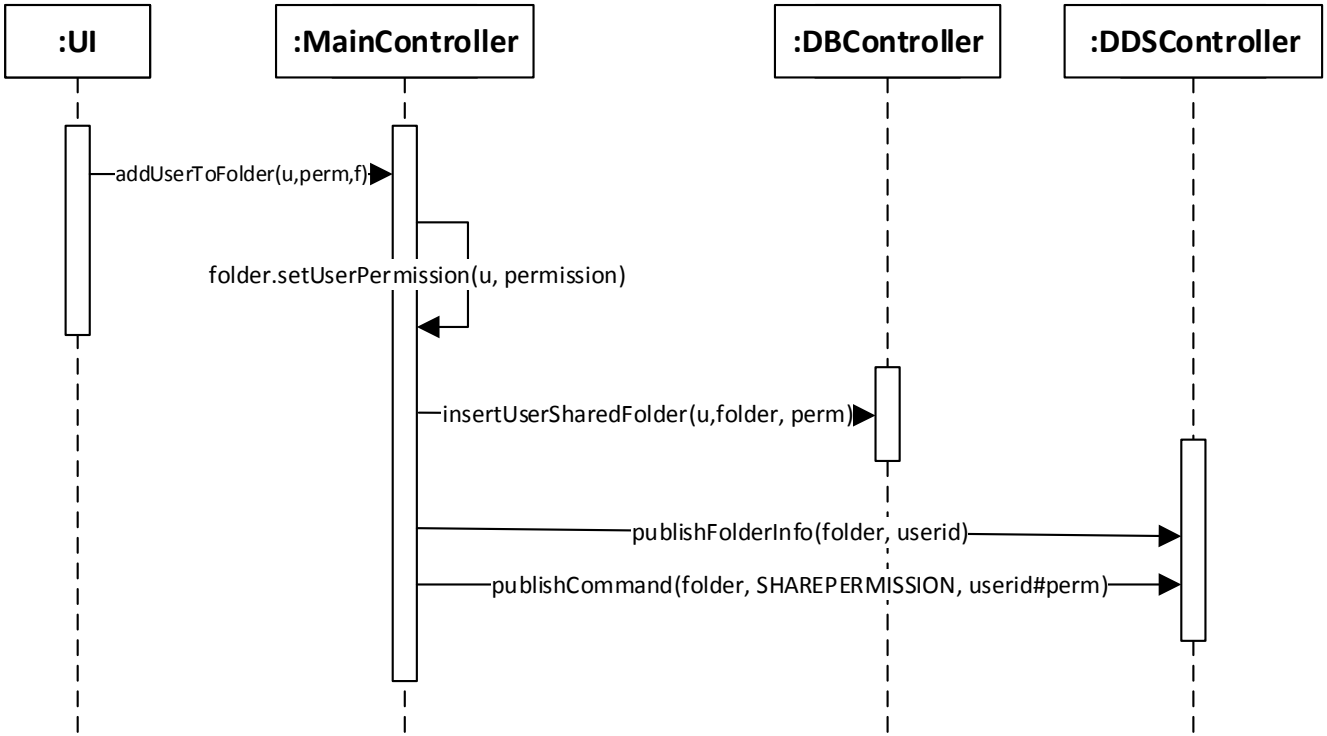
Shared folder creation



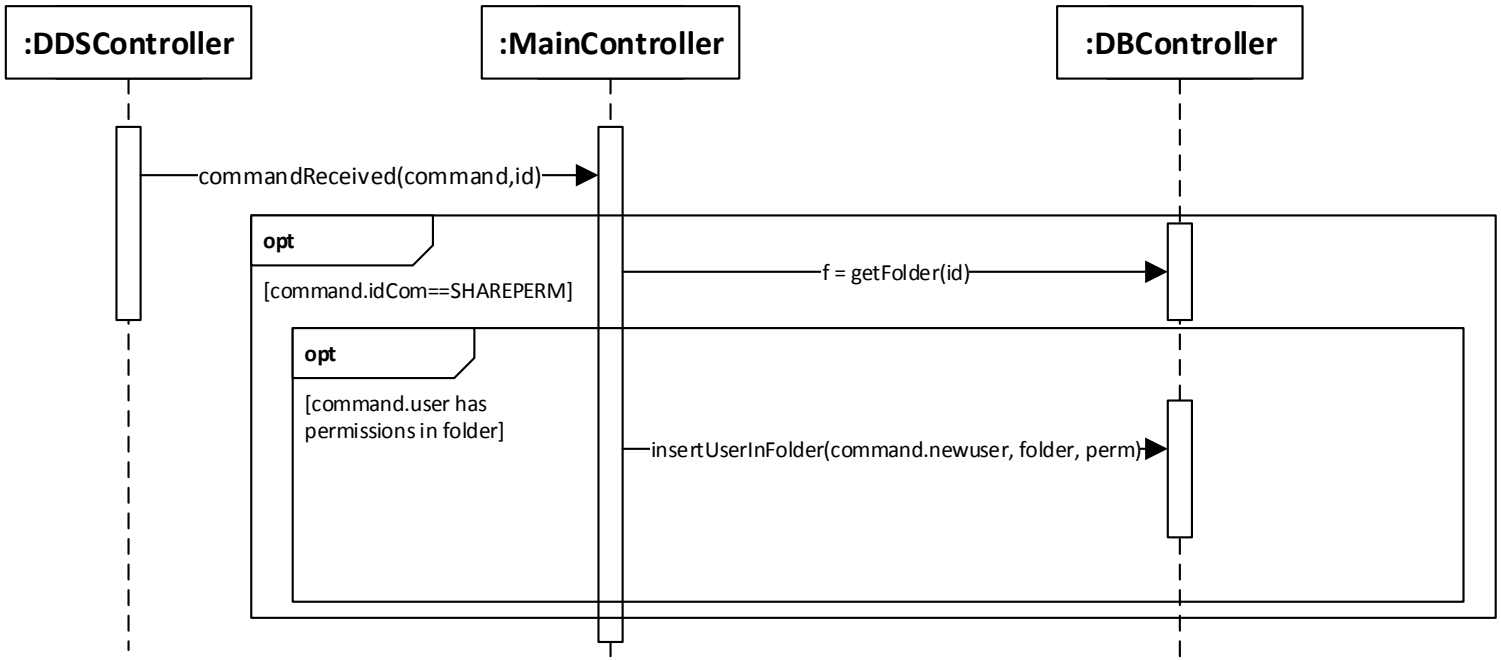
