Travaux Dirigés No1 – Composants

Frédéric Peschanski

30 janvier 2019

Exercice 1 : Communication événementielle

Dans ce TD, nous souhaitons tout d'abord concevoir un petit système de communication événementielle basé sur la diffusion (*broadcast*). Cela ressemble aux *event listeners* des Javabeans mais nous souhaitons que l'architecture du system utilise le *design pattern* «require/provide» vu en cours. Le principe de base est que les composants communiquent par événements. Pour chaque type d'événement on définit une classe dédiée. Un événement est généralement une structure de donnée *immutable*. Dans ce TD, les composants échangeront essentiellement des entiers, on utilisera donc la structure d'événement suivante :

```
import java.math.BigInteger;

public class IntEvent {
    private BigInteger val;
    public IntEvent(BigInteger val) { this.val = val; }
    public BigInteger getValue() { return val; }
}
```

Question 1 : interfaces de service

Définir l'**interface de service** pour la réception d'événements IntEvent.

Remarque : on exploitera également une interface pour les émetteurs, mais elle ne sera pas réifiée au niveau de service pour la composition. Il s'agit d'une interface «classique» en Java.

```
public interface IntEventSenderService {
    public void send(IntEvent event);
}
```

On utilisera également un second service permettant l'activation des composants à distance :

```
public interface ActivatorService {
    public void activate();
}
```

Question 2 : interfaces de liaison

Définir les **interfaces de liaison** pour les services de réception d'événement et d'activation à distance.

Question 3 : Composant émetteur

Le code de notre premier composant GenConst est donné ci-dessous.

Représenter le **diagramme de composant** correspondant à cet émetteur.

```
public interface Component extends ActivatorService {
    /* marker interface */
}

public abstract class SimpleGenerator implements Component,
        /* provide */
        IntEventSenderService,
        /* require */
        RequireIntEventReceiverService {

    protected List<IntEventReceiverService> receivers;

    protected SimpleGenerator() {
        receivers = new ArrayList<IntEventReceiverService>();
    }

    @Override
    public void bindIntEventReceiverService(IntEventReceiverService serv) {
        receivers.add(serv);
}

    @Override
    public void send(IntEvent event) {
        for(IntEventReceiverService receiver : receivers) {
            receiver.onIntEvent(event);
        }
    }
}

public class GenConst extends SimpleGenerator {
    private BigInteger constValue;

    public GenConst(BigInteger constValue) {
        this.constValue = constValue;
    }

    public GenConst(int constValue) {
        this(BigInteger.valueOf(constValue));
    }

    @Override
    public void activate() {
        send(new IntegerEvent(constValue));
    }
}
```

Question 4 : Composant récepteur

Définir un composant récepteur d'événements IntEvent. Ce composant Printer, à chaque activation, affiche la valeur du dernier événement reçu, ou un point «.» si aucun événement n'a été reçu précédemment.

Question 5 : Composition

Soit le programme suivant :

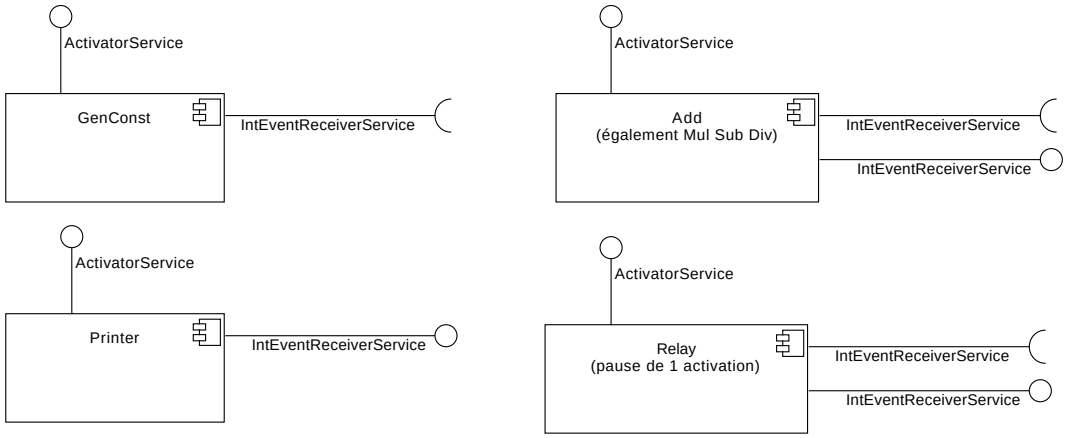
```
public class Composition {
    public static void main(String[] args) {
```

```
        GenConst gen = new GenConst(1);
        Printer printer = new Printer();
        gen.bindIntEventReceiverService(printer);
        // Point (1)
        for(int i = 0; i < 10; i++) {
            gen.activate();
            printer.activate();
        }
    }
}
```

— que produit ce programme ?
— décrire le **diagramme de composition** au point `(1)` du programme.

Exercice 2 : Dataflow

On souhaite dans cet exercice manipuler une petite boîte-à-outils (*framework*) de *dataflow* construite à partir des briques élémentaires suivantes :



Question 1 : Génération des entiers naturels

On souhaite définir un dataflow pour générer et afficher les entiers naturels successifs (1, 2, ...) à partir de la bibliothèque de composants élémentaires.

