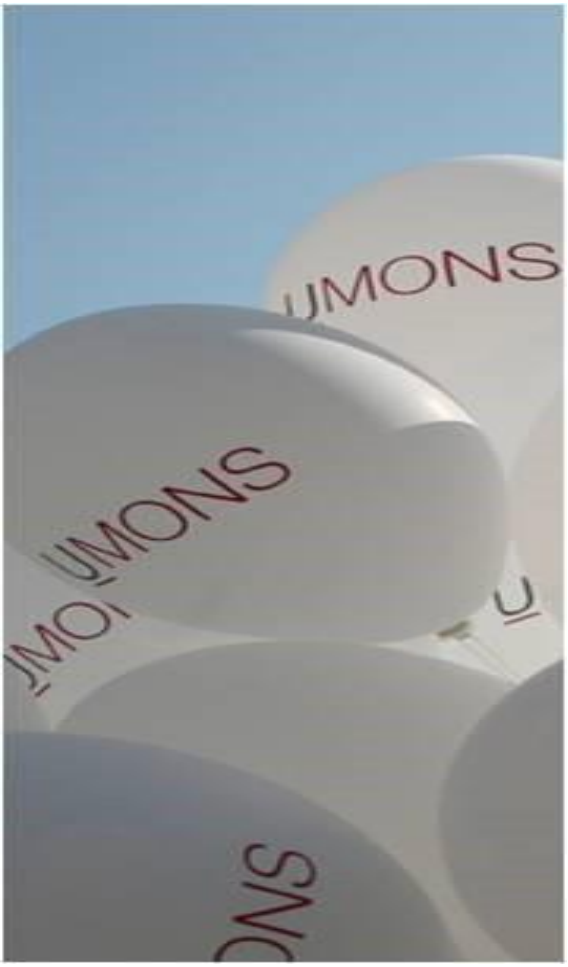
UMONS
Université de Mons

Faculté des Sciences



Développement dirigé par modèles

Séance 1 - Statecharts

Rapport
23 novembre 2021 à 11h (du matin)

Prénom NOM

Faculté des Sciences

Sous la direction de :
Prof. Stéphane DUPONT
Ass. Bastien VANDERPLAETSE

Année académique 2021-2022

1 Génération de code avec Yakindu

Question 1 Observez le code généré par Yakindu à l'issue de l'exercice 1. Comment les états, les actions et les transitions sont représentés dans le code? Comment ces objets interagissent-ils ensemble?

- **Un enumérateur énumère tous les états**
- **Un tableau de taille 1 représente l'état actif**
- **L'application commence avec l'état par default (ici DisplayHour)**
- **B1 est une variable booléenne**
- **Lorsque l'on appuis sur b1 une méthode avec l'état actuel en paramètre est appelé et un switch permet de réagir en fonction de l'état donner, une autre fonction est appelé et quitte l'état actuel pour entrés dans l'état suivant.**
- **C'est une fonction conditionnelle qui permet de dire qu'on a cliqué sur b1**

Question 2 Comparez le code généré par Yakindu avant et après avoir ajouté l'*event b2*. Comment cette action a-t-elle été ajoutée dans le code et quels sont les changements impliqués?

- **Des fonction conditionnelle (if else) sont imbriqué avec comme condition b1 ou b2, selon quel bouton est cliqué ça sera une transition différente**

Question 3 Comparez le code généré par Yakindu avant et après avoir ajouté les états *SetDate*, *SetHour* et *SetSecond*. Comment ces états ont-ils été ajoutés dans le code et quels sont les changements impliqués?

- **L'énumérateur a maintenant 3 valeurs en plus**
- **Le switch a donc 3 condition supplémentaire**

Question 4 Comparez le code généré par Yakindu avant et après avoir ajouté l'*event b3*. Comment cette action a-t-elle été ajoutée dans le code? Comment les fonctions servant à incrémenter les valeurs ont-elles été ajoutées dans le code?

- **Une condition supplémentaire dans la structure conditionnelle.**
- **Dans les méthodes react des états SET, la condition impliquant b3 quitte l'état, incrémente de 1 la valeur impliquée et retourne dans le même état.**

2 Comparaison des stratégies d'implémentation

Question 1 Laquelle des stratégies (hors Yakindu) vous a semblé la plus difficile à modifier pour ajouter des actions et des états? Pourquoi? Laquelle a necessité le plus de temps?

La méthode TableState, car c'est celle qui ajoute le plus de classe et donc plus de code, ce qui prend beaucoup de temp

Question2 Si vous deviez implémenter un projet impliquant une *statechart*, laquelle de ces trois stratégies utiliseriezvous? Pourquoi?

La méthode State pattern car c'est celle qui m'a demandé le moins de temps pour faire les modifications, car moins de code à rajouter et c'est aussi la méthode avec laquelle je suis le plus à l'aise car déjà utilisé

Question 3 Imaginez un projet, contenant une statechart, qui soit favorable à l'utilisation de la stratégie basée sur le *State Design Pattern*. Quelles seraient les caractéristiques de ce projet et pourquoi ces caractéristiques seraient idéales pour l'utilisation de cette stratégie? Faites de même pour la stratégie basée sur le *switch case* et pour celle basée sur la *table state.*

Lorsqu'on a beaucoup d'état, je dirais que la stratégie serait le state pattern car pour chaque état on aurait qu'à rajouter une classe

Lorsqu'on a beaucoup de transition (d'évènement), la meilleure stratégie serait plus la méthode des switch case, car lorsqu'on a un évènement à rajouter, au lieu de rajouter une méthode dans chaque classe, on peu tous simplement rajouter des cas dans un switch

Pour la méthode des tableaux, je ne sais pas trop dans quelle situation l'utiliser car chaque états et chaque évènement demande d'ajouter une classe, et on se retrouve très vite avec beaucoup de classe pour pas grand-chose.