Présentation de Jakarta Common VFS

Common Virtual File System

Flavien BACH

Ingénieurs 2000 – IR3

Novembre

2003

Plan de la présentation

- Présentation de l'API
- Mise en œuvre
- Principe de fonctionnement général
- Techniques avancées
  - Recherche de fichiers
  - Mise en œuvre de l'API dans un modèle MVC
- Limitations
- Solutions alternatives
- Conclusion

Présentation de l'API

- API développée dans le contexte du projet Jakarta Commons
  - Ensemble de composants java réutilisables (net, logging, httpclient, …)
  - Inclusion de VFS à la branche SandBox de Commons
  - Apache Software Licence Version 1.1
- Permet de manipuler les fichiers de manière indépendante du système de fichier sous-jacent.

Présentation de l'API

- Support de nombreux systèmes de fichiers
  - Fichiers locaux, FTP, SFTP, HTTP, WebDav, CIFS, ZIP, JAR …
  - Interface permettant de rajouter d'autres systèmes de fichiers
- Mise en cache dans la JVM des informations concernant les fichiers (locaux, ou non)
- Possibilité de créer des systèmes de fichiers logiques avec des jointures sur plusieurs systèmes de fichiers réels.
- Gestion d'évènements concernant les fichiers.

Installation de l'API

- Téléchargement de la dernière archive sur : http://cvs.apache.org/builds/jakarta-commons/nightly/commons-vfs/
- Mettre le fichier jar dans le CLASSPATH
- Dépendance avec d'autres API :

API	Requise pour
Commons Logging	Obligatoire
Commons Net	FTP
Commons HTTPClient	HTTP
Slide	WebDav
JCIFS	CIFS
JSch	SFTP

Principe de fonctionnement général

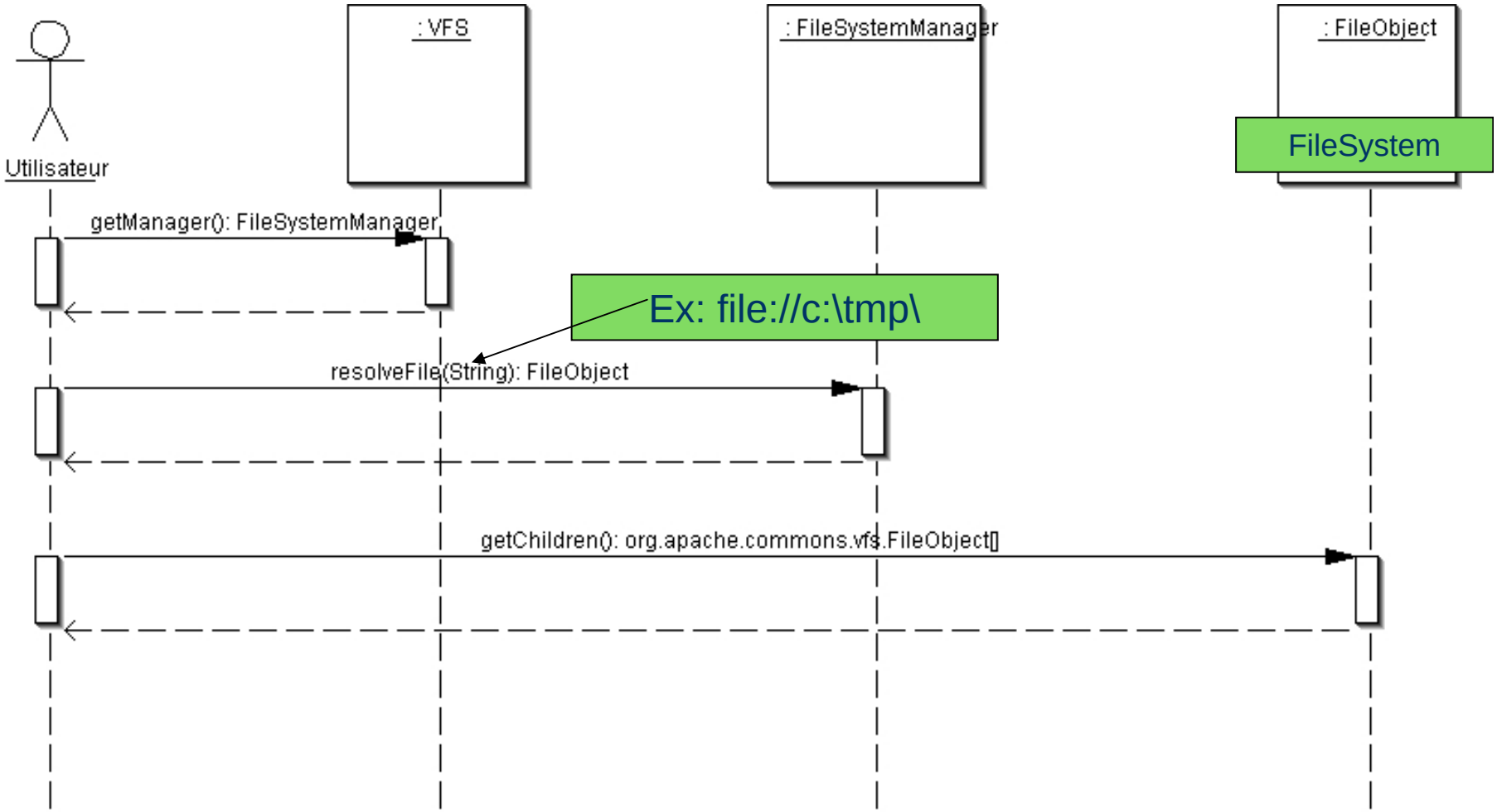
- La manipulation des fichiers est rendue transparente par l'utilisation de nombreuses interfaces.
  - Interfaces représentant les fichiers, les systèmes de fichiers et les gestionnaires de système de fichier….
- Point d'entrée de l'API par la classe VFS
  - static FileSystemManager VFS.getManager()
  - static FileSystemManager VFS.createManager(String managerClassName)
- Résolution des noms de fichiers basée sur le format des URI.
  - FileSystemManager.resolveFile(String name);

