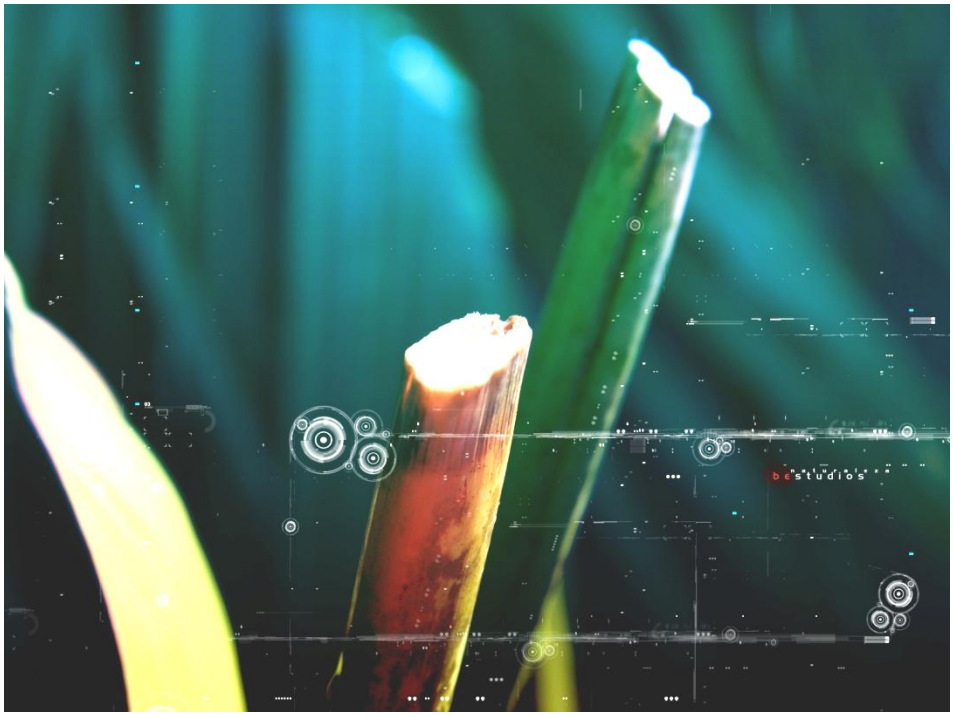
EASYFS



Système de fichiers virtuel

Eléonore KLEIN

Mathieu MARTIN

Pascal MIETLICKI

Jihed OTHMANI

EasyFS

Sommaire

EasyFS

OBJECTIF ET DESCRIPTION DU PROJET

Objectif du projet

- Implémenter un protocole de communication pour les échanges de données avec un serveur Web sous apache grâce au langage PHP.
- Implémenter un système de fichier virtuel représentant l'espace disque de l'utilisateur.

Description

Mise en place d'un système de fichiers virtuels à l'aide de la librairie FUSE (userspace) à partir d'un fichier XML qui contiendra l'arborescence du répertoire de l'utilisateur. L'échange devra être sécurisé, une authentification de l'utilisateur doit être faite au préalable.

Le système de fichiers, une fois monté, devra être navigable, l'utilisateur, selon ses droits, pourra créer, modifier et lire les différents fichiers et répertoires lui appartenant, une mise à jour sur le serveur distant sera, par la suite, effectuée. Une gestion précise des droits sur l'ensemble des fichiers de son répertoire doit être mise en œuvre.

LES OUTILS DE GESTION DU PROJET : EASYFS.FR.NF

Dès la première semaine, nous avons mis en place une interface Web afin d'améliorer la communication interne concernant notre projet. Celui-ci se trouve à l'adresse : http://easyfs.fr.nf

L'utilisation de cette interface est relativement intuitive et permet de :

- poster des nouvelles
- proposer des liens Web
- publier le compte rendu de chaque réunion
- attribuer des tâches
- modifier l'état d'avancement de chaque tâche
- voir l'avancement global du projet
- envoyer et stocker des documents
- discuter et créer des sujets à travers le forum

Vue d'ensemble

Description du projet

Détails du projet

Gestion des tâches

Nouvelle tâche

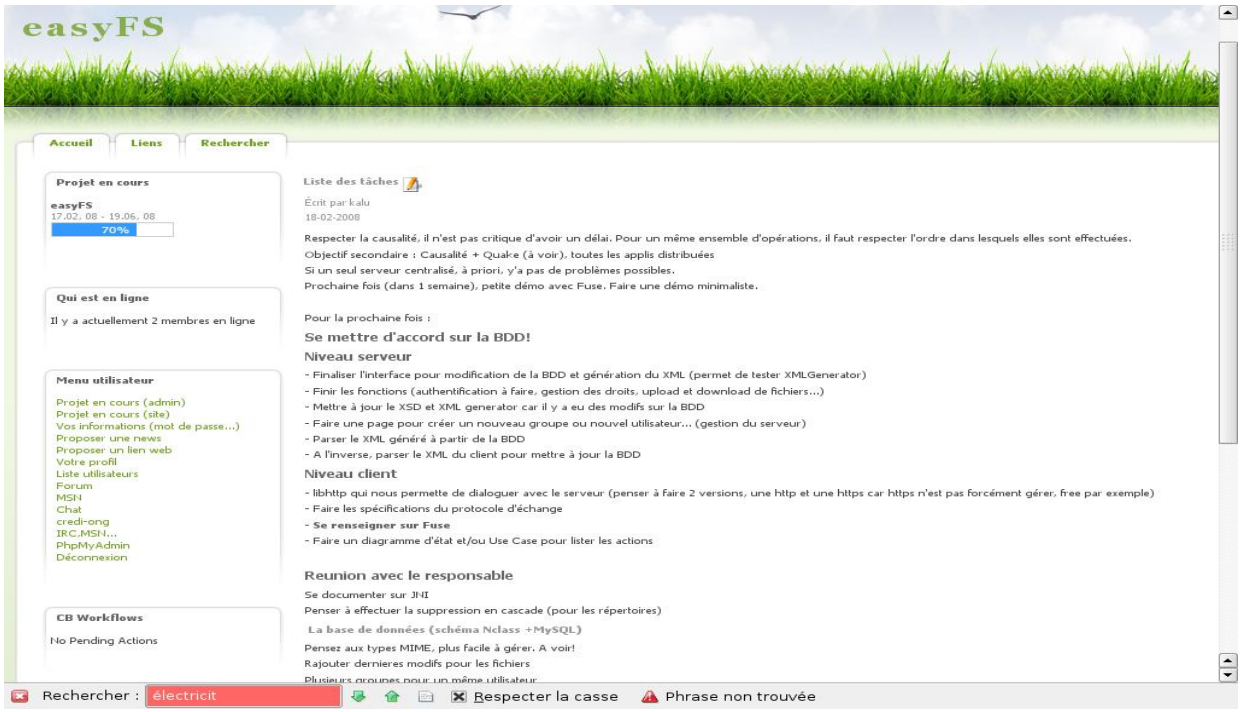
Détails d'une tâche

Vue par mois

Vue par jour

Cette interface nous a été très utile en permettant un gain de temps non négligeable. En effet, nous avons pu partager directement toute les informations récoltées par chacun des membres du groupe, discuter de problèmes comme la gestion des accès concurrents ainsi que nous mettre d'accord sur les solutions à apporter.

Cet outil de travail collaboratif s'est avéré être un atout dans pour le bon déroulement de ce projet. Malheureusement, nous ne l'avons pas utilisé de façon constante du début à la fin. En effet, au début, nous avons fortement profité des avantages de cette interface mais, vers la fin, en nous focalisant sur l'évolution et l'amélioration du codage du logiciel, nous avons un peu délaissé cette interface.



PROTOCOLE DE COMMUNICATION CLIENT/SERVEUR

Communiquer consiste à transmettre des informations, mais tant que les interlocuteurs ne leur ont pas attribué un sens, il ne s'agit que de données et pas d'information. Les interlocuteurs doivent donc non seulement parler un langage commun mais aussi maîtriser des règles minimales d'émission et de réception des données. C'est le rôle du protocole de s'assurer de tout cela.

Voici les règles de communication qu'on a définie pour gérer le dialogue entre le client et le serveur :

1. Le client commence par faire une demande de connexion avec le login et le mot de passe de l'utilisateur

2. Le serveur vérifie les paramètres d'authentification et renvoi un code retour, 200 si tout se passe bien sinon un code erreur bien défini.

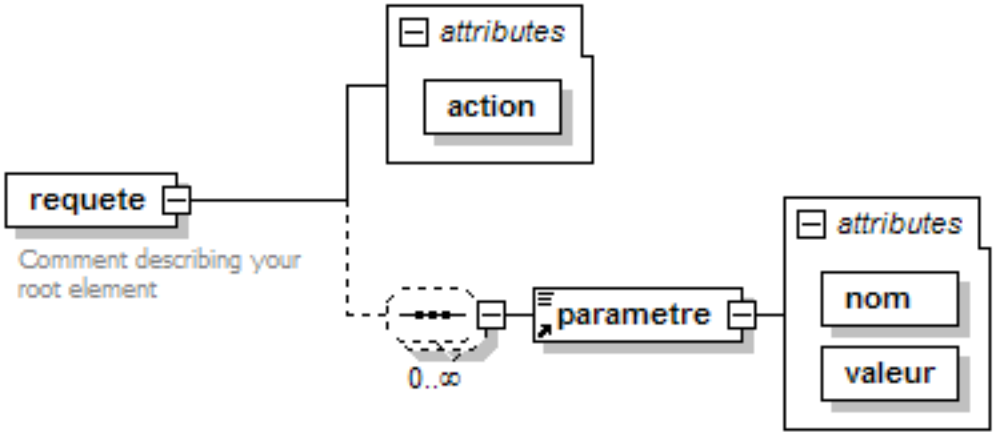
3. Pour toutes les autres actions, on procède de la même manière, c'est-à-dire le client envoi une requête avec les paramètres adéquats et attend une réponse de la part du serveur. La réponse du serveur peut être un code retour, un fichier, la description de l'arborescence du groupe auquel l'utilisateur appartient ou bien un lien.

Cette méta-communication n'est autre que la mise en œuvre d'un protocole de communication.

Les requêtes et les réponses sont écrites en XML, dont l'objectif initial est de faciliter l'échange automatisé de contenus entre systèmes d'informations hétérogènes (interopérabilité).

Les requêtes

Schéma XSD



Chaque requête possède un nom (un type d'action) et une liste de paramètres.

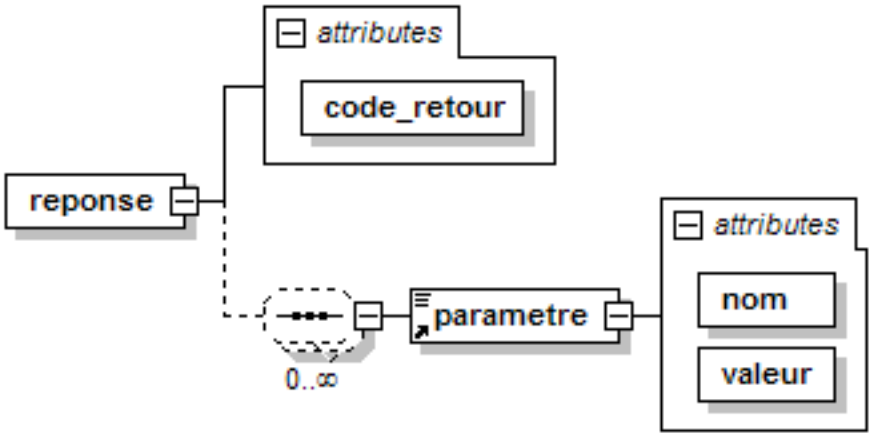
Les paramètres sont composés de deux champs :

- Un Nom, exemple : nom_d_utilisateur

- Et une valeur, exemple : jihedo

Les réponses

Schéma XSD



Comme pour les requêtes, chaque réponse possède des attributs et une liste de paramètres.

