École de technologie supérieure
Département de génie logiciel et des TI

Trimestre : Automne 2014
Enseignants : R. Champagne
Y. Ross
Chargé de lab. : S. Djeffal

LOG430 - ARCHITECTURE LOGICIELLE

LABORATOIRE 2: LE STYLE ARCHITECTURAL INVOCATION IMPLICITE

Description du problème

Le but de ce laboratoire est de vous exposer aux architectures à invocation implicite, une spécialisation du style architectural "publication / abonnement". La première partie de ce laboratoire consiste à modifier une implantation existante et fonctionnelle, afin de vous permettre de faire le lien entre les concepts architecturaux et l'implantation. **Ce cours n'est pas un cours de programmation et l'emphase du laboratoire est mise sur les concepts architecturaux**. Le laboratoire est divisé en deux parties.

Pour la première partie, un système existant vous est fourni. Ce système fait partie d'un système plus large et consiste en un système élémentaire de gestion de projet. Il sert à assigner des ressources (humaines) à des projets de développement logiciel. Votre tâche pour la première partie consiste à modifier le code source du système original afin de satisfaire des nouveaux besoins, lesquels sont énumérés plus bas.

La seconde partie du laboratoire consiste à analyser la structure de ce système. Après avoir analysé et modifié le système, vous devrez répondre à des questions liées aux décisions de design que votre équipe a prises dans la première partie.

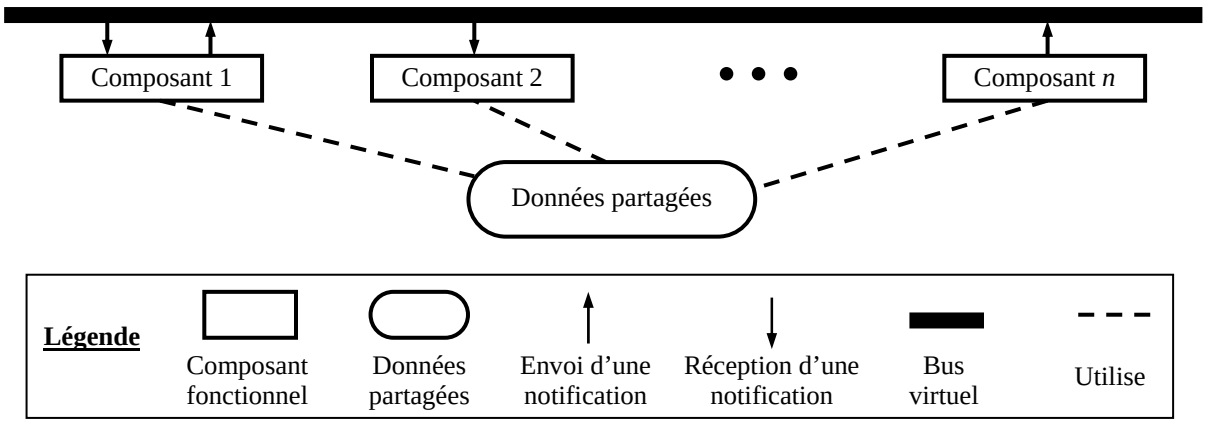
Notez que le système de départ qui vous est fourni est fonctionnellement équivalent à celui du laboratoire précédent, et les modifications à apporter au système original sont également fonctionnellement identiques à celles exigées au laboratoire précédent.

Fonctionnalité du système existant

Voir la donnée du premier laboratoire.

Architecture du système existant

Le système existant est caractérisé par une architecture à invocation implicite, où chaque composant est un fil d'exécution indépendant.



La fonctionnalité du système est répartie et encapsulée dans chacun des *composants*. Les composants écoutent le *bus virtuel* afin de savoir quand ils doivent livrer des services, ou ils notifient le bus virtuel afin d'obtenir des services des autres composants. Il est important de remarquer que certains composants ne font qu'envoyer des notifications, d'autres ne font qu'en recevoir, alors qu'un troisième groupe en envoie et en reçoit. L'implantation du système est réalisée en Java et utilise les classes `Observable` et `Observer`. L'état du système (listes de projets, de ressources, de composants) est maintenu via l'utilisation d'un objet de données partagées. Le système est initialisé par la classe principale `SystemInitialize`. Les fichiers suivants font partie de l'implantation qui vous est fournie :

`AssignResourceToProject.java` : Assigne une ressource à un projet.

`CommonData.java` : Contient les données partagées par tous les objets (liste de ressources, liste de projets, liste de composants). Fournit des services permettant d'enregistrer et localiser des composants pour la communication.