Shared folder creation II



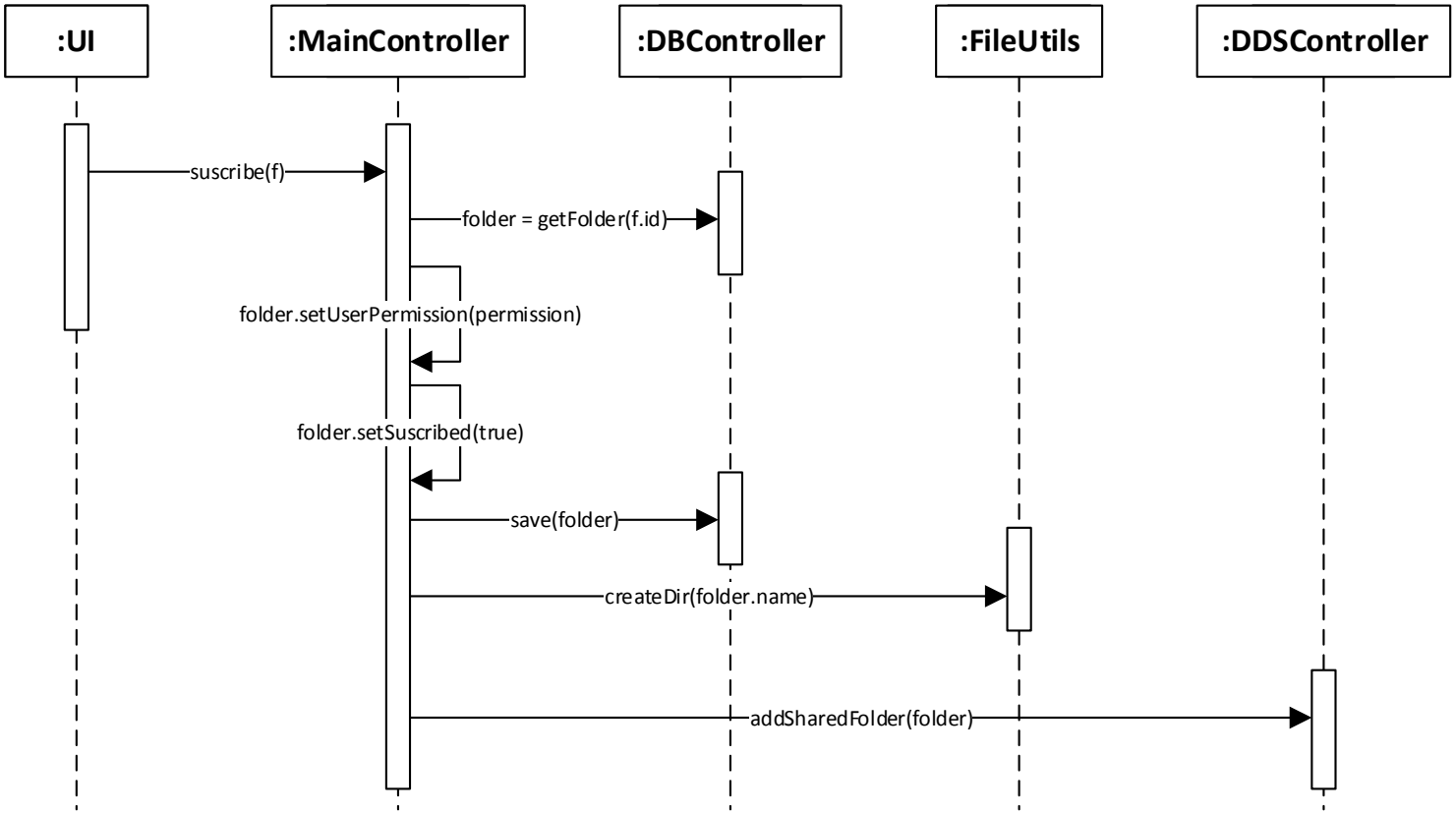
Add user to shared folder



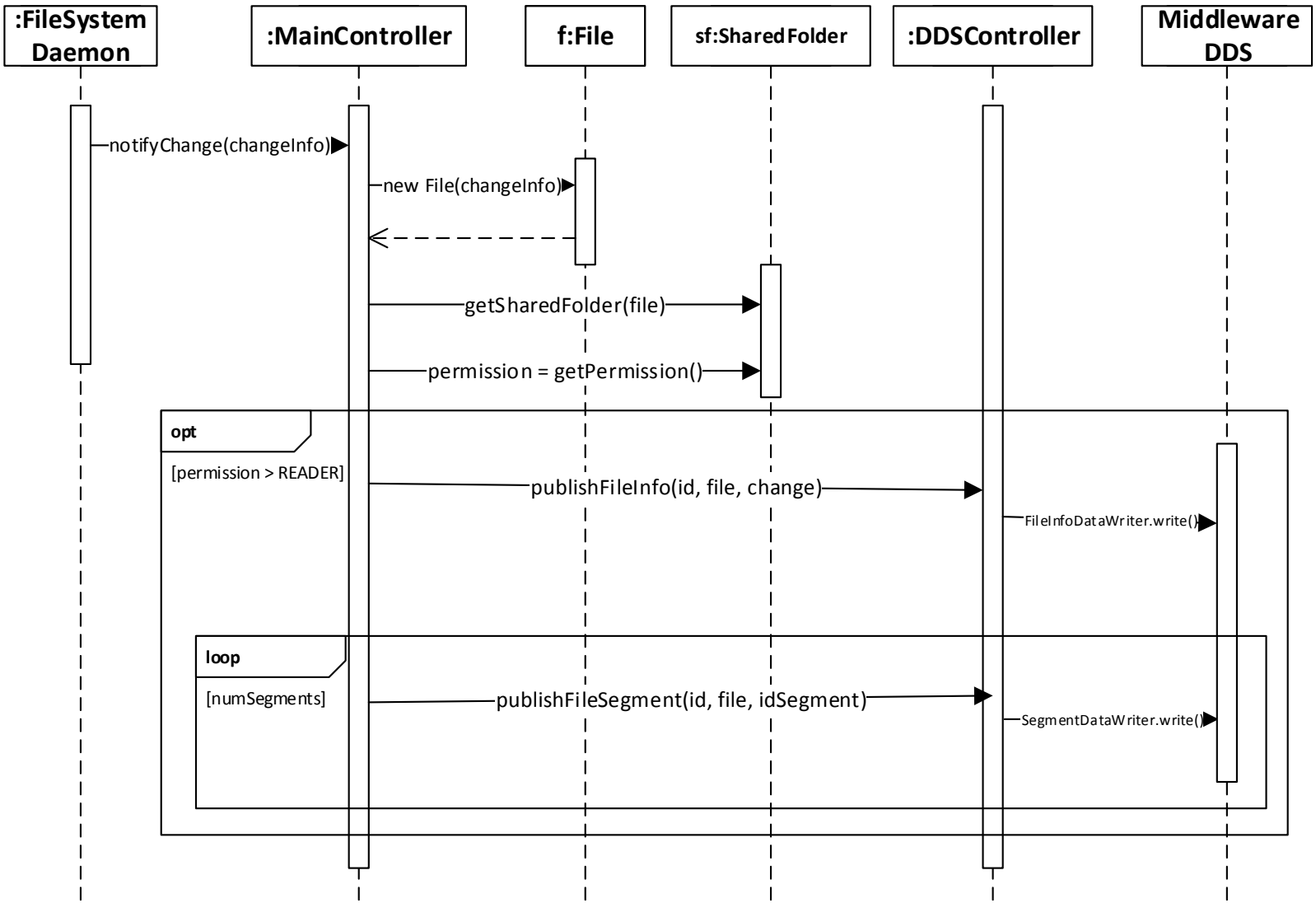