L'attribut code_retour permet d'informer le client si tout s'est bien passé ou pas. On distingue plusieurs codes d'erreur, ils sont définis dans la classe 'Codes' du paquetage 'constantes' dont voici un extrait :

```
package constantes;

public class Codes {

    /*CODES DE NOUS*/
    public static final int OK = 0; /* Tout va bien */
    public static final int NO_SESSION = 201;
    public static final int FILE_ALREADY_EXIST = 202;
    public static final int UPLOAD_ERROR = 203;


    /*CODES ERREURS DE FUSE */
    public static final int EPERM = 1; /* Operation not permitted */
    public static final int ENOENT = 2; /* No such file or directory */
    public static final int ESRCH = 3; /* No such process */
    public static final int EINTR = 4; /* Interrupted system call */
    public static final int EIO = 5; /* I/O error */
    public static final int ENXIO = 6; /* No such device or address */
    public static final int E2BIG = 7; /* Arg list too long */
    public static final int ENOEXEC = 8; /* Exec format error */
    public static final int EBADF = 9; /* Bad file number */
    public static final int ECHILD = 10; /* No child processes */
    public static final int EAGAIN = 11; /* Try again */
    public static final int ENOMEM = 12; /* Out of memory */
    public static final int EACCES = 13; /* Permission denied */
    public static final int EFAULT = 14; /* Bad address */
```

Exemples de communication

Requête d'authentification

```
<requete action="authentification">
        <param name="login" value="nessie" />
        <param name="password" value="md5_password" />
</requete>
```

Si tout se passe bien, on envoi une réponse dont le code de retour est égal à 200 avec deux paramètres : l'identifiant de la session et un lien vers le fichier XML qui décrit l'arborescence du répertoire du groupe auquel l'utilisateur appartient. Ce fichier sera par la suite parsé au niveau du client tout comme les réponses du serveur.

```
<reponse code_retour="200" >
       <param name="id_session" value="gsf245gdf2g4df532gdf" />
       <param name="url_xml" value="http://blabla.com/telecharger.php?id_dl=autre_id" />
</reponse>
```

Si on détecte un problème, on envoi le code d'erreur correspondant.

```
<reponse code_retour="numéro_d_erreur" />
```

Requête de création d'un fichier

```
<requete action="authentification">
        <param name="login" value="nessie" />
        <param name="password" value="md5_password" />
</requete>
```

Si tout se passe bien :

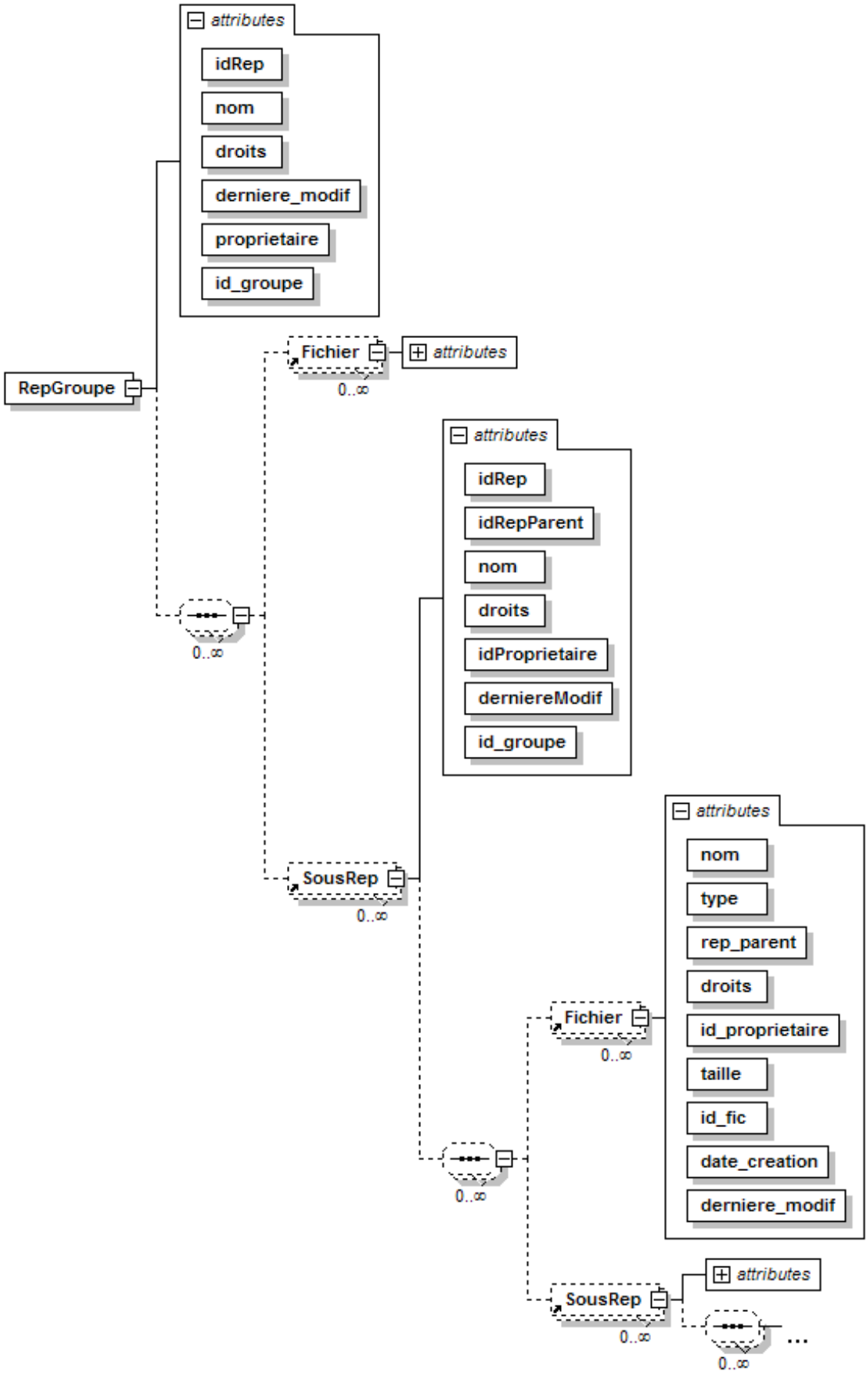
```
<reponse code_retour="200" >
       <param name="id_session" value="gsf245gdf2g4df532gdf" />
       <param name="url_xml" value="http://blabla.com/telecharger.php?id_dl=autre_id" />
</reponse>
```

Si on détecte un problème, on renvoie le code d'erreur correspondant :

```
<reponse code_retour="numéro_d_erreur_correspondant" />
```

La description de l'arborescence au sein du protocole

Afin de décrire l'arborescence du répertoire du groupe au client, le serveur fait appel lors de l'authentification à la classe XMLgenerator développée en PHP et qui génère un fichier XML respectant le schéma XSD suivant :


attributes
idRep
nom
droits
derniere_modif
proprietaire
id_groupe
RepGroupe
Fichier
attributes
0..∞
attributes
idRep
idRepParent
nom
droits
idProprietaire
dernieremodif
id_groupe
0..∞
SousRep
0..∞
attributes
nom
type
rep_parent
droits
id_proprietaire
taille
id_fic
date_creation
derniere_modif
Fichier
0..∞
0..∞
SousRep
attributes
0..∞
...
0..∞

Il n'existe quasiment pas de différence entre RepGroupe et SousRep puisque tous les deux représentent un répertoire. Ce choix a été retenu parce qu'en XML, on est obligé d'avoir un nœud racine.

Toutes les informations concernant les répertoires et les fichiers sont récupérées de la base de données bien évidement.

Comme vous pouvez le remarquer, le schéma XSD a une structure récursive puisque chaque répertoire peut contenir à son tour des fichiers et des sous répertoires.

Exemple d'un fichier XML décrivant une arborescence

Voici le fichier XML généré par le serveur décrivant l'arborescence du groupe d'id =1.

```
<?xml version="1.0" encoding="UTF-8"?>
<RepGroupe id_rep="1" rep_parent= "0"  nom="RepGrp1"  droits="700" proprietaire="0" derniere_modif="2008-02-18 14:11:13" id_groupe="1" >
    <Fichier id_fic="1" nom="unPremierDoc" type="doc" rep_parent="1" droits="777" id_proprietaire="1" taille="28880"  date_creation="0000-00-00 00:00:00"
derniere_modif="2008-02-17 23:33:23" />
    <SousRep idRep="2" idRepParent="1"  idProprietaire="0" nom="RepGrp2" dernieremodif="2008-05-15 10:41:23" droits="700" id_groupe="2">
    </SousRep>
    <SousRep idRep="3" idRepParent="1"  idProprietaire="6" nom="test" dernieremodif="2008-05-15 10:42:20" droits="777" id_groupe="1">
      <SousRep idRep="28" idRepParent="3"  idProprietaire="6" nom="monRepdemerde" dernieremodif="2008-05-15 12:13:40" droits="777" id_groupe="1">
          <SousRep idRep="29" idRepParent="28"  idProprietaire="6" nom="monRepdemerde2" dernieremodif="2008-05-15 12:14:50" droits="777" id_groupe="1">
          </SousRep>
      </SousRep>
        <SousRep idRep="27" idRepParent="3"  idProprietaire="6" nom="monRep" dernieremodif="2008-05-15 12:13:12" droits="777" id_groupe="1">
        </SousRep>
    </SousRep>
</RepGroupe>
```

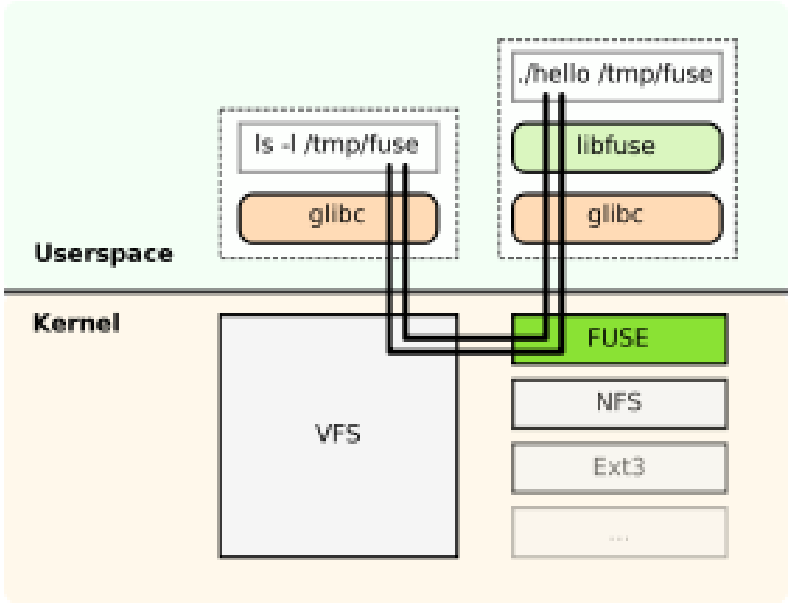
LE CLIENT

Fuse

FUSE (Filesystem in UserSpacE, système de fichiers en espace utilisateur) est un logiciel libre permettant à un utilisateur sans privilèges particuliers d'accéder à un système de fichiers sans qu'il soit nécessaire de modifier les sources du noyau Linux.

Fuse est un module du noyau dont le code s'exécute en espace utilisateur : le module FUSE ne fait que fournir un pont vers l'interface du noyau.

FUSE est particulièrement utile pour écrire un VFS (Virtual File System) : Un système de fichiers traditionnel doit principalement sauvegarder et retrouver des données, alors qu'un système de fichiers virtuel ne stocke pas les données lui-même. Il agit comme une vue ou une traduction d'un système de fichiers existant ou d'un périphérique de stockage.



L'intérêt de Fuse pour notre projet est qu'il sert d'« interface » aux appels systèmes. En effet, Fuse permet, par l'intermédiaire de l'établissement d'un système de fichier virtuel, de capturer les appels systèmes.

