Systèmes ouverts

Compte-rendu

ENSISA Mulhouse

3A Informatique et Réseaux

David Brun

Michaël Muré

Décembre 2011

I. Présentation du sujet

Dans le cadre de la 3ème année de formation d'ingénieurs en Informatique et Réseaux de l'ENSISA, un cours dénommé "Systèmes Ouverts" est proposé aux étudiants. Son thème est orienté vers la découverte du langage Smalltalk et des notions associées : "tout est objet", tout est modifiable, le typage est dynamique, etc.

Toutes ces notions sont plutôt anciennes puisqu'elles sont nées dans les années 70 avec les premières versions de Smalltalk. Néanmoins elles sont toujours d'actualité et ont influencé de nombreux langages de programmation plus récents comme Objective-C, Java, Ruby, etc. Elles permettent notamment de réaliser des systèmes ouverts, entièrement reconfigurables à chaud par les utilisateurs.

Afin de mettre en pratique les notions enseignées pendant ce cours, il nous a été demandé de concevoir et d'implémenter un système ouvert dans un langage de notre choix.
Il convient donc de réifier au maximum les notions utilisées dans notre système, et de concevoir un système modulable qui permette un câblage à chaud des différentes classes.

Le sujet consistait donc en la réalisation d'une application générique, disposant de commandes prédéfinies, et capable de charger dynamiquement d'autres commandes pour pouvoir enrichir son comportement et les mettre à disposition de l'utilisateur.

Afin de tester le bon fonctionnement de notre système ouvert et de voir à quel point il était modulable, nous avons souhaité réaliser deux librairies concrètes.

II. Présentation du travail réalisé

Étant donné que nous avons déjà été confrontés à l'introspection dans le langage Java, que ce soit par des développements personnels ou dans le cadre d'autres projets de l'ENSISA, nous avons décidé d'implémenter notre système ouvert dans ce langage. Le projet a été réalisé sous Mac OS X et Linux avec l'IDE Eclipse. Ces choix sont dictés par la maîtrise de la plateforme et de l'IDE.

L'architecture logicielle étant libre, il a tout d'abord fallu concevoir un système modulable tout en identifiant les fonctions de chacun des modules. Nous avons 6 entités de base :

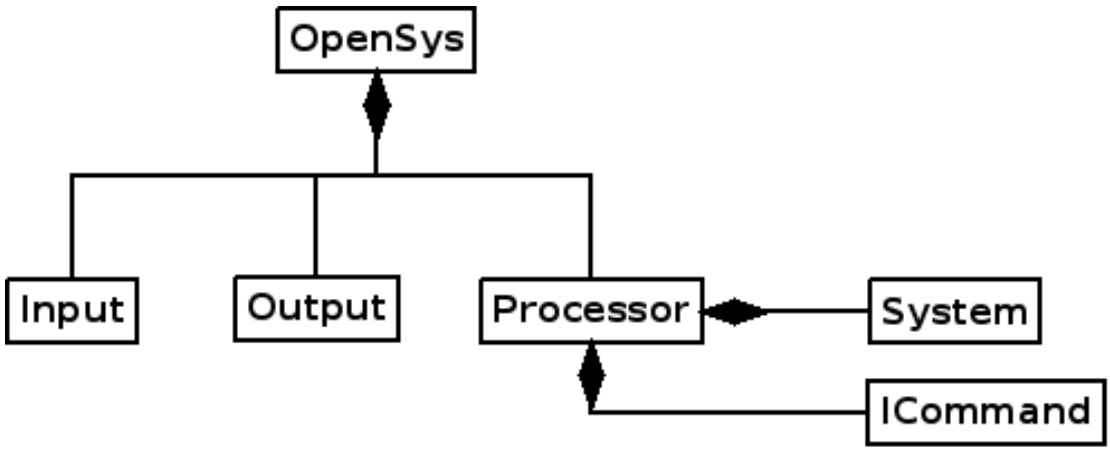
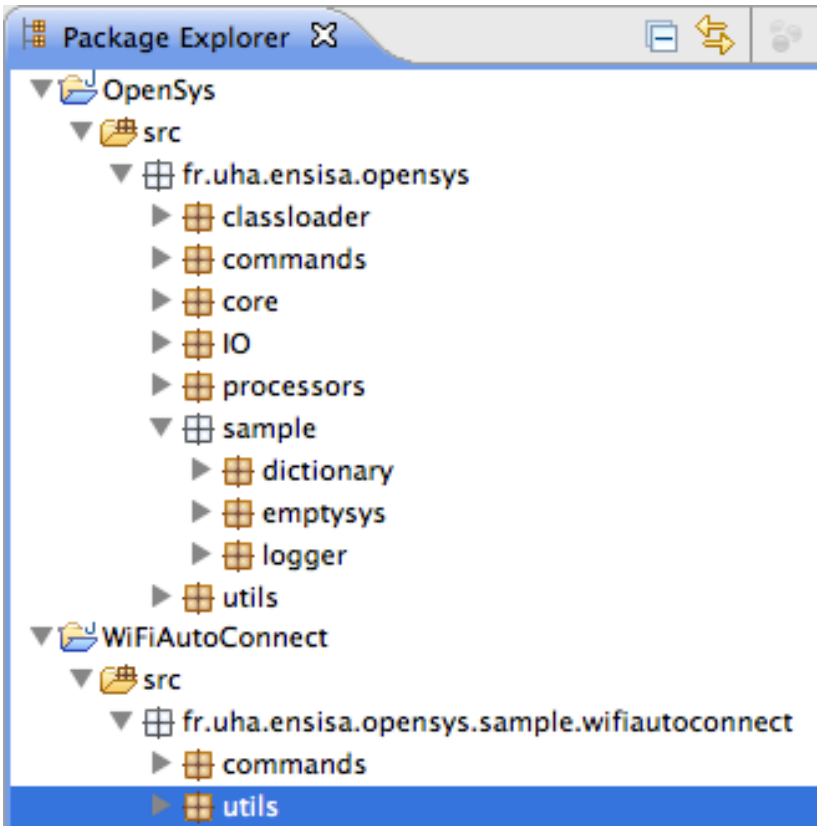