Donner la réponse sous la forme d'un diagramme de composition.

Question 2 : Composant composite

Pour réifier le générateur d'entiers naturels (sans la partie affichage) au statut de composant à part entière, on introduit la notion de *composant composite*.

Pour cela, on utilise simplement une interface marqueur :

```
public interface Composite extends Component {
    /* marker interface */
}
```

Dans le constructeur du composant composite, on construit la composition qui réalise le générateur d'entier. De plus, on précise :

— les liaisons au niveau composite, en général en reliant un composant interne au composite
— la procédure d'activation, qui doit activer les sous-composants dans le bon ordre.

Donner le code source ainsi que le **diagramme de composite** du composant GenInt.

Remarque : dans toutes les questions qui suivent, on construira systématiquement un composant composite pour répondre à la question. On ne pourra utiliser que les composants élémentaires ou les composants composites des questions précédentes.

Question 3 : Composant factorielle (TME)

Concevoir et implémenter un composant composite GenFact permettant d'énumérer la factorielle : (1!, 2!, 3!, ...).

A rendre :

— un fichier GenFact.uxf avec le diagramme de composite pour le composant GenFact
— le code source complété dans src/dataflow/examples/fact/GenFact.java

Question 4 : Composant fibonacci (TME)

Concevoir et implémenter un composant composite GenFibo permettant d'énumérer les éléments de la suite de Fibonacci : (1, 1, 2, 3, 5, 8, ...).

A rendre :

— un fichier GenFibo.uxf avec le diagramme de composite pour le composant GenFibo
— le code source complété dans src/dataflow/examples/fibo/GenFibo.java

Conseil : on pourra exploiter des composants de type Relay pour répondre à la question.

Question 5 : Crible d'Eratosthène (TME)

Le crible d'Eratosthène consiste à énumérer les nombres premiers en évitant de tester la divisibilité par autre chose qu'un nombre premier. Pour simplifier, on ne considérera que les nombres premiers strictement supérieurs à 1.

En terme de dataflow, l'idée est de partir d'un générateur d'entiers GenInt et de produire une chaîne dynamique de composants PrimeFilter dont le rôle est double :

— si le composant PrimeFilter est en bout de chaîne – on notera $\text{PrimeFilter}_\perp$ – et qu'il reçoit un certain entier n alors, après activation, le composant "sait" que n est premier. Ce composant $\text{PrimeFilter}_\perp$ devient alors PrimeFilter_n et il crée un nouveau composant $\text{PrimeFilter}_\perp$ qui va servir de nouveau bout de chaîne.
— si le composant est avant le bout de chaîne, alors il s'agit d'un PrimeFilter_k pour un certain nombre premier k précédemment détecté. Lorsque ce composant reçoit un entier n (et qu'il est ensuite activé) son rôle est de vérifier si n est divisible par k ou non.
  — si k divise n alors on sait que n n'est pas premier, on stoppe donc sa propagation dans la chaîne,
  — si en revanche k ne divise pas n, alors on transmet n au prochain composant PrimeFilter (sans oublier de l'activer).

Réaliser ce dataflow dynamique avec la boîte à outils proposée.

A rendre :
— le code source complété dans src/dataflow/examples/primes/PrimeFilter.java

Question 6 : Parallélisation du Cribble (TME)

Le dataflow proposé à la question précédente offre un potentiel intéressant de parallélisation. En effet, il est possible de faire remonter plusieurs entiers successifs en parallèle dans la chaîne. Plus la chaîne est longue et plus le potentiel de parallélisation augmente. Proposer une variante du crible en synchronisant n threads dans la chaîne (on pourra tester avec $n = 4$).

Annexe : Consignes pour le TME

Dans un premier temps, vous récupérerez le squelette du projet Dataflow sur le site de l'UE : TME1-Dataflow-skel-<date>.jar

Dans un répertoire de votre choix (exemple : CPS/TME1/dataflow) vous décompresserez l'archive.

```
$ jar xvf TME1-Dataflow-<date>.jar
...
```

Pour tester l'archive, vous pourrez lancer le script ant :

```
$ ant compile
...
$ ant run
...
```

Pour éditer les fichiers, il sera utile d'importer le projet sous eclipse (les fichiers .project et .classpath sont présents dans l'archive).

Après avoir répondu aux questions, vous devez soumettre une archive du projet sur le site de l'UE (page de soumission). Pour cela, vous devez utiliser la cible dist du script ant après avoir indiqué les noms des binômes concernés.

Voici les lignes concernées dans le fichier build.xml

```
<!-- METTRE ICI LE NOM DU BINOME -->
  <property name="nom1" value="XXXXX"/>
  <property name="nom2" value="YYYYY"/>
```

Pour générer l'archive, la commande est la suivante :

```
$ ant dist
...
```

IMPORTANT : les diagrammes de composants et de composition au format UXF (générés par UMLet) seront à placer à la racine du projet (au même niveau que le build.xml. Ils seront donc intégrés à l'archive.

Si tout se passe bien, un fichier de la forme TME1-Dataflow-<NOM1>-<NOM2>-<DATE>.jar sera généré dans le répertoire parent. Ce fichier est à soumettre sur la page de soumission.