Rapport de TP
SEcurité et COmposants

Benoit Meilhac
Master 2 SSL

Table des matières

1 Exercices 1 à 5

1.1 Exercice 1

Le but de cet exercice est la modification de l'exemple fourni : *helloworld* afin que le serveur renvoi un message d'erreur au client quand le message à imprimer est vide.

Les fichiers concernés sont :

- **ServerImpl.java** : la fonction *void print(final String)* retourne désormais un String. Ce String sera un message d'erreur si le message en paramètre est vide ou null. Dans le cas contraire, il signifiera juste que l'opération a bien été effectuée ;
- **ClientImpl.java** : un String récupère maintenant l'appel à la fonction *print()* du serveur. Ce String est ensuite affiché pour informer le client du traitement de sa demande.

1.2 Exercice 2

Le but de cet exercice est la mise en place de l'architecture présentée dans la figure 2 du sujet en utilisant l'API Java.

Pour ce faire, le package **lanceur** a été créé, il contient le fichier **MonServer.java** qui permet de répondre à l'exercice.

1.3 Exercice 3

Le but de cet exercice est la même que pour l'exercice 2 mais en utilisant l'ADL Fractal.

Le package **comanche** contient le fichier **Comanche.fractal** qui permet de répondre à l'exercice.

1.4 Exercice 4

Le but de cet exercice est d'implanter un nouvel Handler : *HttpHandler* qui permettra de rediriger l'utilisateur sur la page web Google.

Plusieurs fichiers du package *comanche* ont été modifiés pour répondre à l'exercice :

- **Comanche.fractal** :
  - mise en commentaire du composant **FileHandler** et de son lien avec le dispatcher ;
  - ajout du composant **HttpHandler** et de son lien avec le dispatcher.
- création du fichier **HttpRequestHandler.java** qui implémente RequestHandler. La surchage de la fonction *handleRequest()* contient la ligne fourni dans l'énoncé de l'exercice 4 ;
- **Server.java** : mise en commentaire du FileRequestHandler et de son lien avec le dispatcher, création du HttpRequestHandler et de son lien avec le dispatcher.
  Il est à noter que ce fichier est utilisable dans le cas où on souhaite exécuter l'exercice sans le script Ant. Dans le cas contraire, il est inutile.

1.5 Exercice 5

Le but de cet exercice est de créer une redirection dynamique seulement quand on entre l'adresse suivant : *http ://localhost :8080/reconf*.

Pour ce faire, plusieurs fichiers du package *comanche* ont été modifiés :

- **Comanche.fractal** : on reprend celui de l'exercice 4, à la différence où **FileHandler** ainsi que son lien ne sont plus en commentaire ;
- **HttpRequestHandler.java** est le même que pour l'exercice 4 ;
- **RequestAnalyser.java** : la fonction *void handleRequest(Request)* est modifiée. Elle intégère le fait que si on récupère un **/reconf**, alors le lien avec **FileRequestHandler** est supprimé et celui avec **HttpRequestHandler** est créé.

2 Exercice 6

2.1 Modélisation

La modélisation suivante représente le système du photocopieur.