Diagramme UML simplifié de notre système ouvert

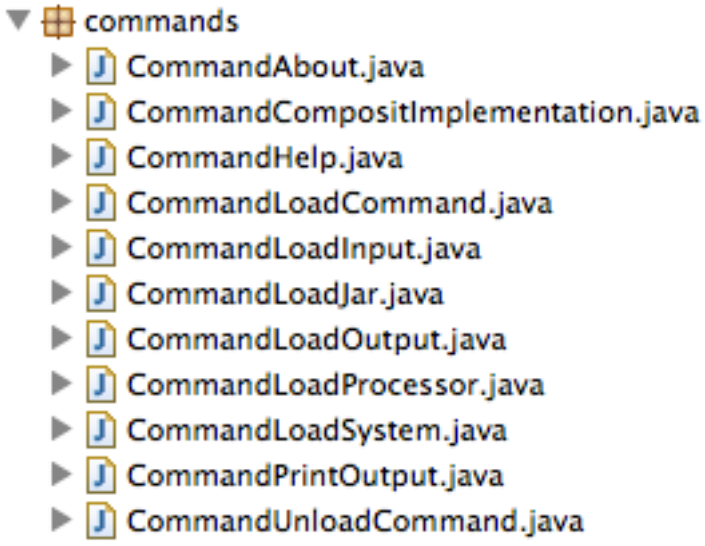
Nous avons décidé d'organiser notre code sous forme de packages logiques :



Packages de notre système ouvert

À l'aide des commandes de base, nous pouvons modifier `OpenSys` en redéfinissant les `Input`, les `Output` et les `Processor`.

À l'aide des commandes de base, nous pouvons modifier `Processor` en ajoutant ou supprimant des commandes internes ou externes (chargées depuis un jar).



Commandes de base

L'application a été codée par ajouts successifs de fonctionnalités. Chacune d'elles était testée et validée avant de continuer le développement. Nous avons commencé par développer le `Core` de l'application, puis les éléments de base pour avoir un système fonctionnel (IO standards, processeur par défaut, commandes de base). Ensuite nous avons développé la partie qui permet de charger des commandes externes via un `ClassLoader` enrichi.

Puis nous avons réalisé une implémentation concrète avec un `Dictionary` qui fourni des commandes spécifiques (afficher sa taille, traduire, ajouter ou supprimer une entrée, afficher son contenu).

Ensuite nous avons enrichi les fonctionnalités au niveau des IO en permettant d'utiliser des fichiers, ou d'utiliser JLine, qui permet d'obtenir un vrai shell Java avec de l'autocomplétion et un historique de commande.

Enfin, pour bien montrer la modularité du système ouvert, nous avons réalisé un projet externe, qui référence `OpenSys`, et qui définit des commandes pour tester la connectivité à internet et s'authentifier sur les réseaux UHA ou FreeWiFi. Cette librairie est packagée dans un jar et on peut la charger grâce aux commandes de base.

La clarté du code source étant essentielle pour le bon déroulement d'un projet, nous nous sommes appliqués à le formater correctement en respectant notamment des conventions d'écritures (CamelCase, indentations, etc.). L'utilisation d'un IDE permet aussi d'automatiser les bonnes pratiques.

Nous avons utilisé Git comme gestionnaire de version afin de pouvoir travailler de manière collaborative et garder une trace des révisions.

III. Conclusion

Le domaine des systèmes ouverts en lui même est très intéressant puisqu'il propose de créer des applications évolutives, où des utilisateurs pourraient ajouter à chaud des fonctionnalités en important des "librairies" codées de façon à respecter l'architecture mise en œuvre.
On pourrait même imaginer des extensions pour rendre les systèmes intelligents et leur permettre de créer d'autres systèmes à partir de fonctionnalités piochées dans différentes "librairies" de base.

Nous avons réussi à créer un système évolutif, où l'utilisateur a la possibilité de câbler à chaud son système. Il peut donc changer d'entrées / sorties, changer de processeur, charger ou décharger des commandes prédéfinies, ou même charger des "librairies" externes sous forme de jar.
Il semble que le projet soit globalement correct puisque les fonctionnalités sont présentes et opérationnelles. En outre, l'architecture est robuste et l'esprit du sujet est bien compris.

D'un point de vue technique, le mécanisme d'introspection de Java nous a permis d'explorer les différents packages présents pour repérer les différentes entités modulables disponibles. Le `ClassLoader`, couplé à un explorateur de fichiers zip, nous a permis de charger dynamiquement des classes venant d'une "librairie" jar externe potentiellement inconnue respectant l'architecture. Enfin, la `CommandComposit` permet à un utilisateur de construire en temps réel sa propre commande à partir de commandes de bases.