Add user to shared folder II



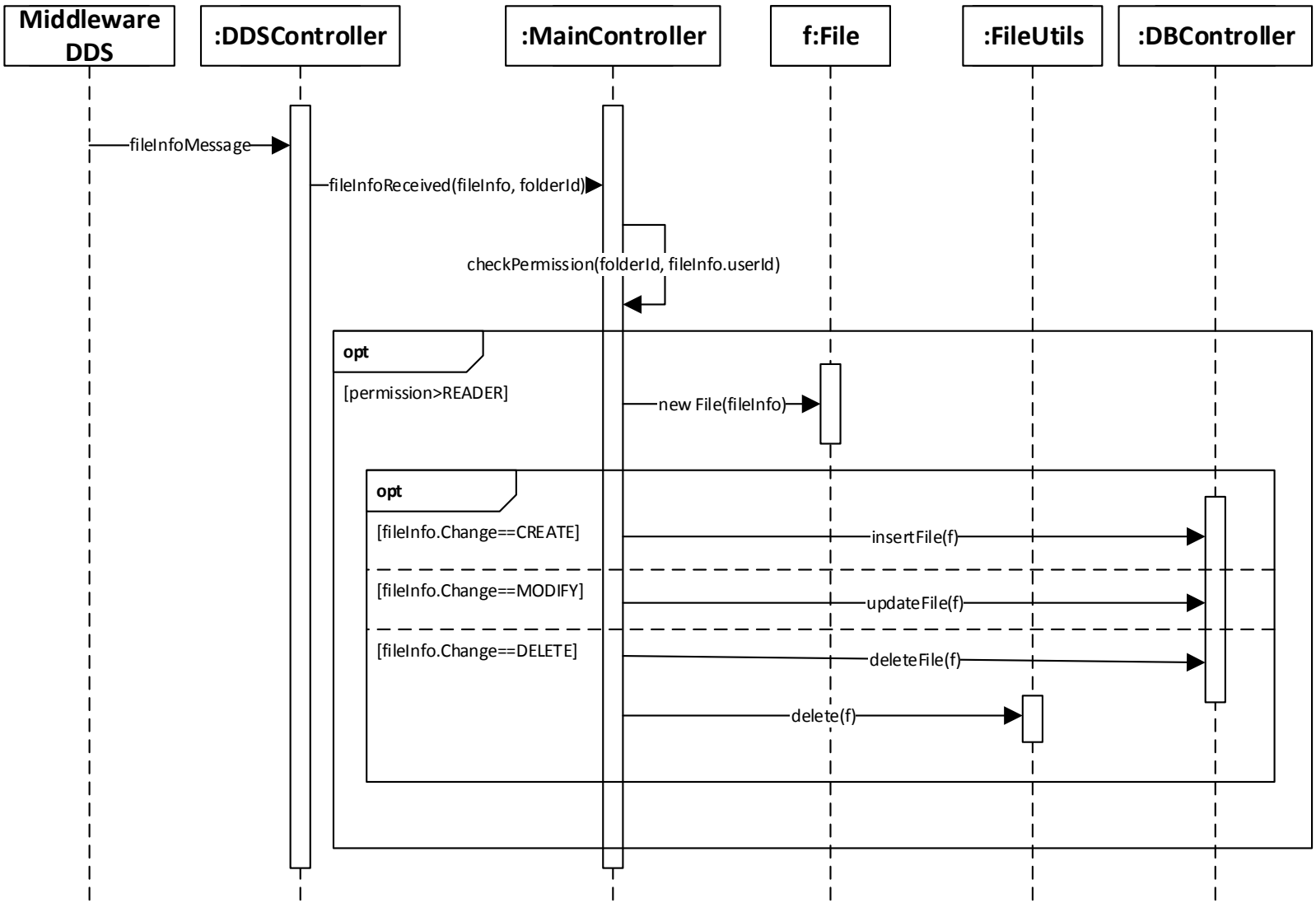
Shared folder subscription



