Département de génie logiciel et des TI

Cours : LOG430 – Architecture Logicielle

Rapport de Laboratoire

|  |  |
| --- | --- |
| Laboratoire | 3 |
| Équipe | 5 |
| Noms | Filteau-Tessier, Émile  Hinse, Marc-Antoine  Gilber, Alex |
| Codes permanents | FILE05039202  HINM11039207  GILA16079207 |
| Cours | LOG430 |
| Trimestre | Hivers 2014 |
| Enseignant | Dominic St-Jacques |
| Chargés de laboratoire | Samir Djeffal |
| Date | 10 Mars 2014 |

# Introduction

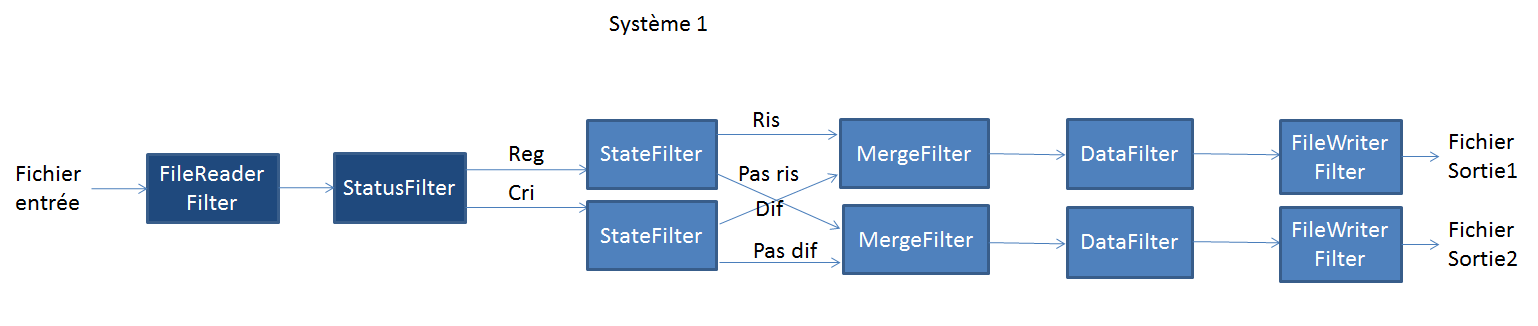
Ce 3ième laboratoire est différent des deux premiers. Un système initial est fourni utilisant le style architectural « Pipe & Filter », mais il doit être utilisé pour créer deux nouveaux sous-systèmes. Ces deux sous-systèmes comporteront des modifications différentes tout en gardant les fonctionnalités originales. Ensuite, une analyse architecturale des nouveaux systèmes sera présentée.

# 1. Sommaire de l’implémentation

Comme expliquer plus haut, l’implémentation se divise en deux parties soit le système A, et le système B.

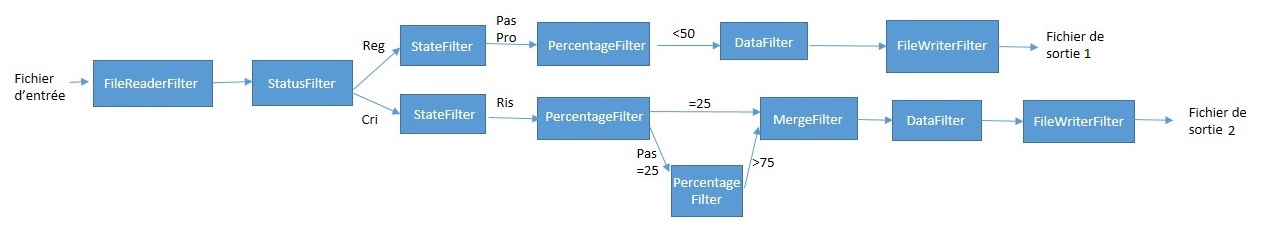
Système A :

Ce premier système comporte 3 modifications du système orignal. La première modification consiste à changer le contenu et le format des données de sortie du système. La sortie doit donc contenir uniquement les champs : *Statut, État, Taux, Numéro Projet* dans cet ordre. La deuxième modification consiste à créer un deuxième fichier sortie contenant les projets qui ont un autre état que celui en paramètre (ceux qui ne seront pas dans le premier fichier). Le format de sortie de ce fichier doit être le même que le premier. Finalement, les deux fichiers doivent être tirés en ordre alphabétique selon le champ *État*.



