25 janvier 2011
David Brun
Théophile Helleboid

API Java
Réalisation d'un Gestionnaire de Téléchargement en Java

Introduction

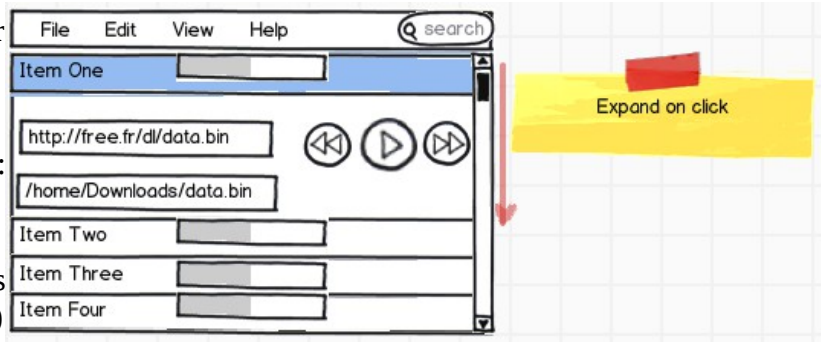
Après une présentation des APIs disponibles sur la plateforme, un projet devait être réalisé en utilisant au mieux une grande partie de ces APIs.

Nous avons choisi de réaliser un gestionnaire de téléchargement, permettant de télécharger plusieurs fichiers en parallèles et pouvant s'exécuter sur toutes les plateformes supportées par les plateformes Java.

Ce gestionnaire de téléchargement a été nommé Jadoma, comme Java Downloader Manager.

Problématique

Jadoma a donc été réalisé à partir du cahier suivant :



- GUI intuitive
- Gestion des états des téléchargements : démarrer, suspendre, reprendre, arrêter
- Import de listes de téléchargements
- Planificateur de téléchargements (démarrer à N, suspendre de N à M, etc.)
- Limite de téléchargements concurrents
- Gestion du protocole HTTP
- Gestion des paramètres (dossier de destination par défaut, planificateur, etc.)
- Gestion de l'historique des téléchargements (terminés, en cours, états, etc.)

Les API Java que nous avons prévues d'utiliser sont :

- Réseau
- Accès aux fichiers
- Multi-threading
- Interfaçage graphique
- Expressions régulières

Conception

Le projet a été conçu autour du pattern MVC : Modèle – Vue – Controlleur. Nous avons donc scindé le code source autour de ces trois axes, afin de nous permettre d'avoir une architecture claire du projet et de pouvoir nous répartir les tâches.

Modèle

Le modèle des objets a donc été pensé afin de respecter au mieux le cahier des charges et les bonnes pratiques de la programmation Java. La classe *Download* décrit donc les objets principaux de Jadoma, à savoir un fichier en cours de téléchargement ou déjà téléchargé.

Le modèle de téléchargement concurrent est représenté quant à lui par les classes *DownloadThread* et *MultiPartDownloadThread* qui étendent la classe *Thread* de l'API Java et qui permettent de gérer de façon fine la gestion des télécharements parallèles.

Des classes supplémentaires, telles que *Scheduler* ou *UserPreferences* permettent de modéliser les

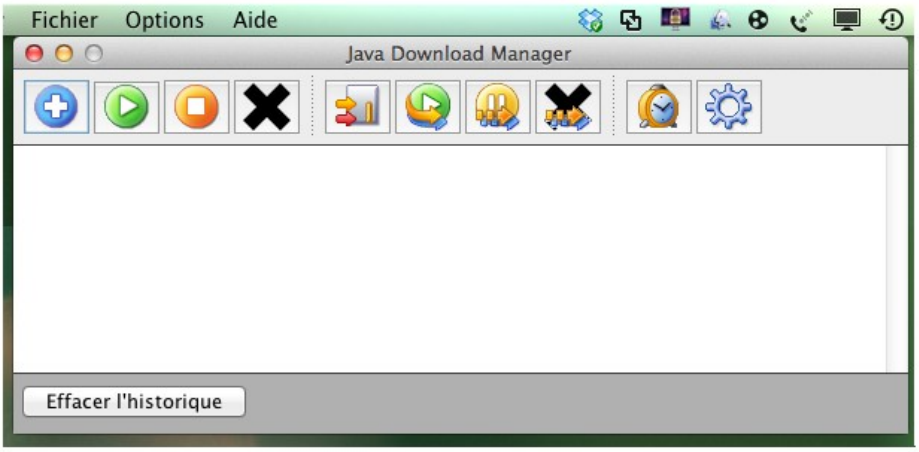
informations manipulées par Jadoma.

Vue

La vue a elle aussi été découpée en plusieurs parties afin de pouvoir programmer efficacement les parties indépendantes de l'interface graphique.

La fenêtre principale, *FrmMain*, est constituée d'une barre de menu, d'une barre d'outils et d'un *JscrollPanel* qui accuille les vues de chaque télécharement.

Chaque événement de la vue est transmis au controlleur, qui agit sur le modèle, et qui met à jour la vue en conséquence. Chaque vue principale possède un Controlleur (*ControllerFrmMain*, *ControllerFrmPreferences*, etc.), ce qui permet de dissocier les actions spécifiques à ces vues.