Dès lors que cette capture est effectuée, il nous est possible d'envoyer les requêtes voulues au serveur, de récupérer la réponse à cette requête et de la renvoyer à l'utilisateur. Finalement, on « simule » un système de fichier normal de telle manière que l'utilisateur ne se rend pas compte, qu'en réalité, il n'effectue que des requêtes HTTP.

```
public int mknod(String path, int mode, int rdev) throws FuseException
{
      Integer retour = null;
      Spath nouv = new Spath(path);
      try {
          /* touchfile va créer un nouveau fichier sur le serveur par l'intermédiaire
          d'un requête HTTP et nous renvoyer son résultat*/
          retour = ap.touchFile(nouv.nom,(bdd.searchIdfrompath(nouv.chemin)).id(),640
      } catch (Exception e) {
          // Traitement de l'erreur
          e.printStackTrace();
      }

      return (Integer) retour;
}
```

EXEMPLE DE L'INTERCEPTION DE MKNOD AVEC LES ACTIONS POUR TRAITER CET EVENEMENT

Le code des différentes fonctions implémentées est disponible dans le code source easyFS.java fourni avec ce rapport.

Fuse permet la mise en œuvre d'un système de fichiers, défini par un ensemble de callbacks. Pour fuse, les appels ne représentent que des méthodes crée par le programmeur. Pour définir un système de fichiers, il suffit de réaliser une partie ou la totalité des méthodes pour la gestion de ce système de fichiers. Dans notre cas (et comme dans la plupart des cas), nous n'avons pas implémenté toutes les méthodes, seulement les plus usuelles comme :

- getattr

getattr(path : String, FuseGetattrSetter getattrSetter)

Permet de récupérer les attributs du fichiers en fonction du chemin (path). l'attribut getattrsetter permet à la fois d'accéder aux attributs du fichier (taille, type, droits) mais aussi de les positionner.

- readlink

readlink(String path, CharBuffer link)

Fait référence à un lien symbolique. La méthode copie la cible du lien dans la mémoire tampon.

- mknod

mknod(String path, int mode, int rdev)

Crée un fichier ou un périphérique avec les droits spécifies.

- mkdir

mkdir(String path, int mode)

Crée un répertoire avec les droits spécifiés.

- unlink

unlink(String path)

Permet de supprimer le fichier spécifié.

- rmdir

rmdir(String path)

Permet de supprimer le répertoire spécifié.

- rename

rename(String from, String to)

Cette fonction permet, comme sous linux, à la fois de renommer ou de déplacer un fichier ou répertoire. Il suffit donc de vérifier les deux chemins, si ils sont égaux, on renomme sinon on déplace.

- chmod

chmod(String path, int mode)

Permet de modifier les droits d'un fichier ou d'un répertoire.

- utime

utime(String path, int atime, int mtime)

Permet de modifier la date de dernière modification d'un fichier.

- open

open(String path, int flags, FuseOpenSetter openSetter)

Pour ouvrir un fichier. Elle permet d'initialiser, par exemple, des structures de données lors de l'ouverture.

- read

read(String path, Object fh, ByteBuffer buf, long offset)

Permet de lire les données d'un fichier ou un dossier. La fonction retourne le nombre total d'octets lus ou une erreur en cas d'échec.

- write

write(String path, Object fh, boolean isWritepage, ByteBuffer buf, long offset)

Ecrire des données dans un fichier. La fonction doit retourner le nombre total d'octets écrits, ou ou une erreur en cas d'échec.

- statfs

statfs(FuseStatfsSetter statfsSetter)

Récupère ou positionne les informations du système de fichier tel que l'espace totale, l'espace disponible...

- flush

flush(String path, Object fh)

Appelé lors de la fermeture définitive d'un fichier. Permet surtout, de manière optionnelle, de déclarer ou positionner des informations pour traiter cette fermeture.

- release

release(String path, Object fh, int flags)

Appelé lors de la fermeture d'un fichier par un processus. Ne signifie pas forcément que le fichier est totalement fermé car il peut être ouvert dans un autre processus.

- fsync

fsync(String path, Object fh, boolean isDatasync)

Permet la synchronisation des fichiers (refresh).

- getxattr

getxattr(name : String, attr : String, buffer : char, size : natigetxattr(String path, String name, ByteBuffer dst)*

Permet d'obtenir les attribut étendus d'un fichier ou répertoire.

- listxattr

listxattr(String path, XattrLister lister)

Liste les noms de tous les attributs étendus.

Il faut savoir que, selon les versions de Fuse, de nouvelles fonctions ont été mises en place. Cependant, toutes les versions disposes, au moins, des fonctions nécessaires pour la gestion d'un système de fichiers “standards”.

Toutes les fonctions implémentées renvoient un entier qui permet d'indiquer si tout s'est bien passé ou si une erreur est survenue. Ces codes d'erreurs sont indiqués nativement dans la librairie fuse.

```
/*CODES ERREURS DE FUSE */
public static final int EPERM = 1; /* Operation not permitted */
public static final int ENOENT = 2; /* No such file or directory */
public static final int ESRCH = 3; /* No such process */
public static final int EINTR = 4; /* Interrupted system call */
public static final int EIO = 5; /* I/O error */
public static final int ENXIO = 6; /* No such device or address */
public static final int E2BIG = 7; /* Arg list too long */
public static final int ENOEXEC = 8; /* Exec format error */
public static final int EBADF = 9; /* Bad file number */
public static final int ECHILD = 10; /* No child processes */
public static final int EAGAIN = 11; /* Try again */
public static final int ENOMEM = 12; /* Out of memory */
public static final int EACCES = 13; /* Permission denied */
public static final int EFAULT = 14; /* Bad address */
public static final int ENOTBLK = 15; /* Block device required */
public static final int EBUSY = 16; /* Device or resource busy */
public static final int EEXIST = 17; /* File exists */
public static final int EXDEV = 18; /* Cross-device link */
public static final int ENODEV = 19; /* No such device */
public static final int ENOTDIR = 20; /* Not a directory */
public static final int EISDIR = 21; /* Is a directory */
public static final int EINVAL = 22; /* Invalid argument */
public static final int ENFILE = 23; /* File table overflow */
public static final int EMFILE = 24; /* Too many open files */
```

Une partie des codes d'erreurs standards gérés par Fuse

Difficultés rencontrées

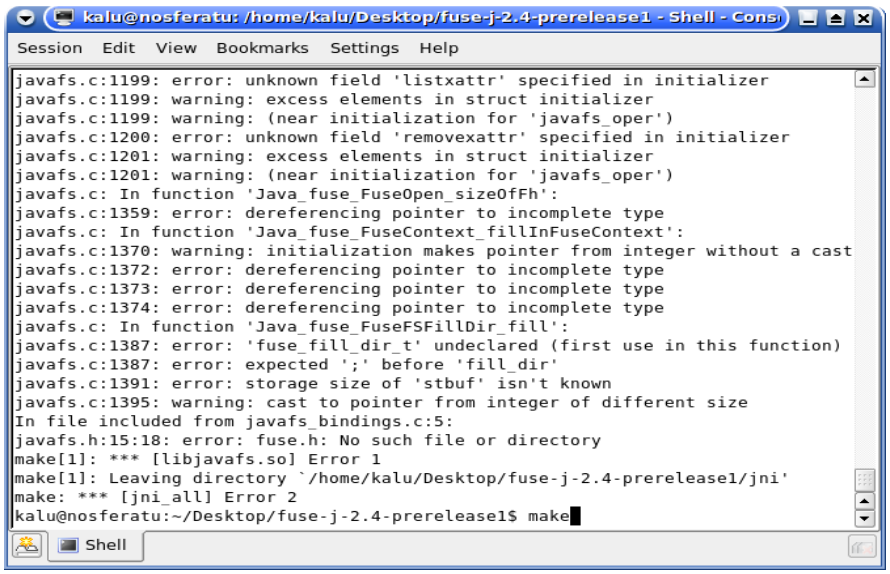
Fuse est un projet d'une grande ampleur, stable et activement supporté. Fuse est un modules intégré au noyau Linux et inclus en standard depuis la version 2.6.

C'est, de par sa popularité, que Fuse a été porté sur de nombreux langages (Python, Perl, Ocaml...) et notamment Java.

C'est parce que le langage Java est bien connu notamment pour sa portabilité que nous l'avons choisi. Ce choix comporte de nombreux avantages mais a aussi été source de plusieurs difficultés.

Compilation et utilisation de fuse-j

Il a tout d'abord fallu générer une archive Java « ***.jar*** », fuse-j étant encore en développement, cela n'a pas été une opération si triviale. De plus, fuse-j n'est fournie avec aucune documentation ni fichier fournissant des indications sur l'installation et l'utilisation.

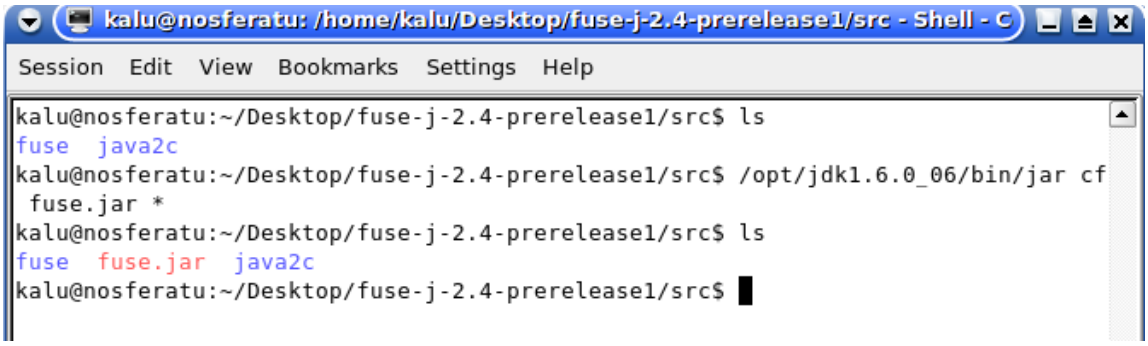


Exemple type des erreurs de compilation de fuse-j

Heureusement, à force de persévérance et en lisant des documentations sur Fuse, il a été possible de savoir quelles librairies sont nécessaires pour la compilation. Il a aussi fallu modifier une fonction incluse dans les sources de fuse-j qui contenait une petite erreur empêchant la compilation.

Une fois la compilation effectué, fuse-j génère un fichier ***libjavafs.so*** (librairie indispensable pour faire fonctionner fuse correctement) ainsi que les fichiers java traduit de fuse.

Une fois ces fichiers générés, il ne suffit plus que de créer l'archive java voulue grâce à la commande ***jar***.



Génération de l'archive java

Création de easyFS.java

Ensuite, nous avons écrit le code afin d'utiliser fuse comme décrit précédemment. Cela a été un peu plus simple car il y a beaucoup d'exemples dans la documentation de Fuse (en C mais facilement adaptable sous Java).

Il a aussi fallu, pour plus de commodités, crée un script shell pour que l'utilisateur puisse l'utiliser comme un logiciel normal et « masqué » qu'en réalité, il exécute du bytecode Java.

