Projet Applications réflexives

Une plateforme de services dynamiques Bri

Introduction

Le projet que vous allez réaliser est une introduction à la problématique des services dynamiques telle que l'offre l'alliance OSGi (d'où le nom-canular de ce projet). Le but est d'offrir un lieu d'échanges entre des **programmeurs** Java développant des services aussi diversifiés qu'un service de mail, une analyse syntaxique XML, la journalisation des activités, etc et des non-programmeurs ayant besoin de ces services. Le framework OSGi est d'une grande complexité et ce projet ne vise qu'à vous aider à terme à mieux comprendre de telles plateformes.
Pour des raisons de clarté, les non-programmeurs seront ici dénommés **amateurs** (à la fois parce qu'ils sont amateurs des services proposés et pour les distinguer des professionnels que sont les programmeurs).
Dans cette description du projet, les parties *(en italique et entre parenthèses)* sont des options - parfois complexes - qui ne concernent que les pires geeks d'entre vous.

Serveur BRiLaunch

La plateforme dynamique de services BRiLaunch est donc un serveur Java qui accueille les amateurs sur un port et les programmeurs sur un autre. Les services fournis par les programmeurs peuvent être installés, mis à jour, *(arrêtés, démarrés, et désinstallés)* de manière distante par leur créateur-programmeur sans avoir à redémarrer BRiLaunch. Les services sont fournis par le programmeur sous forme d'une classe respectant la norme BRi et que le serveur ira chercher sur le serveur ftp du programmeur : chaque programmeur dispose d'un serveur ftp d'adresse stable et déclarée à la création de son compte.
(Si le service est un regroupement de classes, éventuellement avec des ressources partagées, une bibliothèque .jar avec la règle qu'une seule classe de cette bibliothèque sera une service BRi et que le bloc static de cette classe définira l'initialisation du service (création des ressources nécessaires au service))

Deux applications clientes seront donc développées, **clientprog** et **clientama**, pour se connecter et utiliser BRiLaunch. Le problème du déploiement des clients sur des postes précis ne sera pas abordé dans ce projet.

Programmeur de services

Les programmeurs sont référencés par un login/mdp dans BRi. Un nouveau programmeur devra donner ces informations ainsi que l'url de son serveur ftp pour être certifié BRi et utiliser la plateforme. Cette certification sera ici supposée automatique à la création du compte.
La norme BRi que doit respecter le programmeur est celle que vous aurez testée au tp4. En plus, afin d'éviter des conflits de noms de classe, un programmeur doit mettre tout ce qu'il développe dans un package portant comme nom son login (mon login est *brette*, je développe toutes mes classes dans un package *brette*), ce qui bien sur doit aussi être contrôlé.
(Dans le cas où le service est une bibliothèque (.jar), s'ajoute à la norme l'unicité de la classe de service.)

Un programmeur se connecte avec **clientprog** au port PORT_PROG et, après authentification, indique ce qu'il veut faire :

- Fournir un nouveau service ;
- Mettre-à-jour un service ;
- Déclarer un changement d'adresse de son serveur ftp
- *(Démarrer/arrêter un service ;*
- *Désinstaller un service.)*

Le programmeur peut aussi être intéressé par des services existants et les déclencher en tant qu'amateur mais il devra pour ça se connecter au port amateur avec clientama (voir paragraphe suivant).
Les données nécessaires à chacune de ces options sont suffisamment évidentes pour ne pas nécessiter une description exhaustive.

Amateur de services

L'amateur se connecte avec **clientama** au port PORT_AMA et il choisit dans la liste des services que lui communique BRi celui qu'il souhaite utiliser. (*Seuls les services démarrés lui sont proposés)*. Le lancement du

service est réalisé suivant le modèle étudié au TP4. Le code de l'application cliente de l'amateur devra être généralisé au maximum de façon à fonctionner quelles que soient les modalités de communication liées au service rendu.
(Pour bénéficier de certains services (messagerie interne par exemple), l'amateur devra aussi être authentifié par login/mdp mais ceci restera géré en interne par le service).

Un service simple : inversion d'un texte (non ? si !)

La donnée du service : un texte (String)
Le service : le résultat est l'inversion de la donnée
Le résultat : un texte (String)
Pour ce service, les données/résultats peuvent très facilement utiliser la socket.

Un service avec échange de fichiers : analyse de fichier xml

La donnée du service : un fichier xml (lire ce fichier sur un serveur ftp)
Le service : analyser le fichier (règles d'analyse à définir)
Le résultat : un rapport d'analyse (fichier déposé sur serveur ftp)

Un service avec ressource partagée : messagerie interne

Envoi d'un message
La donnée du service : le pseudo du destinataire, le message
Le service : stocker le message dans une ressource du service
Le résultat : une confirmation
Lecture des messages :
La donnée du service : aucune
Le service : récupérer les messages de la personne identifié
Le résultat : les messages (String ? sérialisation d'une liste de messages ?)

Cahier des charges du projet (service minimum)

La réalisation demandée doit fonctionner pour des classes de services sans ressources partagées et dont les échanges sont limités à des textes (au minimum donc le service d'inversion et quelques variantes de travail sur une String).
Les services à base de bibliothèques .jar (messagerie) et les services nécessitant des échanges de fichiers (fichier à analyser-rapport d'analyse) sont des options.

Quand, quoi, comment ?

Le projet est à réaliser individuellement ou par binôme. Dans le cas d'un binôme, quelques questions seront posées lors de la dernière séance aux 2 membres du binôme individuellement pour estimer l'apport de chacun au projet. La note finale sera donc individuelle - sauf exception, ce sera la même pour les membres du binôme mais des écarts conséquents sont possibles.

Il faut rendre par mail le projet à **jean-francois.brette@parisdescartes.fr** au plus tard le lundi 18 juin à l'aube. Le mail devra avoir **UN** fichier compressé (.zip, .rar) attaché nommé des noms des membres du binôme (exemple : le binôme Dupont et Durand enverra un fichier **DupontDurand.zip**). Vérifiez que votre fichier fait une taille raisonnable (ne mettez pas apache-mina ou eclipse !) et ne mettez aucun .exe ce qui le bloquerait à la réception. Vous aurez de toute façon un accusé de réception.
Une remise non conforme (envoi en plusieurs mails, plusieurs fichiers attachés, un fichier nommé *projet.zip*, les .class et pas les .java, etc) sera refusée et entrainera un rappel avec 5 pts d'amende.
Le fichier .zip doit contenir tout le code source bien sur ainsi qu'une présentation (format pdf de préférence) de ce qui a été réalisé, ce qui marche, ce qui ne marche pas, ce qui pourrait être amélioré.