Principe de fonctionnement général

- Différents formats URI actuellement supportés :
  - file://*absolute-path*
  - zip://*zip-file-uri*[!*absolute-path*]
  - jar://*zip-file-uri*[!*absolute-path*]
  - http://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - https://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - webdav://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - ftp://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - sftp://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - smb://[*username*[:*password*]@]*hostname*[:*port*][*absolute-path*]
  - tmp://[*absolute-path*]

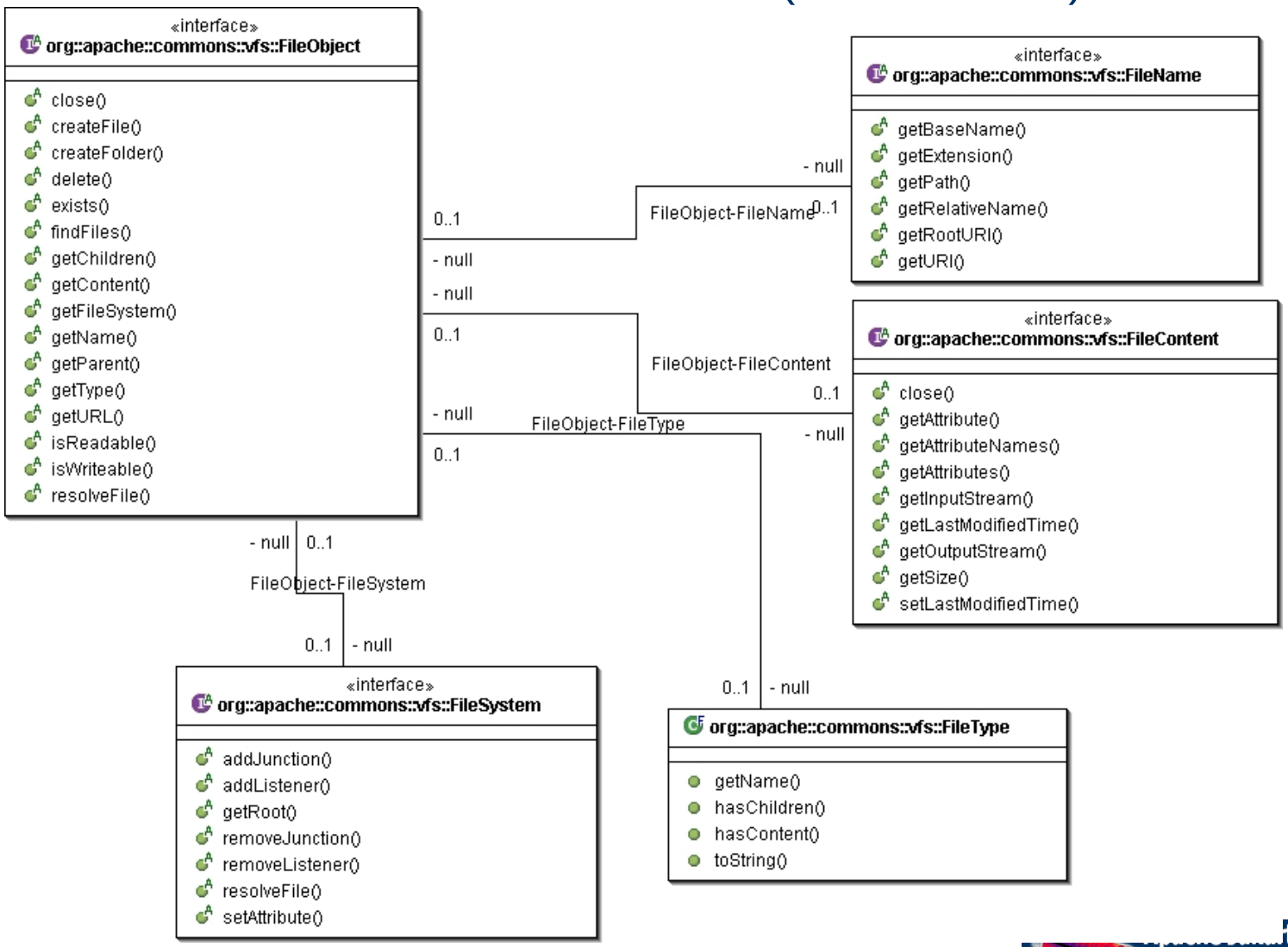
Principe de fonctionnement général

- Un premier exemple



Principe de fonctionnement général

- Interfaces et méthodes communes (non exhaustif)



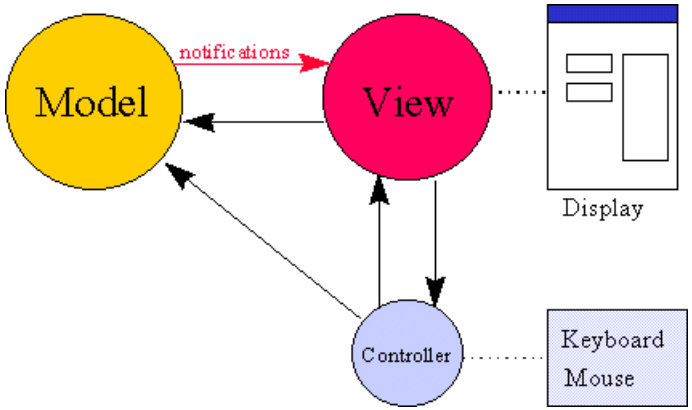
Méthodes et classes de l'API

- Méthode FileObject.findFiles(FileSelector selector)
- Interface FileSelector
  - boolean includeFile(FileSelectInfo fileinfo);
  - boolean traverseDescendents(FileSelectInfo fileinfo);
- Interface FileSelectInfo
  - FileObject getFile();
- Différentes implémentations de l'interface FileSelector
  - AllFileSelector, FileTypeSelector, selectors.SELECT_SELF_AND_CHILDREN, …

Techniques avancée – Recherche de fichiers

- Recherche personnalisée
  - Implémentation de l'interface FileSelector
  - Exemple : Recherche de fichiers en fonction du nom et/ou du type.