Nous avons donc crée le petit programme shell « ***easyfs_mount.sh*** » spécialement :

```
#!/bin/sh

#
# Usage: ./easyfs_mount.sh /mount/point -f [ -s ]


. ./build.conf

LD_LIBRARY_PATH=./jni:$FUSE_HOME/lib $JDK_HOME/bin/java \
   $JAVA_OPTS \
   -classpath ./build:./lib/commons-logging-1.0.4.jar:./lib/fuse.jar:./lib/java-getopt.jar \
   -Dorg.apache.commons.logging.Log=fuse.logging.FuseLog \
   -Dfuse.logging.level=DEBUG \
   -Dcom.sun.management.jmxremote \
  easyFS.EasyFS -s $*
```

script pour l'utilisateur : easyfs_mount.sh

Le fichier « build.conf » auquel il fait référence ne doit contenir que des variables rentrées par l'utilisateur afin d'indiquer où se trouve les binaires java ainsi que les librairies fuse. Par exemple :

```
# répertoire de JDK 1.5 ou plus
JDK_HOME=/opt/jdk1.6.0_06

# répertoire librairie FUSE & headers
FUSE_HOME=/usr/local
```

Tests associés

Connexion de l'utilisateur

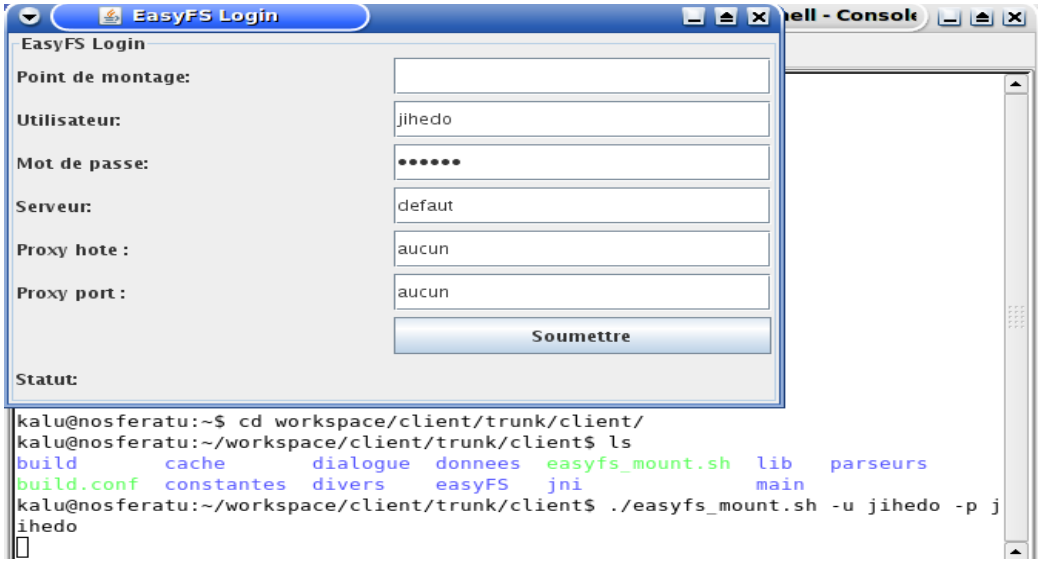
Pour se connecter, l'utilisateur doit exécuter le script « ***easyfs_mount.sh*** », il peut utiliser un mode graphique ou un mode ligne de commande. Si l'utilisateur demande de l'aide *(option -h)*, on lui indique

```
nosferatu:/home/kalu/workspace/client/trunk/client# ./easyfs_mount.sh -h
Easyfs : a client to mount a remote disk space with fuse.

usage: easyfs [options]

options:
    -h, --help                Show this help message (default)
    -d, --directory=<path>    The directory to mount the filesystem
    --                        Stop processing options
    -s, --server=<url>        Specify the URL of the remote server
    -u, --user=<name>         Specify the username for authentication
    -p, --password=<name>     Specify the password for authentication
    -P, --proxy=<url>         Specify the proxy to use
    -c, --proxyport=<int>     Specify the proxy port to use
    -V, --version             Show the version and exit
    -g, --graphic             Graphic version for the command options

Use example (text mode) :
./easyfs_mount.sh -u name -p password -d /mnt/directory -shttp://server.com/serv
er.php -P195.83.8.14 -c985
```

Aide pour l'utilisation du programme client fuse

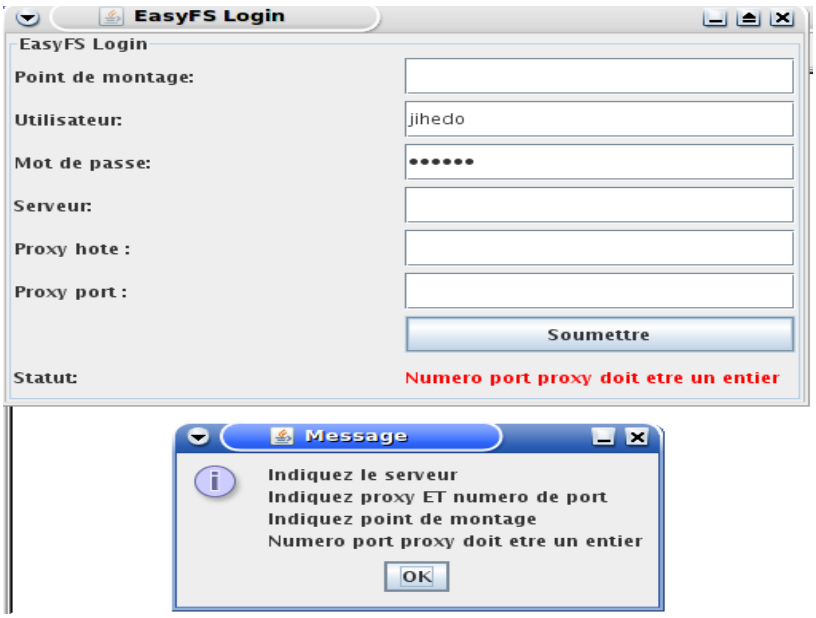
ses possibilités :

L'option graphique apparaît soit si l'utilisateur le demande soit si l'utilisateur a oublié un paramètre en ligne de commande. Pour cela, nous utilisons une méthode intrusive. On essaie de lancer une applet graphique Java et si cela échoue, on indique à l'utilisateur qu'il faut qu'il fournisse l'ensemble des paramètres en ligne de commande. De cette manière, nous gérons le cas où l'utilisateur n'a pas de serveur graphique lancé. L'interface graphique récupère les paramètres déjà saisis par l'utilisateur :



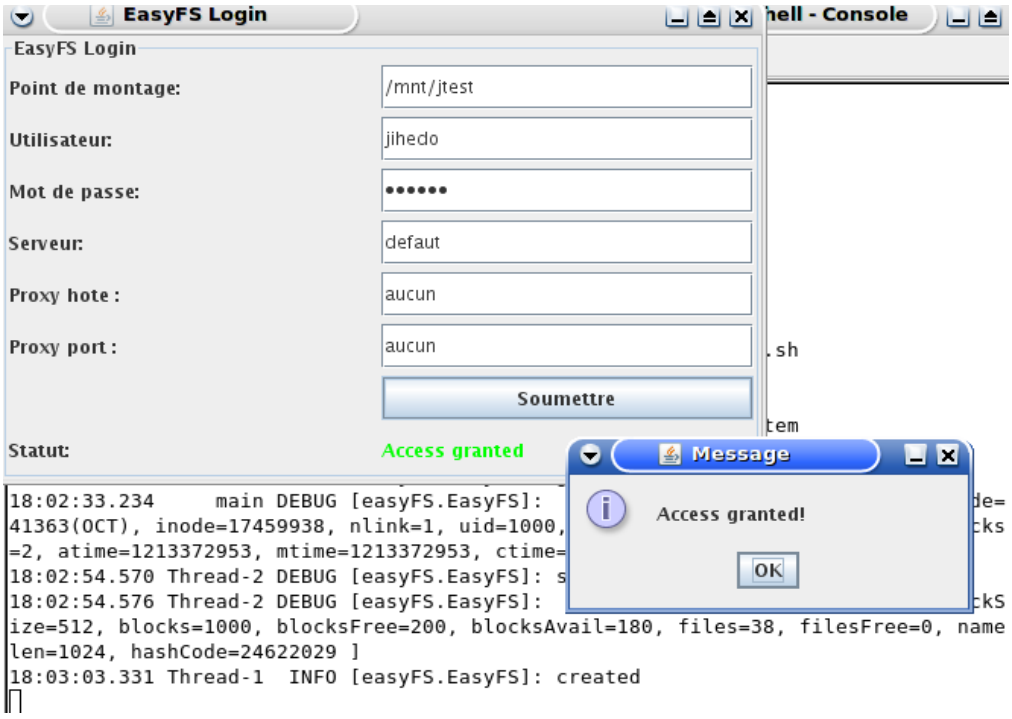
Interface graphique pour Fuse

Nous avons donc testé si les paramètres étaient bien récupérés à la fois en mode graphique et en mode ligne de commande. Nous avons ensuite testé si l'interface graphique affichée bien les erreurs de saisie à l'utilisateur :



Affichage des erreurs

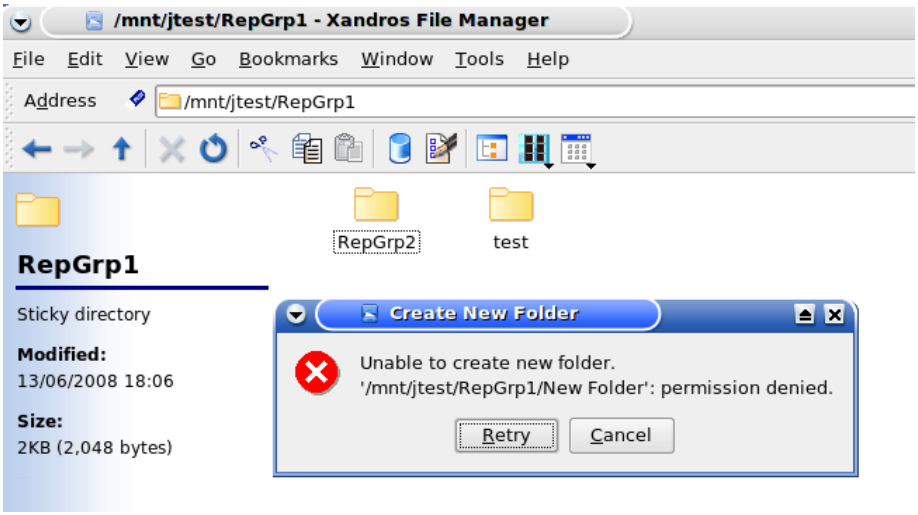
Une fois le système de fichiers montés, nous affichons un message pour indiquer à l'utilisateur que tout s'est bien déroulé.



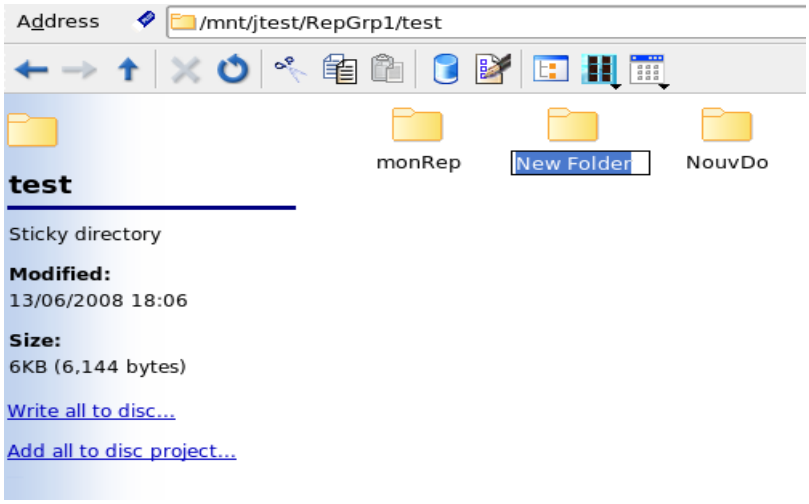
Connexion acceptée

A partir de ce moment là, il est possible de manipuler le système de fichiers monté sur « ***/mnt/jtest*** » soit en ligne de commande soit en mode graphique. Pour des raisons de lisibilité, nous avons majoritairement choisi de faire des captures d'écran du navigateur graphique.