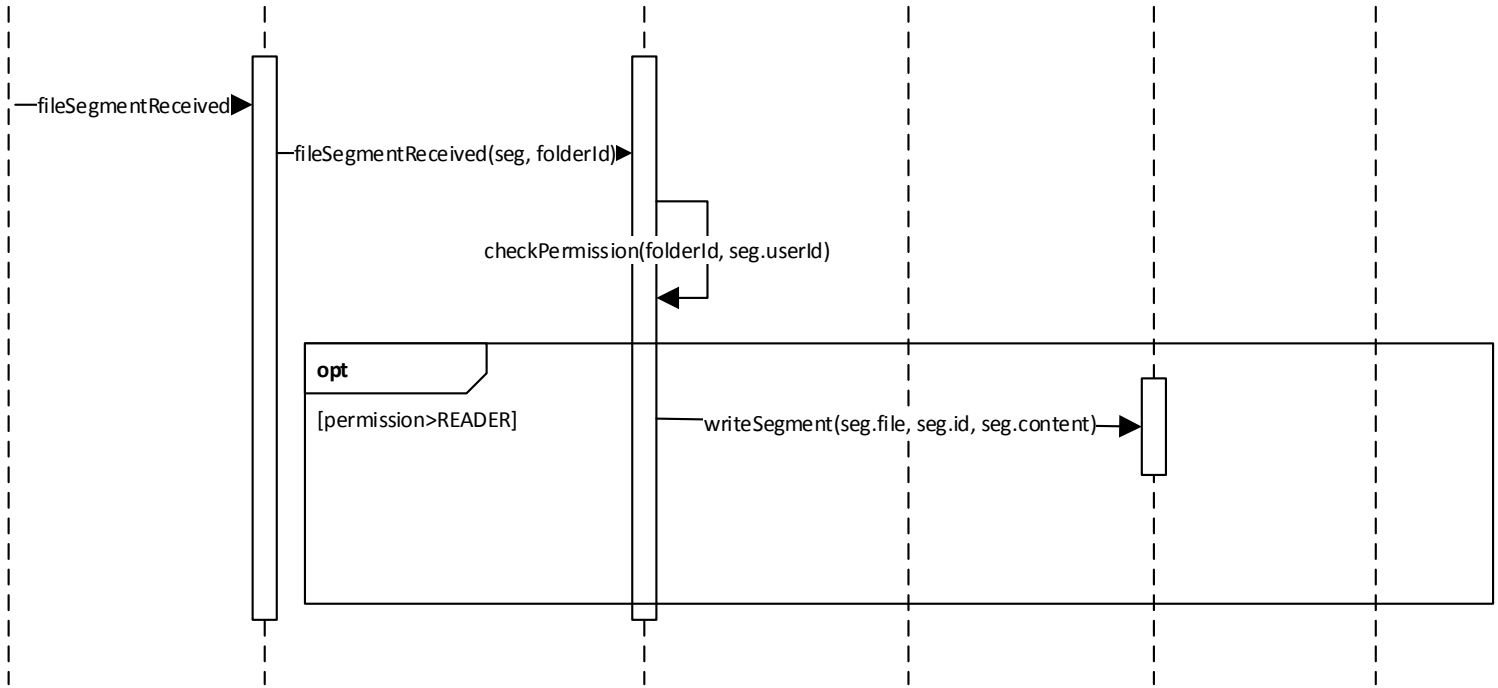
Make modification



Receive modification



Receive modification II



Encryption I

- The application allow private file transmission.
- The need of encrypt this information is mandatory.