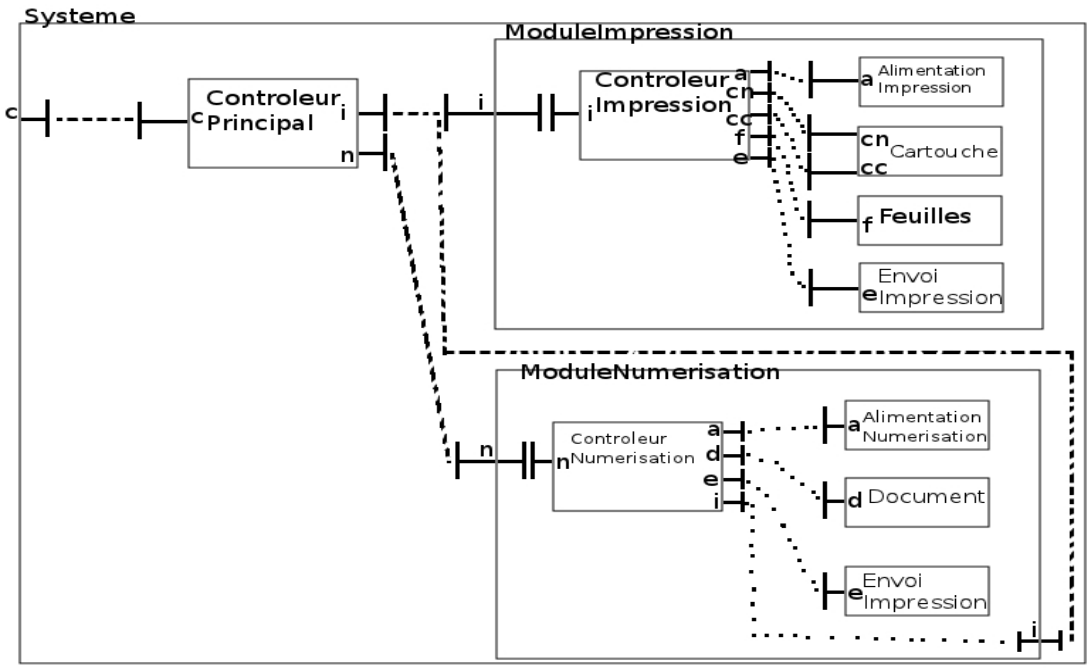

FIGURE 2.1 – Modélisation Fractal ADL du photocopieur

Le contrôleur principal reçoit une demande de l'utilisateur. En fonction de cette demande, il l'envoi soit au module numériseur, soit au module imprimante.

Ces deux modules contiennent chacun un contrôleur spécifique qui leur permet de vérifier si leurs pré-requis sont respectés. Dans le cas de l'imprimante, elle aura besoin des composants suivant :

– alimentation ;
– cartouches d'encre couleur ou noire ;
– feuilles.

Pour le numériseur, il aura besoin des composants :
– alimentation ;
– document.

Une fois ces pré-requis satisfait, le contrôleur fait suivre l'information à un composant d'envoi qui affiche sur la sortie standard.

2.2 Structure de l'exercice

L'exercice est décomposé en trois packages et un fichier fractal. Le fichier fractal reprend l'architecture précédemment définie. Les trois packages représentent :
– les outils externes utilisés : **outils** ;
– l'ensemble des interfaces : **interfaces** ;
– l'ensemble des modules : **systeme**.

2.2.1 Le package *outils*

Ce package contient une classe **Clavier** qui permet de gérer les entrées clavier et d'en retourner un type spécifique (int, ...).

Il contient également la classe **RecuperationClavier** qui permet d'utiliser la classe **Clavier** à plus haut niveau. C'est la classe qui sera appelée lors des demandes de saisie clavier.

2.2.2 Le package *interfaces*

Ce package contient l'ensemble des interfaces nécessaires à la communication entre composants. Liste des interfaces :
– **RequestAlim.java** : gestion de l'alimentation ;
– **RequestCartoucheC.java** : gestion d'une cartouche couleur ;
– **RequestCartoucheN.java** : gestion d'une cartouche noire ;
– **RequestDoc.java** : gestion d'un document à numériser ;
– **RequestEnvoiImpression.java** : gestion d'une envoie d'impression ;
– **RequestEnvoiNumerisation.java** : gestion d'un envoi de numérisation ;
– **RequestFeuille.java** : gestion d'un bac de feuilles ;
– **RequestImpression.java** : gestion d'une demande d'impression ;

– **RequestNumerisation.java** : gestion d'une demande de numérisation.

2.2.3 Le package *systeme*

Ce package contient l'implémentation des composants du système. Ces composants sont décrit dans la suite du rapport.

AlimentationImpression

Cette classe implémente *RequestAlim* et permet de gérer et de connaître l'état actuel de l'alimentation pour une impression. Une alimentation ne possède que deux états symbolisés par un booléen à *vrai* ou *faux*.

Par défaut, l'alimentation est éteinte.

AlimentationNumerisation

Cette classe implémente *RequestAlim* et permet de gérer et de connaître l'état actuel de l'alimentation pour une numérisation. Une alimentation ne possède que deux états symbolisés par un booléen à *vrai* ou *faux*.

Par défaut, l'alimentation est éteinte.

Cartouche

Cette classe implémente *RequestCartoucheC* et *RequestCartoucheN* et permet de gérer et de connaître l'état actuel d'une cartouche noire ou de couleur. Les deux types de cartouche possèdent un niveau d'encre respectif.

Ce niveau est fixé à 10 par défaut.

ControleurImpression

Cette classe implémente *RequestImpression* et *BindingController*.

Elle a 2 fonctions principales :

– elle gère les composants dont elle a besoin ;

– elle lance la procédure d'impression.

Pour ce faire, elle demande un document, un nombre de pages et si oui ou non on souhaite une impression couleur.

Ensuite, elle contrôle l'état des différents composants à l'aide de fonctions internes. Si un composant est éteint (alimentation) ou vide (feuilles, encre), une demande est formulée à l'utilisateur pour résoudre le problème. Si la demande n'est pas satisfaite, l'impression est annulée.