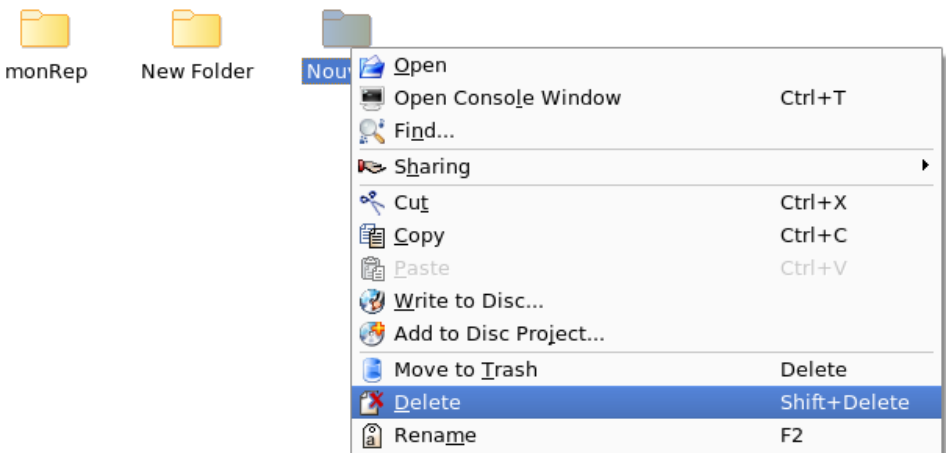
Droits insuffisants



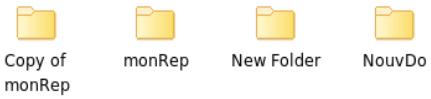
Création de répertoire

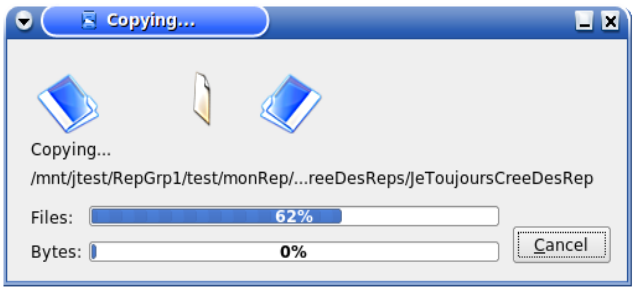


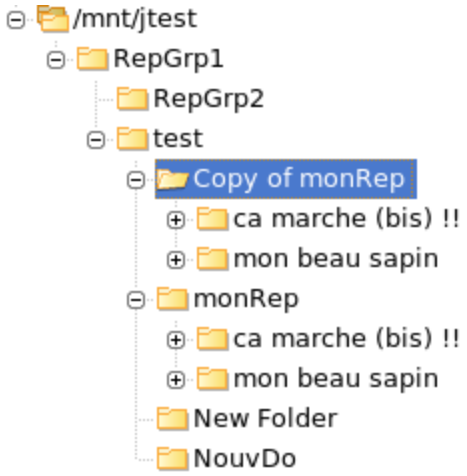
Suppression de répertoire



Création récursive de répertoire

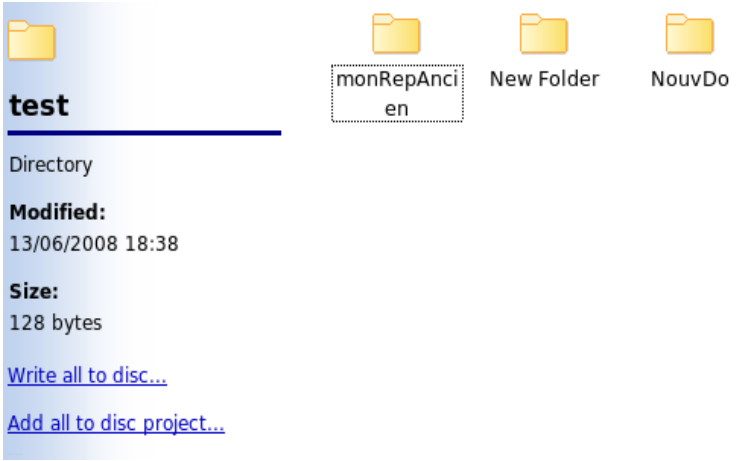


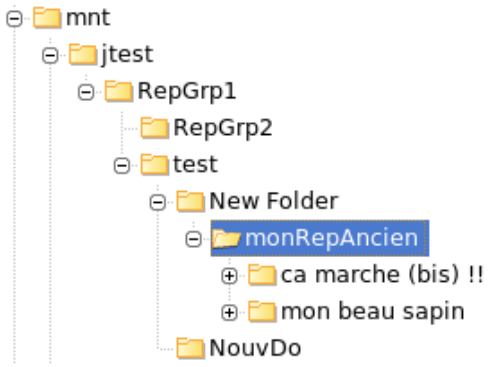




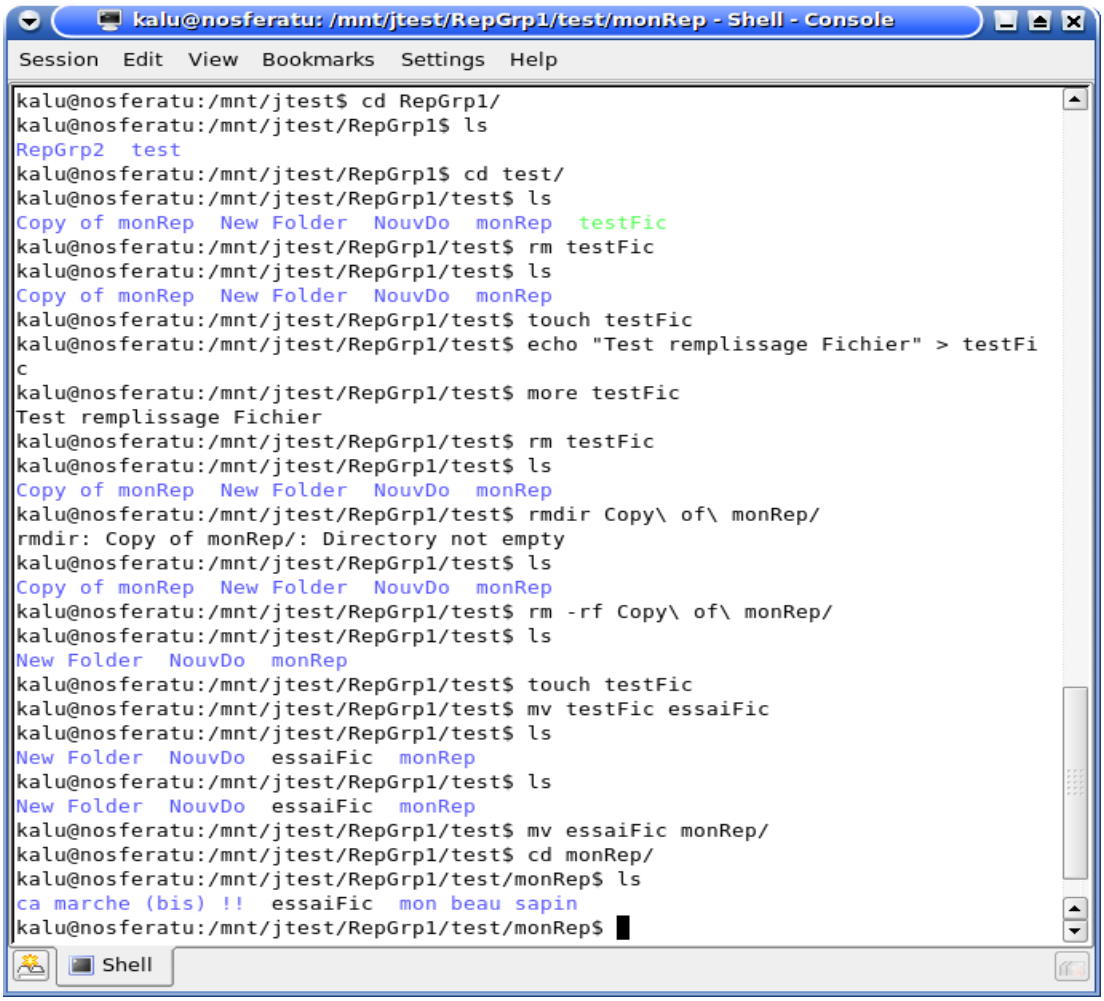
Le répertoire « ***copy of monRep*** » contient toute l'arborescence de monRep une fois recopié.

Renommage et déplacement de répertoire





Manipulation de fichiers et ligne de commande



Nous avons dans la mesure du possible testé l'ensemble des fonctions codées à l'aide de la librairie Fuse. Il arrive, parfois, surtout lors de la copie d'un répertoire contenant des sous-répertoires, qu'il y est une erreur. Nous avons remarqué que cela arrivait à cause de mini coupures réseaux. Il suffit donc de reprendre la copie (si navigateur graphique) ou de la relancer (si ligne de commande).

Les parseurs XML

Les méthodes SAX et DOM adoptent chacune une stratégie très différente pour analyser la syntaxe des documents XML, elles s'utilisent donc dans des contextes différents. DOM charge l'intégralité d'un document XML dans une structure de données, qui peut alors être manipulée puis reconvertie en XML. Cependant pour cela il faut que la taille de la structure représentant le document XML ne soit pas supérieure (ou pas trop) à ce que peut contenir la mémoire vive. La méthode SAX apporte alors une alternative dans les cas de figure où les documents XML sont de taille très importante (on parle alors de mise à l'échelle, en anglais scalability).

L'intérêt majeur de DOM est donc la possibilité qu'il offre d'aller et venir à votre gré dans l'arborescence, son inconvénient majeur reste la lourdeur du traitement. En effet, SAX étant événementiel, son traitement se fait au fil du flux entrant.

Pour les raisons cités ci-dessus, on a opté pour SAX afin de parser notre fichier XML décrivant l'arborescence du répertoire du groupe, les réponses envoyés par le serveur au niveau client et les requêtes au niveau serveur.

Le parseur de l'arborescence

La classe « ParseurArborescence » du paquetage « parseurs » prend en entrée le fichier XML envoyé par le serveur et renvoi un objet de type répertoire décrit ci-dessous. Comme la structure du fichier XML est récursive, l'opération de parsage est elle aussi récursive.

L'objet répertoire est lui aussi une structure de donnée récursive.

L'objet Repertoire

Un répertoire est composé de plusieurs champs :

Repertoire	
public int	idRep : L'identifiant du répertoire.
public int	idRepParent : L'identifiant du répertoire parent.
public int	idProprietaire : L'identifiant du propriétaire.
public String	nom : Le nom du répertoire.
public String	derniereModif : Date de dernière modification.
public int	droits : Les mêmes utilisés dans linux.
public int	id_groupe : L'identifiant du groupe.

public	List<Repertoire> : La liste de sous répertoires
public	List<Fichier> : La liste des fichiers contenus dans le répertoire

L'objet Fichier

Un fichier est composé de plusieurs champs :

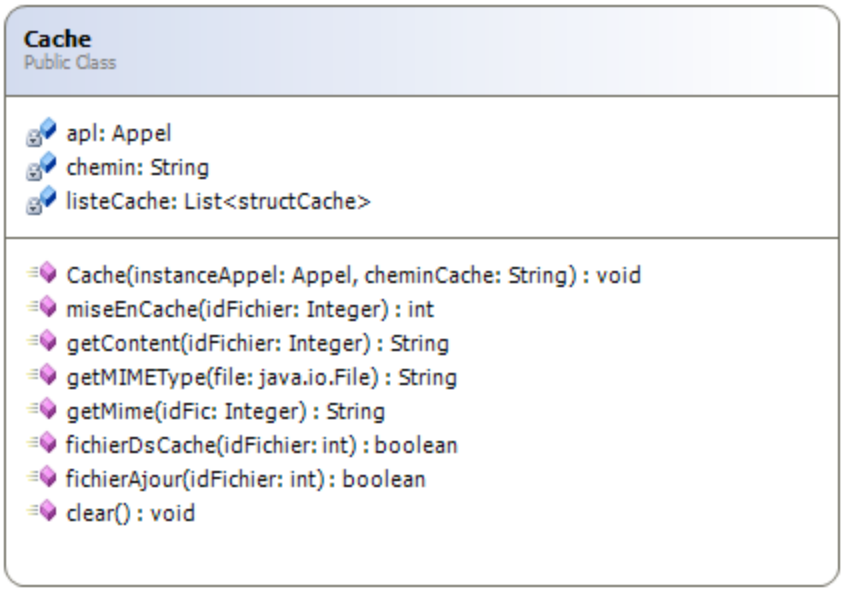
Fichier	
public int	id_fic : L'identifiant du fichier.
public int	rep_parent : L'identifiant du répertoire parent.
public int	id_proprietaire : Identifiant du propriétaire.
public float	taille : La taille du fichier.
public String	nom : Le nom du fichier.
public String	type : Le type du fichier.
public String	derniere_modif : Date de denière modification.
public String	date_creation : Date de création.
public int	droits : Les mêmes utilisés dans linux.