Two encrypt algorithms are used to encrypt data:

- **RSA**
- **AES**

Encryption II

AES

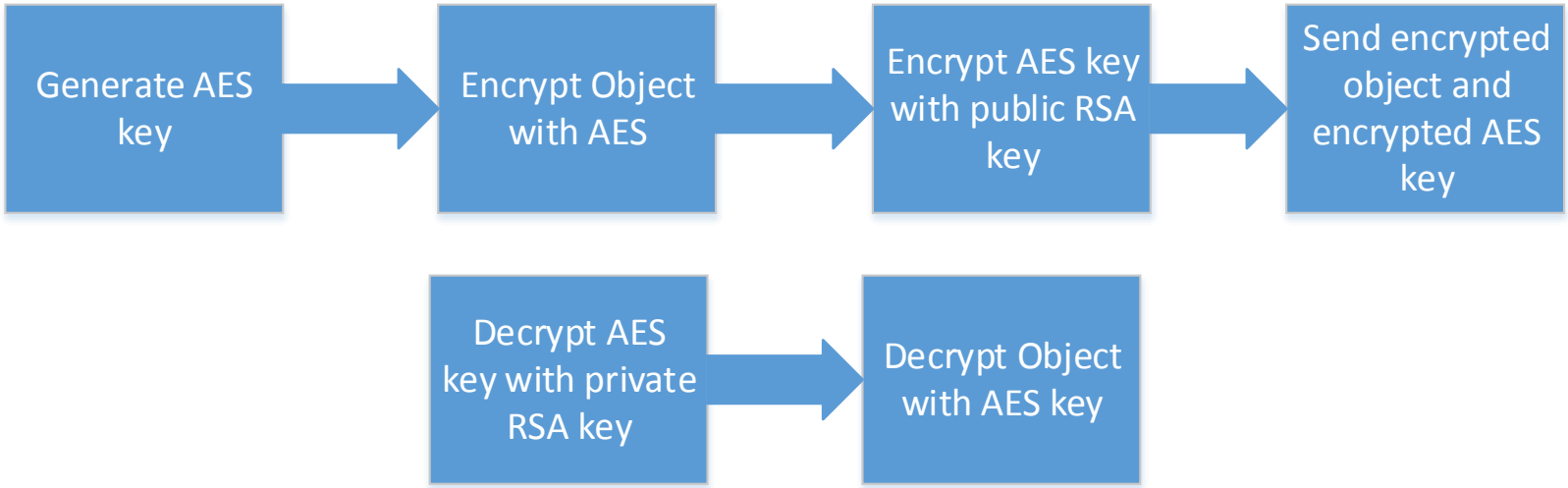
- It is used for the transmission of shared folder topic information.
- Symmetrical cypher.
- Each shared folder has a different AES key.
- Each user inside a shared folder have the AES key.
- That folder data is only available to users with the AES key.

Encryption III

RSA

- It is used to transmit shared folder information from an user to another.
- Asymmetric cypher.
- It is used with AES encryption because RSA cant encrypt large data chunks.