`Communication.java` : Fournit les fonctionnalités de base afin que les composants puissent être observateurs et observables. Toutes les communications inter-composant sont facilitées via cette classe. **Tous les composants du système (sauf `SystemInitialize`) doivent hériter de cette classe.**

`ComponentList.java` : Fournit une structure de données (`Vector`) pour stocker les composants du système.

`Project.java`: Objet représentant un projet dans ce système.

`ProjectList.java`: Fournit une liste d'objets de classe `Project`.

`ProjectReader.java`: Décode ("parse") les lignes de texte du fichier des projets, créé un objet de classe `Project` pour chaque ligne et retourne une liste de projets.

`Displays.java`: Affiche les diverses listes sur le terminal.

`Resource.java`: Objet représentant une ressource dans ce système.

`ResourceList.java`: Fournit une liste d'objets de classe `Resource`.

`ResourceReader.java`: Décode ("parse") les lignes de texte du fichier des ressources, créé un objet de classe `Resource` pour chaque ligne et retourne une liste de ressources.

`Executive.java` : Présente le menu principal et transmet les événements aux autres composants.

`LineOfTextFileReader.java`: Fournit divers services d'entrées/sorties liés aux fichiers. Lit des lignes de texte à partir d'un fichier.

`List.java`: Fournit les divers services et définitions liés aux listes. Les listes sont implantées dans ce système sous forme de vecteurs Java.

`ListProjects.java` : Affiche la liste des projets à assigner.

`ListProjectsAssignedToResource.java` : Affiche la liste des projets actuellement assignés à une ressource.

`ListResources.java` : Affiche la liste des ressources.

`ListResourcesAssignedToProject.java` : Affiche la liste des ressources assignées à un projet.

`Menus.java`: Affiche les menus sur le terminal.

`SystemInitialize.java`: Fichier principal (contient la méthode `main()`). C'est cette classe qui détermine la structure du système en instanciant les autres classes.

`Termio.java`: Contient divers services d'entrées/sorties pour le terminal.

Compilation et exécution du système original

Vous devez d'abord désarchiver le contenu du fichier comprenant le code source et placer son contenu dans votre répertoire de travail. Pour compiler le programme, ouvrez une fenêtre DOS, déplacez-vous dans votre répertoire de travail et invoquez la commande suivante (cet exemple suppose que votre répertoire de travail est `C:\lab2`):

```
C:\lab2> javac SystemInitialize.java
```

Une fois le programme compilé, vous pouvez l'exécuter en invoquant la commande suivante:

```
C:\lab2> java SystemInitialize
```

Note: Un fichier de projets par défaut vous est fourni et se nomme `projects.txt`. Un fichier de ressources par défaut vous est également fourni et se nomme `resources.txt`.

Partie 1: Modifications au système original

Voir la donnée du premier laboratoire.

Partie 2: Analyse architecturale

a) Élaborez les vues pertinentes pour le système modifié. Vos vues doivent comprendre plus de détails que le diagramme fourni dans ce document. Choisissez une notation ou un format de présentation qui décrit adéquatement ce système, par exemple: vue de composition (STAN), vue processus, diagramme d'états / de transition / de collaboration, etc.

   Vous n'êtes pas obligés d'utiliser une vue en particulier. Vous devez choisir ce qui doit être représenté en fonction des besoins du travail demandé. Réfléchissez à l'analyse que permet chacune des vues et justifiez votre choix de vues. Comme toujours, vous devez inclure une légende avec chaque vue.

b) Discutez toute déviation du système original par rapport à une architecture invocation implicite.

c) Comparez la vue de composition du système de ce laboratoire avec la vue de composition du système du laboratoire 1 en termes de dépendances. Discutez les ressemblances et différences.

d) Discutez comment les modifications apportées à ce système à invocation implicite sont différentes des modifications apportées au laboratoire #1 (en couches, orienté objet). Selon votre expérience à date, pour quels types de changements un système à invocation implicite est-il plus facile à changer que les architectures en couches? De la même manière, pour quels types de changements un système à invocation implicite est-il plus difficile à changer que les architectures en couches?

Critères d'évaluation

Votre solution ainsi que votre analyse seront corrigées selon les critères suivants:

- bon fonctionnement de votre implantation et satisfaction des exigences fonctionnelles;
- la qualité de vos programmes (commentaires, bonne structure);
- la qualité et le contenu de votre analyse, démontrant votre compréhension des implications des modifications sur l'architecture.

Plus spécifiquement, les points seront attribués comme suit:

Partie I - Implantation

- Nouveau système: 50 points.

Partie II - Rapport

- Question a): 15 points.
- Question b): 10 points.
- Question c): 10 points.
- Question d): 15 points.