Le cache

Le cache est un dispositif mémoire à accès rapide utilisé pour accélérer les transferts d'information de manière temporaire ou permanente. Tout ordinateur dispose de mémoire cache ou de disques cache qui contiendront des données ou des instructions fréquemment utilisées. Dans le contexte Internet, la mémoire cache allouée à un navigateur contiendra les pages Web récemment chargées.

Ce même principe a été appliqué pour le projet, dès que l'utilisateur demande d'ouvrir un fichier (en mode console ou graphique), ce dernier est téléchargé dans un répertoire temporaire servant de cache afin d'accélérer la lecture et l'écriture. Toute modification est percutée par la suite sur le serveur.

Diagramme de Classe



Les attributs de la classe Cache

- Apl : une instance de la classe Appel qui nous permet de récupérer toutes les informations de connexion (l'identifiant de la session, login et mot de passe de l'utilisateur…).
- Chemin : le chemin du répertoire nous servant de cache.
- listeCache : une liste qui nous permet de savoir la liste des fichiers téléchargés en local et leur date de dernière modification.

Les méthodes de la classe Cache

- MiseEnCache permet de télécharger un fichier, identifié par son id et appartenant au groupe de l'utilisateur, en local.
- getContent permet de récupérer le contenu d'un fichier identifié par son id.
- getMIMEType retourne le type d'un objet de type java.io.File.
- getMime retourne le type d'un fichier identifié par son id.
- fichierDsCache retourne vrai si le fichier recherché existe bien dans le cache.
- fichierAjour retourne vrai quand le fichier en cache est à jour par rapport au fichier qui se trouve sur le serveur.
- Clear permet de vider le cache.

Accès concurrents

Comme dans tout système partagé, il existe des risques d'accès concurrents qui peuvent nuire à l'intégrité des données. Un cas très fréquent : la lecture d'un fichier en cours de modification (écriture).

Pour notre projet, on a recensé les cas d'accès concurrents suivants :

- La suppression d'un fichier alors qu'il est en cours de téléchargement.
- La suppression ou la lecture d'un fichier lors de sa mise à jour.
- La suppression ou la lecture d'un fichier en cours de création (envoi du fichier vers le serveur pour la première fois).

Solution :

Une solution simple a été mise en œuvre pour répondre à ce genre de problème et qui consiste à la création d'une nouvelle table appelée « unlocked », cette table contient les identifiants des fichiers qui ne sont pas en cours de création, de téléchargement ou de création.

Algorithme de gestion des accès concurrents

Lors de la suppression, la mise à jour ou encore la création d'un fichier :

1/ On commence par vérifier que l'id du fichier existe dans la table unlocked.

2/ Supprimer l'id de la table unlocked, ce qui revient à bloquer l'accès au fichier.

3/ à la fin de la réception (download), le client envoi un accusé de réception au serveur pour débloquer le fichier et rajouter son id dans la table unlocked.

4/ à la fin de l'envoi d'un fichier vers le serveur (upload), le serveur rajoute l'id du fichier dans la table unlocked.

Remarque : Lors de la suppression d'un répertoire, tous les fichiers non utilisées seront supprimés et un message d'erreur sera renvoyé au client si jamais un fichier contenu dans ce répertoire est en cours d'utilisation ce qui ressemble à la suppression sous les systèmes de fichiers linux et windows.

LE SERVEUR

Le serveur a deux rôles principaux :

- Offrir une zone d'administration, qui permet aux **administrateurs** de créer, modifier et supprimer des groupes et des utilisateurs, et aux **utilisateurs** de modifier leurs informations personnelles (telles que leur groupe).

- Manipuler les fichiers ainsi que les informations les concernant dans la base de données du serveur, selon les ordres donnés par le client via le protocole de communication.

Nous avons décidé de coder le serveur en PHP pour trois principales raisons :

- Son interfaçage avec les bases de données MySQL et PostgreSQL
- Ses fonctions de gestion des sessions
- Ses outils intégrés de parage de flux XML

Inconvénients : La communication entre client et serveur se fait par http.

Zone d'administration

Cette zone comprends un formulaire d'identification, qui permet d'accéder aux consoles de gestion de compte utilisateur, et d'administration. Les pages d'administrations sont invisibles si la session courante ne contient pas les droits administrateur.

Une fois identifié, l'utilisateur peut visualiser ses informations personnelles, et les modifier :

Les administrateurs ont accès à une console de gestion des groupes et utilisateurs.

Ils peuvent ajouter/supprimer un groupe d'utilisateurs, ce qui équivaut à créer un espace de stockage virtuel dont on peut spécifier la taille. Ils ont également le droit d'ajouter/supprimer des utilisateurs et des administrateurs dans la base de données.

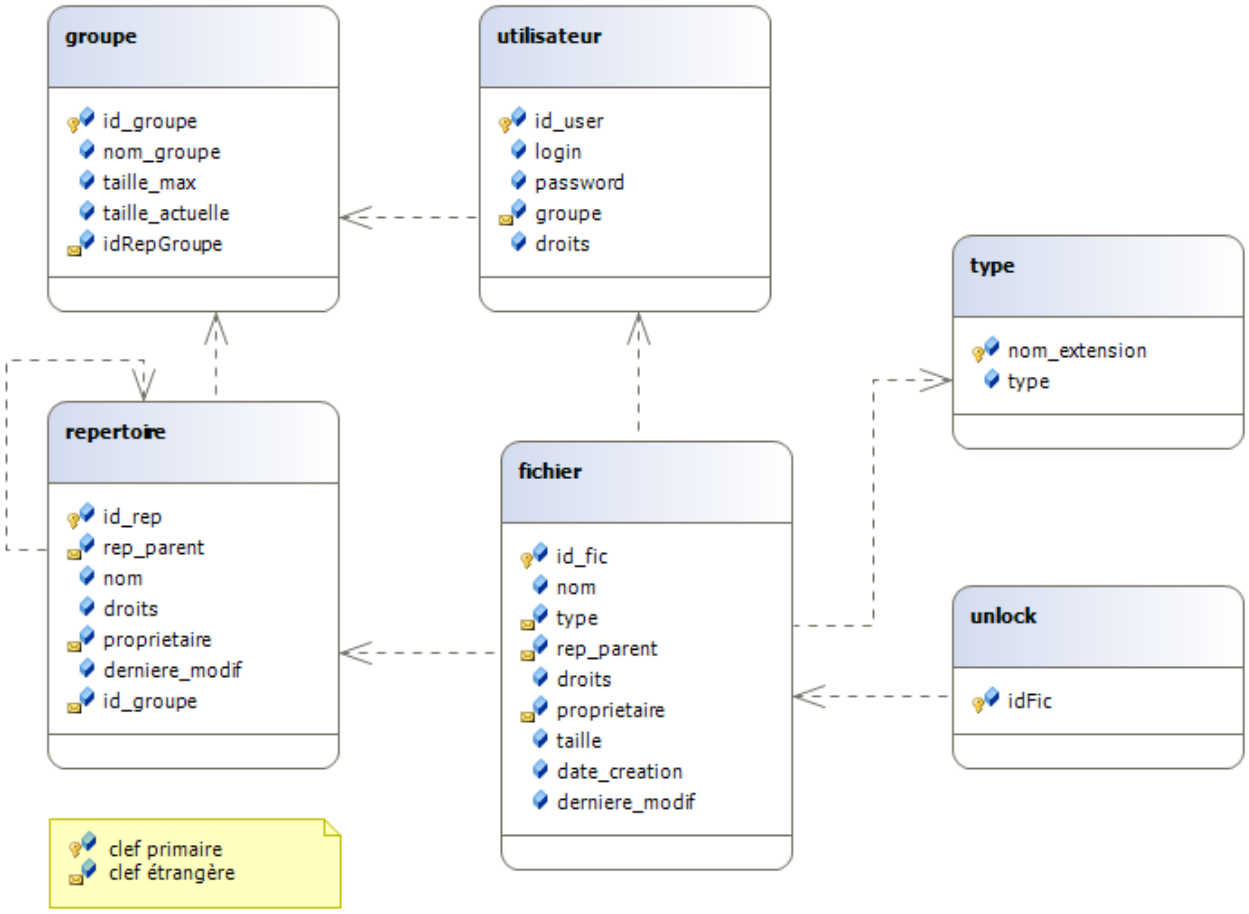
Une fois les utilisateurs créés, ils sont immédiatement actifs dans la base de données. Ils peuvent donc se connecter via le client pour utiliser leur espace virtuel.

La création de groupe crée un dossier de stockage sur le serveur, dans lequel seront stockés les fichiers appartenant au groupe. La gestion de l'arborescence des fichiers et de leurs droits est faite par la base de données.

Traitement des fichiers

La base de données

Le fonctionnement du serveur repose sur une base de données gérant les groupes et les comptes utilisateurs, ainsi que toutes les informations sur les fichiers et l'arborescence des dossiers.



Les utilisateurs et les groupes sont gérés via l'interface d'administration.

La taille maximale d'un groupe limite le nombre de fichiers que l'on peut uploader vers ce groupe (cf. opérations de traitement de fichiers).

Un utilisateur est caractérisé par ses login et password (crypté par un hash md5), et appartient à un groupe, modifiable depuis l'interface d'administration. Il dispose également de droits

(administrateur/utilisateur) lui permettant d'accéder aux différentes consoles d'administration (voir plus haut).

Les fichiers sont uploadés sur le serveur sous une forme très simple (le nom du fichier sur le serveur est son identifiant unique). Toutes les propriétés des fichiers, ainsi que l'arborescence des répertoires, sont gérées par la base de données.

LA GESTION DES PROPRIETES DES FICHIERS ET DOSSIERS DIRECTEMENT PAR LA BASE DE DONNEES PERMET D'AUGMENTER LA RAPIDITE DES OPERATIONS AUTRES QUE CREATION/MODIFICATION DE FICHIERS.

Le script générant l'architecture des dossiers d'un groupe à partir de la base de données agit en partant du répertoire associé au groupe (RepGroupe), puis en recherchant, pour chaque répertoire, les dossiers et fichiers ayant pour 'rep_parent' ce répertoire. Il construit ainsi l'arborescence du groupe.

Outre le nom, les répertoires et les fichiers possèdent des **droits** (équivalents au chmod), un propriétaire (créateur), et un groupe, qui seront nécessaires pour autoriser un utilisateur à les manipuler.

Les fichiers possèdent un **type**, qui doit figurer parmi ceux décrits dans la table type. Dissocier l'extension du nom du fichier sert à envoyer le bon mime/type au client en même temps que le fichier. Cela permettra dans une prochaine amélioration de pouvoir filtrer les fichiers à l'upload, voire proposer différentes fonctions en fonction du type du fichier.

Enfin, les tables permissions et unlock servent à protéger l'accès aux fichiers, respectivement en cours d'upload/download et de modification (voir Accès concurrents).

Les opérations de traitement de fichiers/dossiers

Lorsque l'utilisateur lance le client, il doit s'identifier grâce à ses login et mot de passe. Une requête de création de session est alors envoyée au serveur. Celui-ci récupère les informations concernant l'utilisateur dans la base de données si celles-ci existent, et lance une session contenant id utilisateur, groupe et droits.

Si l'utilisateur n'existe pas ou que le mot de passe est incorrect, le serveur renvoie un code d'erreur (cf. Protocole).

Upload/download de fichier

Les opérations d'upload/ de download se font en deux temps, dans un premier temps, le client envoie une requête au serveur, contenant l'identifiant du fichier à envoyer/télécharger.

A la réception de cette requête, le serveur teste plusieurs choses (l'existence du fichier, les droits de l'utilisateur sur le fichier). Une fois cette étape passée, le serveur bloque le fichier (s'il est déjà bloqué, une erreur « fichier occupé » est propagé jusqu'au client) et renvoie au client une url pour qu'il puisse exécuter son action.

Le fait de renvoyer une url et non un simple accord/désaccord, permet d'être plus souple au niveau serveur, il est en effet ainsi facile d'utiliser plusieurs serveurs distants pour le stockage des fichiers.