Si la procédure de contrôle est satisfaite, l'étape suivante est le lancement de l'impression. Les composants encre et feuilles voient leur niveau diminuer en fonction du nombre de pages demandées. À savoir qu'une copie vaut un niveau d'encre et une feuille.

L'impression est ensuite envoyée au composant *envoiImpression* dont le rôle sera expliqué plus tard.

Il est à noter que si le nombre de copies demandées dépasse le niveau des cartouches ou le nombre de feuilles dans le bac, la procédure de vérification se relance et il est redemandé à l'utilisateur d'interagir avec les composants requis.

Par exemple, si on demande 15 copies, le contrôleur effectue une première vérification des composants, lance l'impression de 10 pages (10 : valeur par défaut pour l'encre et le nombre de feuilles, cette valeur peut être changée), puis revérifie les composants et enfin termine avec l'impression des 5 dernières pages.

ControleurNumerisation

Cette classe implémente *RequestNumerisation* et *BindingController*.

Elle a 3 fonctions principales :

- elle gère les composants dont elle a besoin ;
- elle lance la procédure de numérisation ;
- elle peut effectuer une demande d'impression.

Pour ce faire, elle demande si une impression est demandée, son nombre de copies souhaitées et si oui ou non une impression couleur est requise.

Ensuite, elle contrôle l'état de l'alimentation. Si elle est éteinte, il est demandé à l'utilisateur de l'allumer. S'il ne le veut pas, la procédure de numérisation est annulée.

S'il accepte de l'allumer, le composant *Document* est appelé. Son rôle sera expliqué dans la suite du rapport. Ce composant retourne le contenu d'un fichier.

Une fois ce fichier récupéré, il est transmis au composant *envoiNumerisation* dont le rôle sera lui aussi expliqué dans la suite du rapport.

Si une demande d'impression a été formulée, elle est transmise au composant **ControleurImpression** qui a été détaillé précédemment.

ControleurPrincipal

Cette classe implémente *java.lang.Runnable* et *BindingController*.

Elle permet de gérer les interactions avec l'utilisateur via un menu de sélection, pour ensuite transmettre ses demandes aux modules souhaités.

Le contrôleur demande :
- impression ou numérisation ;
  - si impression :
    - le chemin vers un fichier (chemin absolue sans raccourci) ;
    - le nombre de pages ;
    - impression en couleur ou non ;
    - la demande est ensuite transmise.
  - si numérisation : demande d'impression du document à la fin de la numérisation ;
    - si demande d'impression : le nombre de pages ;
    - impression en couleur ou non ;
    - la demande est ensuite transmise.
- si on veut relancer une nouvelle tâche et dans ce cas, on recommence la boucle.

Dans la demande de chemin pour l'impression, si le chemin est incorrect ou le document non trouvé, alors la chaîne vide sera transmise.

Document

Cette classe implémente *RequestDoc*. Elle permet de numériser en quelque sorte un document. En quelque sorte car en fait, ce composant ne fait (à défaut de pouvoir interagir directement avec une vraie imprimante) que demander un chemin vers un fichier dont le contenu sera extrait et retourné sous forme d'un String.

Si le fichier n'est pas trouvé au chemin indiqué, la chaîne vide est retournée. De ce fait, le numériseur ne plante pas, ne lance pas d'exception, mais fonctionne dans le même état d'esprit qu'un vrai numériseur dans le fait qu'il est possible de lancer une numérisation sans aucun document présent.

EnvoiImpression

Cette classe implémente *RequestEnvoiImpression*. Elle permet d'afficher sur la sortie standard le document venant du contrôleur d'impression.

EnvoiNumerisation

Cette classe implémente *RequestEnvoiNumerisation*. Elle permet d'afficher sur la sortie standard le document venant du contrôleur du numériseur.

J'ai essayé d'implémenter l'envoi par mail, mais avec la configuration réseau de la fac j'ai dû abandonner cette idée, je ne pouvais envoyer un mail qu'à moi-même...

Feuille

Cette classe implémente *RequestFeuille*. Elle permet de simuler un bac de feuilles.

Ce bac de feuilles possède un nombre de feuille et une capacité de contenance maximale de 10 feuilles. Cette capacité peut bien entendu être changée.

Par défaut, le bac de feuilles est vide.

2.3 Notes sur l'exercice

Le lancement s'effectue avec le script ant. Attention cependant à lancer le script en ligne de commande : *ant execute*.

En effet, un problème est apparu dans la saisie clavier lors du lancement de l'application avec eclipse : celui-ci ne prend pas les entrées clavier. C'est pourquoi il faut lancer l'exercice en ligne de commande.

Pour toutes questions : benoit.meilhac@gmail.com