Système B :

Le système B est soumis à la même modification du format de fichier de sortie que le système A. En plus de cette modification, deux fichiers sont produits. Le premier fichier contient les projets avec le statut régulier (REG) dont l’état est (RIS) ou (DIF) ayant un taux de progression inférieur à 50%. Le deuxième fichier de sortie contient les projets avec le statut critique (CRI) ayant l’état (RIS) et un taux de progression de 25% ainsi que ceux qui ont un état différent de (RIS) qui ont un taux de progression supérieur à 75% .



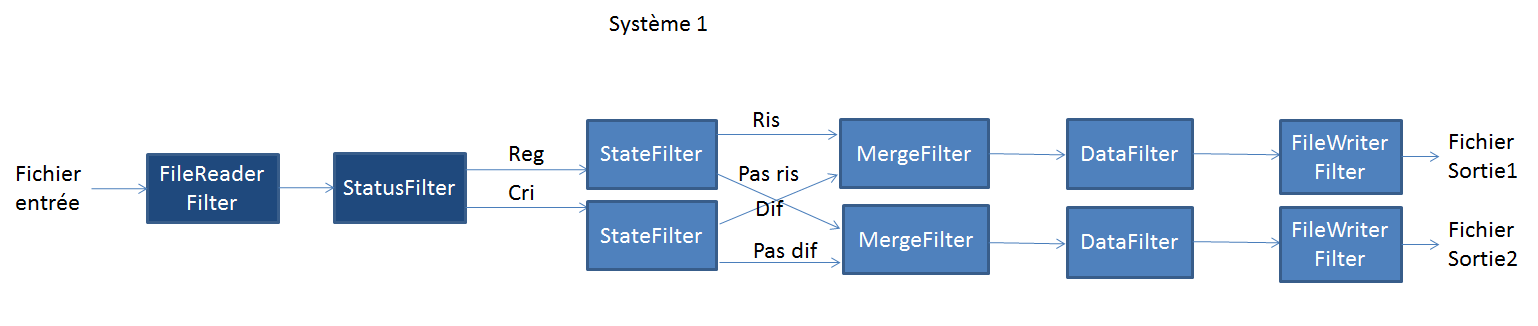
# 2. Analyse architecturale

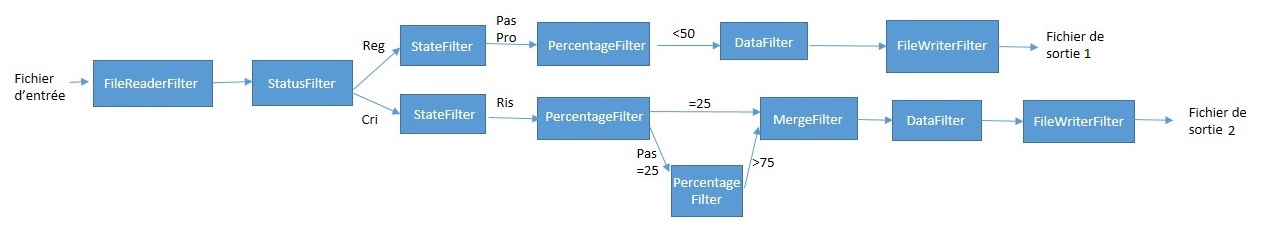
a) Les deux systèmes ont une architecture de style « pipe and filter ». Chacun des systèmes peuvent réutiliser les filtres créés afin d’augmenter l’épuration des informations. Les filtres sont faciles à assembler au système et chacun des filtres travaillent en parallèle, ce qui augmente la vitesse de traitement du système en entier. Tous les filtres créés et modifiés ont des données en entrée et en sortie et ne conservent aucun état entre les instanciations et ne connaissent aucune information des filtres qui ont été utilisé en amont et en aval. Chacun des tubes (pipe) reliant les filtres ont une transmission unidirectionnelle.