Une fois cette url récupérée, le client l'appelle (avec les bons paramètres, c'est à dire comme toujours, la session, mais aussi l'identifiant du fichier). Ce nouveau script au niveau du serveur, va revérifier si l'utilisateur à le droit d'envoyer, de télécharger ce fichier, le cas échéant, il effectue l'opération, et libère le fichier.

Concernant l'upload, le contenu du fichier est envoyé par le client dans une entête HTTP spécifique (tel qu'on l'enverrait via un formulaire d'upload purement web), ce contenu, et diverses informations sont stockées par le serveur (le contenu est stocké dans un dossier temporaire) le script d'upload vérifie donc ces informations, met à jour la base de données en conséquence (concerne surtout la taille du fichier envoyé) et si tout c'est bien passé, déplace le fichier de son emplacement temporaire jusque dans le répertoire de stockage de son groupe. Celà fait, le serveur renvoie son acquittement au client.

Dans le cas du téléchargement de fichier, il n'y a pas d'acquittement en cas de réussite, en effet, on ne peut pas envoyer en même temps, le fichier et la réponse du serveur. Le téléchargement se passe aussi par une modification d'une entête HTTP, ce qui permet de cacher au client l'emplacement physique du fichier, ce qui pourrait engendrer certains problèmes de sécurité (mais heureusement, le dossier où sont stockés les fichiers n'est accessible que par Apache, mais on ne sait jamais)

Modification de fichier

La modification de fichier marche de la même manière que l'upload de fichier, ça se passe en deux temps, dans un premier temps, le serveur accepte (ou non) la modification du fichier, en fonction de l'utilisateur, de ses droits, et de l'occupation du fichier, le serveur bloque alors le fichier, et renvoie une url d'upload au client pour qu'il envoie le nouveau fichier et le débloque.

Modification des propriétés d'un fichier ou d'un dossier

Ces modifications se font uniquement dans la base de données. Elles s'effectuent en plusieurs étapes :

1/ Réception de la requête du client sous forme de flux XML

2/ Parsage des informations à l'aide d'objets XML (gérés en PHP)

3/ Vérification des droits de l'utilisateur par rapport au fichier, ainsi que de la disponibilité du fichier (cf. Accès concurrents)

4/ Modification des propriétés dans la base de données (exemple : nom, dossier parent, droits…)

5/ Renvoi d'un code succès, ou du code d'erreur correspondant (mauvais droits, le fichier n'existe plus…)

Suppression d'un dossier

Avant la suppression d'un dossier, outre la vérification des droits, on effectue une vérification de tous les fichiers contenus dans ce dossier, pour s'assurer qu'ils ne sont pas interdits en écriture ou bloqués (dans le cas d'un accès concurrent). Si c'est le cas, on renvoie un message d'erreur. Sinon, on supprime récursivement tous les fichiers et dossiers contenus dans le répertoire.

Test du serveur et exemples

Nous avons programmé une interface de test très basique pour le serveur, simulant l'envoi de requêtes XML par le client. Cela nous a permis d'effectuer les tests unitaires côté serveur.

Authentification

On envoie une demande d'authentification contenant le login et le hash du mot de passe.

```
<requete action="authentification">
        <param name="login" value="nessie" />
        <param name="password" value="bb218853cb7c48592d53478b8f7018a1" />

</requete>
```

La requête aboutit, on reçoit donc l'id de la session, ainsi que l'adresse du fichier contenant l'architecture de l'espace virtuel.

```
<reponse code_retour="0" >
        <param name="id_session" value="a508e4c34639d62685a59da65fe3e477" />
        <param name="url_xml" value="http://fuse.credi-
ong.org/serveur/temp/a508e4c34639d62685a59da65fe3e477.xml" />
</reponse>
```

Création d'un dossier

Une fois identifié, on peut créer un dossier dans notre espace. On souhaite créer le dossier 'nouveaurep' à la racine :

```
<requete action="creer_rep">
        <param name="nom_rep" value="nouveaurep" />
        <param name="emplacement" value="1" />
        <param name="droits" value="777" />
</requete>
```

Or la racine de ce groupe n'a pas les droits suffisants pour permettre la creation de repertoire. Le code renvoyé est donc 13 (permission denied).

```
<reponse code_retour="13" />
```

Modification d'un dossier

On souhaite modifier le nom du dossier 'mon beau sapin' dans le groupe 1, qui porte l'id 29 :

```
<requete action="renommer_rep">
        <param name="id_rep" value="29" />
        <param name="nouveau_nom" value="noel" />
</requete>
```

Le renommage fonctionne, le serveur renvoie donc le code OK :

```
<reponse code_retour="0" />
```

BILAN

Bilan du projet

Après plus de 4 mois à travailler sur ce projet, l'heure du bilan a sonné, les questions se posent, avons nous respecté notre cahier des charges ? Avons-nous appris beaucoup de chose ? Nous sommes nous bien débrouillés ? Est-ce que nous aurions pu être plus efficaces ?

Les réponses sont diverses, nous n'avons pas respecté toutes les demandes du cahier des charges (voir pistes d'améliorations), en cause, nous avons buté sur des problèmes non prévus (installation/utilisation de fuse). Oui, nous avons appris beaucoup de choses (fuse, création de protocole, gestion client/serveur, protocole HTTP) et améliorer certaines de nos connaissances (java, XML, PHP, Bases de données, gestion d'Eclipse). Sur ces points, ce projet tuteuré aura été très formateur, de même dans les domaines de gestion de projet/conduite de réunion/travail en groupe nous avons appris un certain nombre de choses, et amélioré d'autres.

Mais nous sommes nous bien débrouillés pour autant ? Sans complexes je dirais oui, même si notre projet n'est pas parfait, il marche. Même s'il ne répond pas à toutes les exigences du cahier des charges, le noyau en est solide et est facilement améliorable. Mais (il y a toujours un mais) nous aurions pu être plus efficaces, nous avons perdu beaucoup de temps sur des broutilles, perdu du temps dans des fausses pistes (mais c'est justement ça qui est le plus formateur).

Dans sa globalité, ce projet nous a été très formateur, et nous sommes bien contents qu'il fonctionne!

Pistes d'améliorations

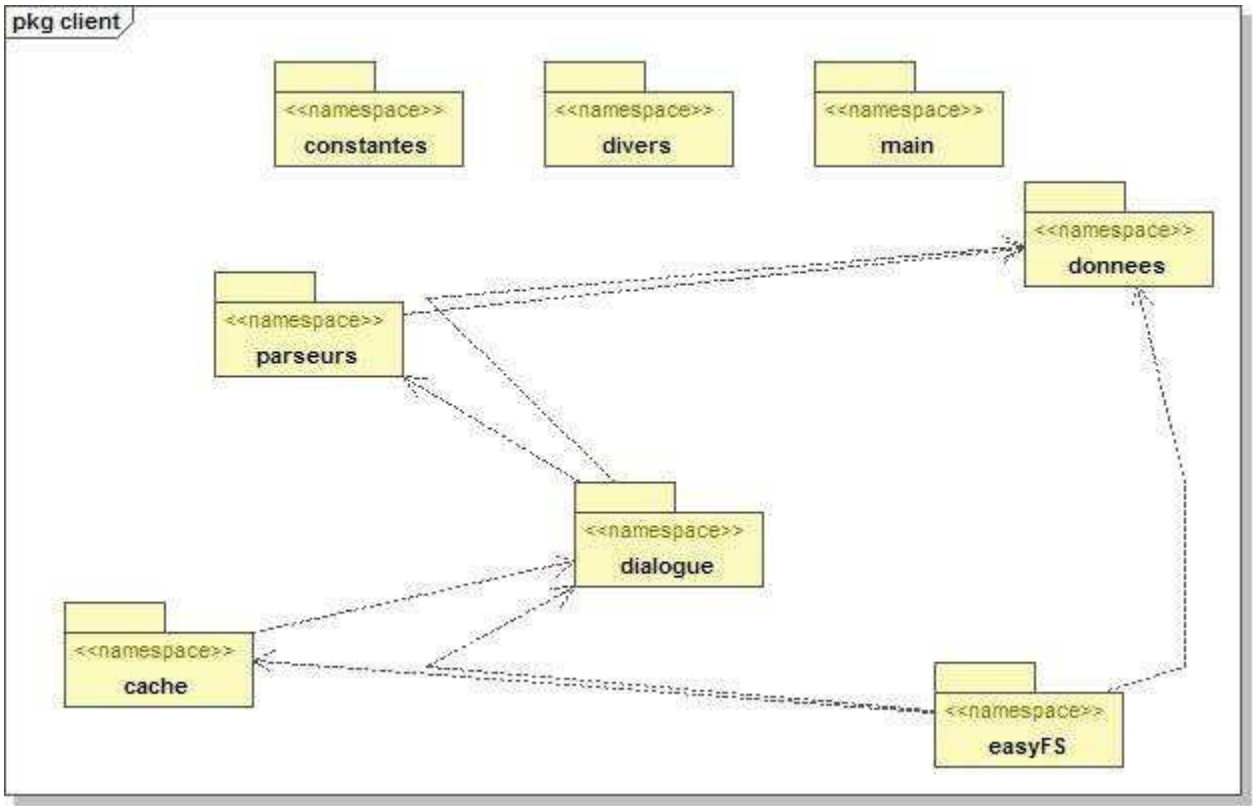
Notre projet est perfectible, il y a plusieurs choses que nous n'avons pas eu le temps d'implémenter (bien qu'ayant déjà une idée de la manière de faire), par exemple, la possibilité d'avoir de gros fichiers, cela est possible en découpant les fichiers sur le serveur (en fichiers de 1 ou 2 Mo par exemple), l'upload et le download se feraient alors sur ces parties, ce qui permettrait de limiter le trafic, en cas de modification d'un fichier par exemple (upload uniquement des parties modifiées, tracées par md5 sans doute), mais compliquerait les accès concurrents.

Deuxième amélioration, sécuriser un peu le transfert, en se basant sur HTTPS par exemple (ajout de certificats au niveau serveur, quelques modifications à faire au niveau de la classe HTTP_Requete au niveau client).

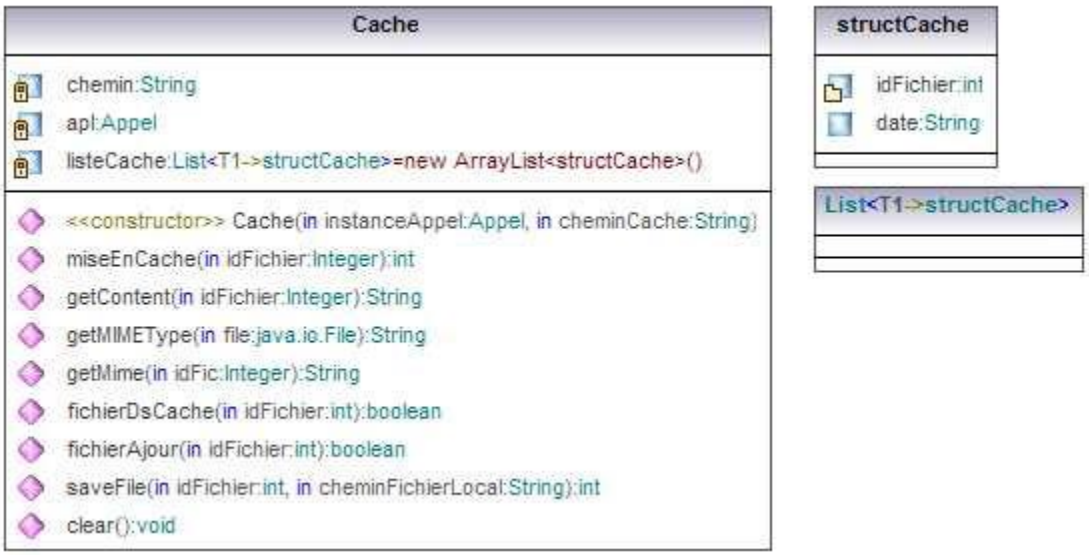
Il serait aussi possible (mais plus compliqué) de faire un mécanisme d'envoi uniquement des modifications apportées à un fichier, ce qui permettrait de limiter le trafic, et d'améliorer la rapidité de l'application. Enfin, nous n'avons pas utilisé toutes les potentialités de fuse, par exemple la possibilité de faire des liens virtuels.