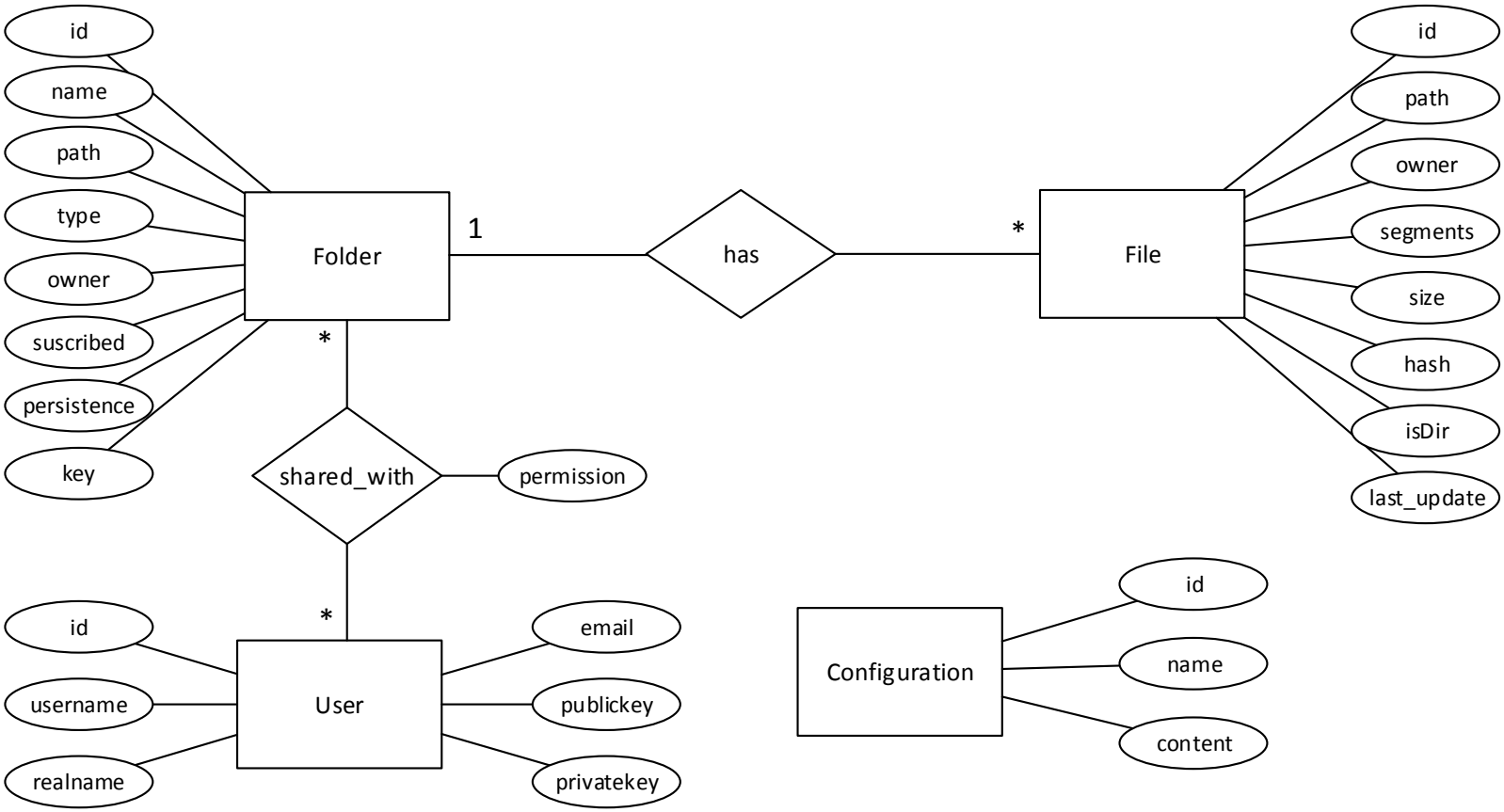
Encryption IV



Local database I

- Each system node should maintain certain information.
- It allows information consistency between application running instances.
- Files modification proccess is improved with local database.

Local database II



Index

Programming language

The application was developed using Java 7:

- Programming language is supported by Middleware API.
- Multiplatform.
- Wide variety of native libraries.

Serialization

The data to encrypt is composed of several attributes. Two options available:

- Encrypt each attribute.
- Encrypt full object: serialize.

Serialization

It allows:

- Object is translated into a binary array.
- Object can be reconstructed from the binary array.

Identifiers

The need of identify each entity is mandatory.

- Sequential identifiers can't be used.
- It must be suitables in a distributed environment (avoiding dupes).

UUID

- 32 hexadecimal characters.
- Randomly generated through various methodologies.
- Java uses a random number generation.
- For 2^{36} keys the possibility of a duplicated identifier is $4x10^{-16}$

Database

Local database engine selected was SQLite:

- Multiplatform.
- Lightweight.
- It doesn't require another installation, just to include a library in the application.

Index

Conclusions

General conclusions

- A system for file sharing using DDS and meeting all established functional requirements has been implemented.
- Two types of shared folders (public and private) and a permission system for the users has been implemented.
- Files are shared in a transparent manner to the user, requiring minimal intervention from the user.
- Multiplatform application, working in Linux and Windows.

Index

Next work

- Implement another transmission systems.
- Implement functionality for WAN networks use.
- Conflicts system detection.
- Customizable version control system.
- Instant Messaging between users.
- Improve GUI.