Correction du partiel de Génie Logiciels et Systèmes session 1 2014

Exercice 1

1

- Diagramme de classe : description statique de l'organisation des classes.
- Diagramme d'instance : instances du diagramme de classe (éléments d'un model)
- Diagramme de cas d'utilisations : modélisation des fonctions du système
- Diagramme de séquence : déscription des interactions entre objets dans un scénario) en privilégiant la structure temporelle
- Diagramme de machine à états : description du comportement d'une classe différents états possibles de cet objet, transitions entre états, séquencemennt des opérations
- Diagramme d'activité : décrire les activités d'un processus

2

M2M (model to model) : transformation d'un model vers un autre (donc entre deux méta modèles différents).

M2T (model to text) : transformation d'un model en texte.

3

Effectuer des transformations entre modèles. Il est dédier à effectuer de telles transformations, sa syntaxe est adaptée donc plus simple.

Exercice 2

1

Patrons généraux

- Patrons fondamentaux
    - Délégation : utiliser un objet comportement plutôt que d'implémenter ce comportement
    - Interface : rendre le client indépendant de la classe instanciant ce comportement
    - Abstraite : factorisation de logiques communes à plusieurs classes
    - Immuable (const) : robustesse des objets partageant l'objet immuable nottament lors de l'utilisation parallèle
    - Interface de marquage : définit une propriété sémantique booléenne (par exemple Serializable)
- Patrons créateurs
    - Builder : discocier la construction d'un objet complexe de sa représentation
    - Factory (constructeur polymorphe) : construire des objets de plusieurs types
    - Factory abstraite : -
    - Prototype : clonage (par exemple depuis une interface de création de diagrammes UML on crée un classe en clonant celle présente dans la barre d'outils)
    - Singleton : garantir qu'une classe n'a qu'une seule instance
    - Adapter (wrapper) : changer un interface en une autre sans modification des fonctionnalités)
    - Pont (poignée) : séparer l'objet réel de celui utilisé
    - Composite : composant - composé (relation de composition) peut permettre une structure arborescente mais pas nécessairement
    - Décorateur : Attacher dynamiquement des responsabilité à un objet (wrap qui rajoute des fonctionnalités)
    - Façade : wrapper un objet par un autre dont les fonctionnalités sont de plus haut niveau
    - Proxy : wrapper un objet en controllant ses fonctionnalités
- Patrons comportementaux
    - Caîne de responsabilité : -
    - Commande : -
    - Interpréteur : -
    - Itérateur : Fournir un moyen pour accéder séquentiellement à chacun des éléments d'un agrégat d'objets sans en révéler la représentation interne
    - Médiateur : -

    - Mémento : -
    - Observateur : Définir une interdépendance de type un à plusieurs de sorte que quand un objet change d'état tous ceux qui en dépendant soient avertis
    - Etat : -
    - Stratégie : Définir une famille d'algorithme, encapsule chacun d'entre eux et les rend interchangeable, les algorithmes peuvent évoluer indépendamment de leurs clients
    - Patron de méthode : -
    - Visiteur : Modéliser une classe par une opération applicable aux éléments d'une structure d'objets et permettre de définir de nouvelles opérations sans modifier les classes de la structure

Ici on a :

- Composite (Container)
- Stratégie (Layout)
- Façade (IHM)

2.1

Faire le diagramme d'instance correspondant et expliquer les correspondances avec le méta modèle.

2.2

Sirius Black détient la réponse.

3

```
inv self.lineCount >= 0 && self.columnCount >= 0;
```

```
inv self.components->forAll(c1, c2 | c1 <> c2 implies c1.name <> c2.name);
```

```
inv self.layout
    ->oclIsKindOf('GridLayout')
    implies self.components->size() <=
        self.layout.lineCount * self.layout.columnCount;
```

4.1

Example1.java

```
import javax.swing.*;
import java.awt.*;

public class Exemple1 extends JFrame {
    private JPanel p1 = new JPanel();
    private JTextArea messages = new JTextArea(10, 50);
    private JPanel exemple1 = new JPanel();
    private JTextField text = new JTextField(40);
    private JButton go = new JButton("Go");

    public Exemple1() {
        p1.setLayout(new FlowLayout());
        p1.add(text);
        p1.add(go);
        exemple1.setLayout(new BoxLayout(exemple1, BoxLayout.Y_AXIS));
        exemple1.add(p1);
        exemple1.add(messages);
        this.getContentPane().add(exemple1);
        this.pack();
        this.setVisible(true);
    }
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Exemple1();
            }
        });
    }
}
```

4.2

Ligne 39 du Listing 1 (entre autre) on fait appel à selectByKind sur components alors que celui-ci pourrait être null car quand il y a 0 Component dans la relation on ne sait pas si le

conteneur est initialisé (tableau vide) ou non (null).

5.1

ihm : on déclare une instance d'ihm

example1 : on défini son nom

root : on dit qu'on va définir son conteneur racine

main : main sera le nom du conteneur racine

layout : on va définir le layout du conteneur principal

Box : il sera de type BoxLayout

{…} : contiendra les éléments dans le layout

…

Définition d'une IHML

```
ihm <name> root <root container>
```

Définition d'un container

```
<name> = layout <container layout> {
    <components>
}
```

Définition d'un component

```
<name> = <component type> <args>
```

5.2

ihm2xtext

```
grammar fr.enseeiht.ihm with org.eclipse.xtext.common.Terminals

generate ihm "http://www.enseeeiht.fr/ihm"

IHM :
    'ihm' name=STRING 'root' root=[Container]
    ;

Component :
    TextArea
    | TextField
    | Label
    | Button
    | Container
    ;

TextArea :
    name=STRING '= TextArea' height=INT width=INT
    ;

TextField :
    name=STRING '= TextField' size=INT
    ;

Label :
    name=STRING '= Label' text=STRING
    ;

Button :
    name=STRING '= Button' text=STRING
    ;
```

```
Container :
    name=STRING '= layout' [Layout] '{'
        components+=Component*
    '}'
    ;

Layout :
    FlowLayout
    | BoxLayout
    | GridLayout
    ;

FlowLayout :
    'Flow'
    ;

BoxLayout :
    'Box'
    ;

GridLayout :
    'Grid'
    ;
```

6.1

Créer une interface VisiteurIHM qu'implémenteront les visiteurs de notre méta modèle.

```
interface VisiteurIHM {
    void visit(IHM ihm);
    void visit(Component c);
    void visit(Layout l);
}
```

Dans chacune des classes du méta modèle rajouter une méthode ‘accept’ qui permet aux objets de type ‘VisiteurIHM’ de venir les visiter :

```
public void accept(VisiteurIHM v) {
    v.visit(this);
}
```

Ecrire les visitieurs en les faisant implémenter l’interface VisiteurIHM.

6.2

VisiteurCountComponents.java

```
public class VisiteurCountComponents implements VisiteurIHM {
    int count;

    public VisiteurCountComponents() {
        count = 0;
    }

    public void visit(IHM ihm) {
        ihm.getRoot().accept(this);
    }

    public void visit(Layout l) {
        count += 0;
    }

    public void visit(Component c) {
        count += 1;
    }

    public void visit(Container c) {
        // count += 0;

        for(Component cb : c.getComponents()) {
            cb.accept(this);
        }
    }
}
```