Étant donné qu’un filtre peut avoir plusieurs tubes de sortie, nous avons modifié le StateFilter du premier système afin qu’il prenne deux flux de sortie dans le but de desservir les demandes du système. Dans le système B, au lieu d’ajouter un flux de sortie au StateFilter, un booléen est demandé en paramètre au filtre afin de déterminer si le système doit vérifier si oui ou non l’état fait partie de la donnée.

Un filtre nommé DataFilter a été créé afin de filtrer les informations en surplus dans les données restantes avant l’écriture du fichier.

Un filtre nommé PercentageFilter a été créé afin de déterminer le taux d’avancement des projets reçus par le filtre qui prend un pourcentage et une opération (>,<,=) en paramètres.





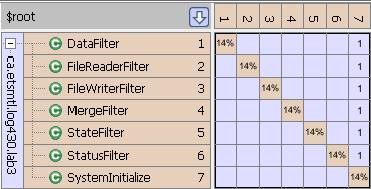
b) Tout d’abord nous avons analysé les deux différents systèmes, ceux-ci possédaient déjà une architecture « pipe and filter » assez bien structuré donc pour effectuer nos modifications nous avons simplement décidé de focaliser sur la constance. En gardant la même structure déjà en place, nous pouvons facilement travailler séparément dès le départ sans avoir à se consulter régulièrement afin de s’entendre sur les standards.

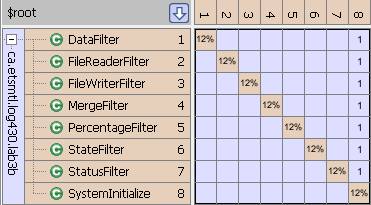
Pour le système les deux systèmes, nous avons choisi de séparer le flot en deux parties distinctes étant donné que nous devions produire deux fichiers différents. La séparation s’effectue au niveau du « StateFilter » étant donné que c’est lui qui possède le critère de séparation des deux fichiers.

L’attribut de qualité constance fut donc notre attribut principal étant donné que nous avons désiré gardé la structure de l’architecture en place. Également, nous avons respecté l’attribut de maintenabilité en gardant un couplage très bas avec l’architecture « pipe and filter » , chaque classes est relié au flux principal d’où on peut facilement ajouter ou retirer des filtres en modifiant seulement une ou deux lignes de codes. Si nous voulions ajouter de nouveaux filtres, nous pourrions simplement le développé indépendamment des autres en ensuite le relié par la classe SystemInitialize.java. Le critère de maintenabilité est donc très bien respecté dans notre architecture.

c)

Matrice de dépendance du système A :



Matrice de dépendance du système B

Les matrices de dépendances permettent de constater que la classe *SystemInitialize* dépend de tous les filtres du système car ils sont tous appelés indépendamment dans le *main* de l’application. On peut donc dire que le système respecte le style architectural « *Pipe & Filter* »

d) Oui.

Selon la définition, un filtre est une transformation incrémentale du flux de données.

* Ajout d’information au flux de données
* Concentration ou extraction d’information
* Changement de format de l’information

Dans le système présenté, le flux de données serait représenté par les projets qui sont passé en entrée et qui en ressorte sous un format différent. Le FileReaderFilter et le FileWriterFilter sont donc des filtres car ils extraient et concentrent respectivement le flux de données du système en transformant le format de ce dernier.

# Conclusion

Ce laboratoire avait pour but de nous familiariser avec le style architectural « *Pipe & Filter*». Pour ce faire, un système utilisant ce style architectural est fourni et il fallait en faire deux systèmes comportant des modifications différentes. Suite à ces modifications, une analyse architecturale des systèmes a été faite pour mieux comprendre comment ces systèmes sont représentatifs du style « *Pipe & Filter*».

# Annexe