- Mise en évidence de la mise en cache d'informations
  - Nouvelle recherche dans le même répertoire
- Problèmes encourus
  - java.lang.OutOfMemoryError
  - Pas de gestion de multi-threading en natif.

- Bref rappel MVC



Limitations

- Gestion des permissions (get, set) pas encore implémentée
- Impossibilité de créer ou de supprimer des fichiers/répertoires dans les systèmes de fichiers ZIP et JAR
- Mécanisme de cache à améliorer
- Pas de gestion d'évènements pour la modification d'un fichier.
- Aucun mécanisme permettant de connaître les modifications externes sur les systèmes de fichiers.
- Manque de documentation !!!

Solutions alternatives

- NetBeans FileSystems API
  - API utilisée par NetBeans pour la gestions des ressources
  - Supporte Fichiers locaux, JAR, ZIP, FTP, CVS
  - Utilisation liée à l'éditeur NetBeans…
  - http://www.netbeans.org/download/dev/javadoc/OpenAPIs/org/openide/filesystems/doc-files/api.html

- SUN Extended FileSystem API
  - API utilisée par SUN WebNFS Client SDK
  - Transition « douce » avec la classe java.io.File
  - Supporte Fichiers locaux, NFS, (CIFS)
  - Nécessite de s'enregistrer auprès de Sun Microsystems!
  - http://wwws.sun.com/software/webnfs/api.html

Conclusion

- API fonctionnelle et simple d'utilisation
- Pas encore finalisée
  - Liste de TODO conséquente
  - Limitations importantes
- Architecture ouverte
- Pérennité de l'API ?
  - Aucun développement depuis 5 semaines
  - 2 développeurs

Questions