ANNEXES

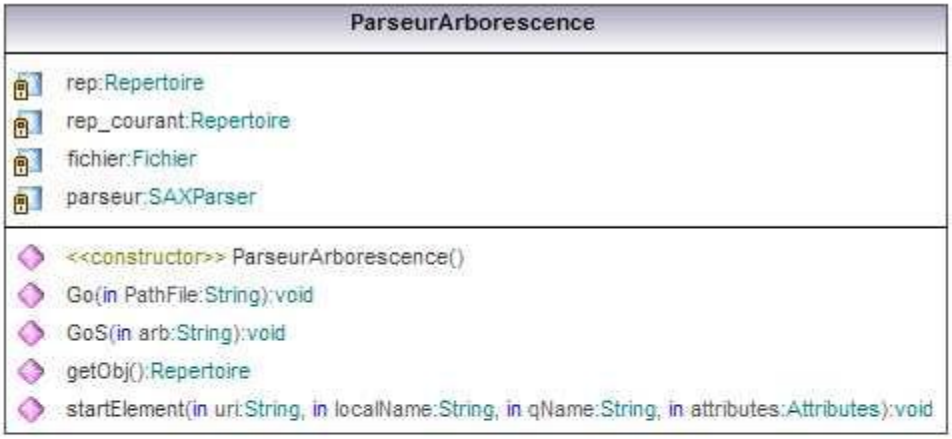
DEPENDANCES ENTRE PACKAGES



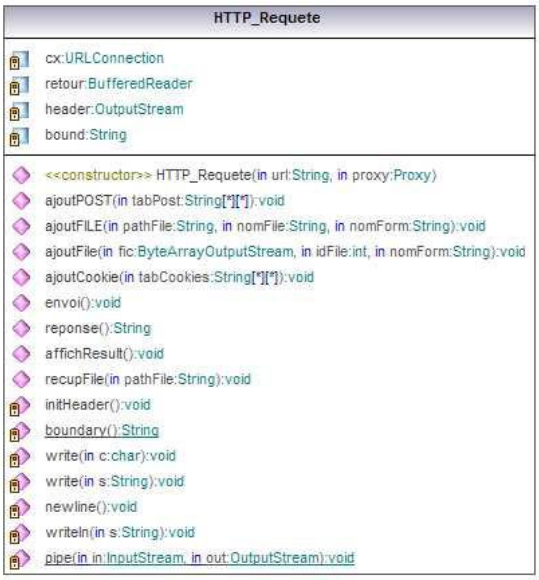
CACHE



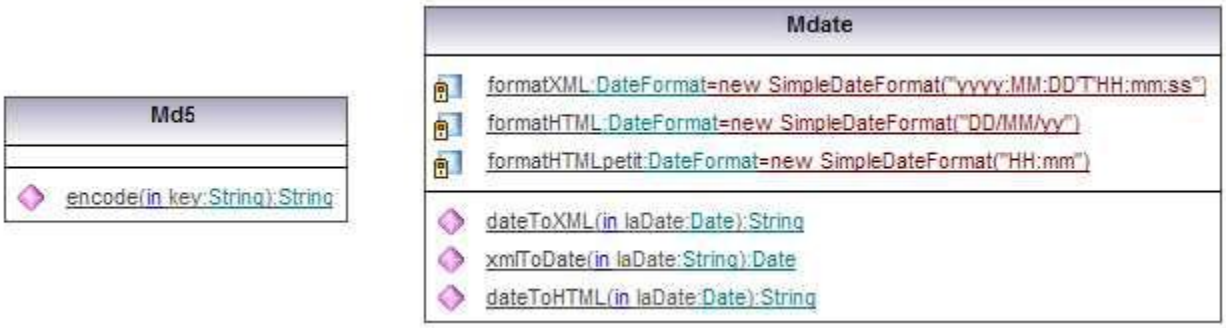
PARSEURS



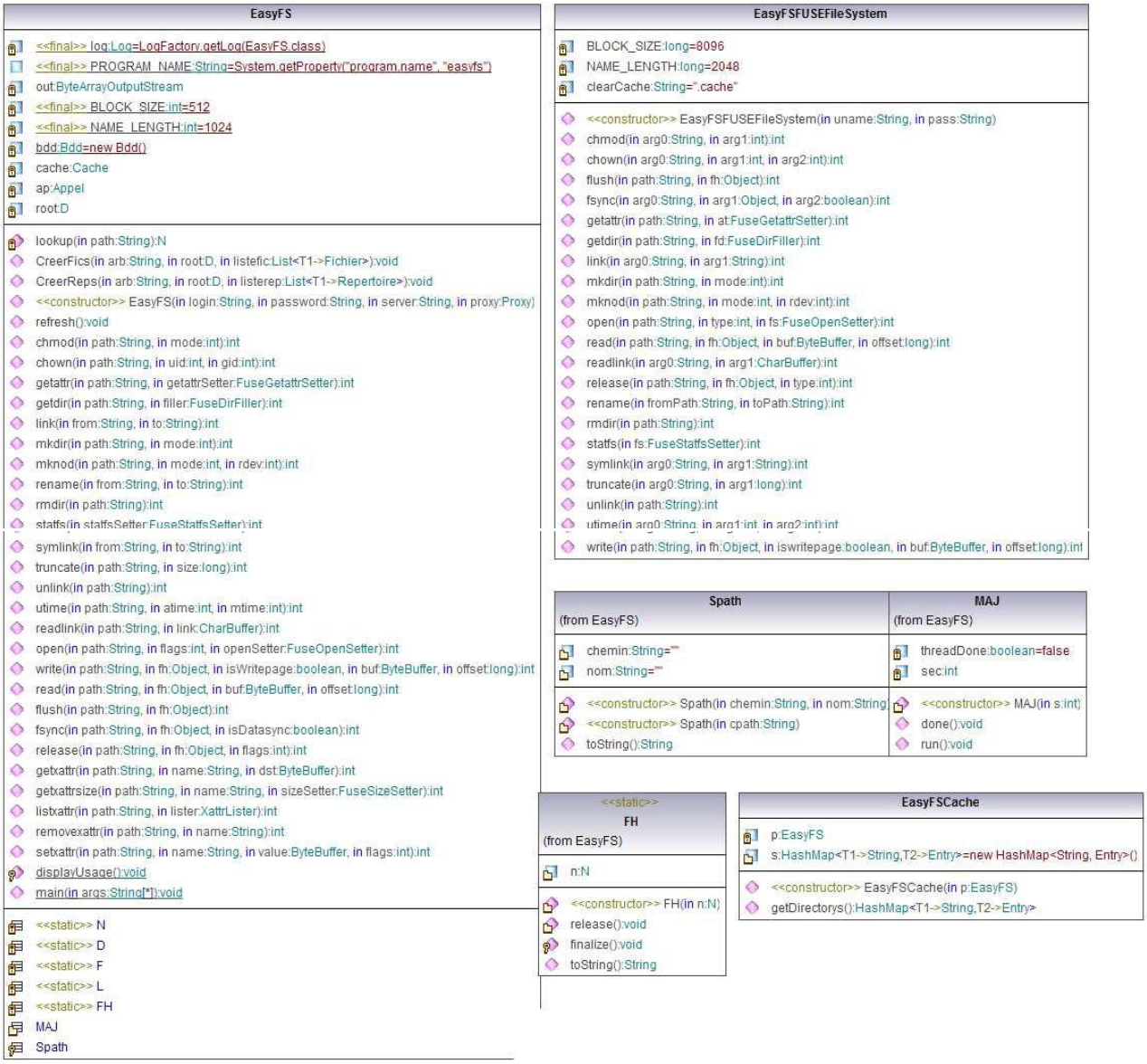
DIALOGUE



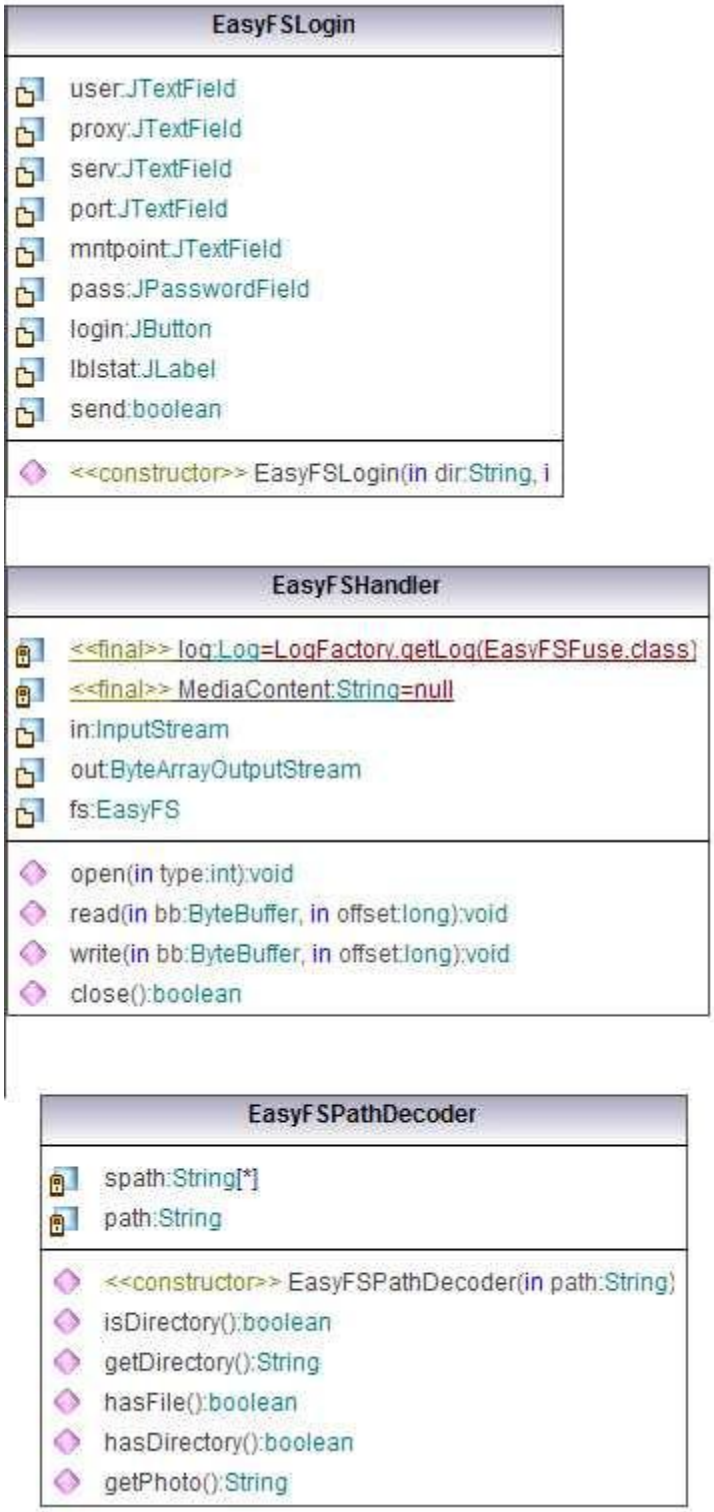
DIVERS



EASYFS



EASYFS(2)


EasyFSLogin
user:JTextField
proxy:JTextField
serv:JTextField
port:JTextField
mntpoint:JTextField
pass:JPasswordField
login:JButton
lblstat:JLabel
send:boolean
<<constructor>> EasyFSLogin(in dir:String, i
EasyFSHandler
<<final>> log:Log=LogFactory.getLog(EasyFSFuse.class)
<<final>> MediaContent:String=null
in:InputStream
out:ByteArrayOutputStream
fs:EasyFS
open(in type:int):void
read(in bb:ByteBuffer, in offset:long):void
write(in bb:ByteBuffer, in offset:long):void
close():boolean
EasyFSPathDecoder
spath:String[*]
path:String
<<constructor>> EasyFSPathDecoder(in path:String)
isDirectory():boolean
getDirectory():String
hasFile():boolean
hasDirectory():boolean
getPhoto():String