Controlleur

Le Controlleur principal, *ControllerFrmMain*, est récipiendaire de tous les évènements de la fenêtre principale, et gère le modèle objet en conséquence.

Il a donc directement connaissance de tous les téléchargements en cours ou terminés, via le DownloadManager à qui il délègue la gestion des Threads Java.

Chaque téléchargement déclenche un *Thread* indépendant, qui se termine lors de la fin du téléchargement. Sans Thread indépendant, il n'était pas possible d'avoir un réel comportement parallèle au niveau du téléchargement.

Développement

Afin de s'assurer du bon fonctionnement de Jadoma et de son aspect multi-plateforme, le développement de l'application a été fait sur deux environnements différents, à savoir MacOS X et Linux Debian.

Chaque exécution du projet a permis de corriger les problèmes graphiques ou d'implémentation spécifique sur l'une ou l'autre des plateformes.

Malgré l'utilisation un maximum des fonctions et des implémentations Java indépendantes du système d'exploitation, nous n'avons pas réussi à obtenir un résultat équivalent au pixel près sur toutes les plateformes.

Le code source a été partagé via git, ce qui a permis un développement rapide et un partage du code source facilité. La possibilité de commit hors-ligne a également été apprécié par les auteurs du projet, qui ont pu maintenir une qualité de commit unitaire tout au long du projet, même sans connexion Internet.

De plus, l'environnement de développement choisi s'est porté sur Eclipse, pour sa facilité d'utilisation et sa connaissance par les deux auteurs du projet.

Tests

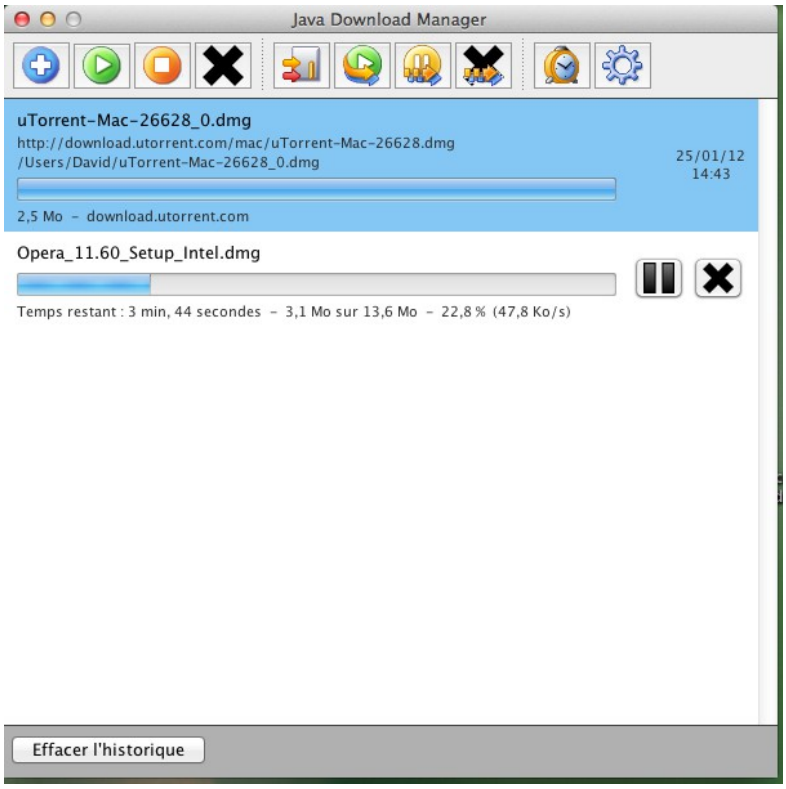
Le projet a été testé tout au long du projet, avec plusieurs URL.

Cependant, nous n'avons pas implémenté de tests unitaires par manque de temps. Il est également difficile de tester l'affichage graphique de l'application, alors qu'elle représente une part importante du développement de Jadoma.

Interface intuitive

Afin d'atteindre au mieux l'objectif de l'interface intuitive, nous nous sommes intéressés à l'existant pour les interfaces graphiques d'autres logiciels de téléchargement, notamment Free Download Manager.

Nous nous sommes apercçus que la conception d'icônes claires et compréhensibles par tous n'était forcément évidente et avons préféré reprendre des icônes déjà existantes dans une bibliothèque d'images.



Conclusion

Respect du cahier des charges

Le cahier des charges de départ a été entièrement respecté, tous les points précisés ont été implémentés.

Par manque de temps, nous n'avons cependant pas réussi à implémenter les « Bonus Points », qui demandent en réalité un travail bien plus approfondi que celui déjà effectué.

Jadoma représente au final 2800 *Single Lines of Code*.

Plus value pédagogique

Lors du développement de cette application, nous avons donc pu appréhender de nombreuses API de la plateforme Java, ainsi que le développement d'un projet somme toute conséquent.

Nous avons pu également appréhender la conception d'un logiciel multi-plateforme, ce qui représente une difficulté et des efforts supplémentaires, notamment au niveau des tests et de la conformité graphique du logiciel.

La grande variété des plateformes n'a par exemple par permis de tester le logiciel sur toutes les

versions de Linux.