

Finite State Machine: TD numéro 1, et plus

Premiers Pas

Objectif

L'objectif de ce TD est de prendre en main l'environnement minimal de développement que l'on va utiliser pour investiguer le pourquoi et comment de l'utilisation de state machines dans le métier d'ingénieur. De multiples outils seront utilisés lors des différents TDs. Afin de ne pas prendre un environnement trop académique je vous propose d'installer Yakindu State Chart¹. L'idée est de vous faire faire trois versions pour résoudre un même problème, le premier en vous laissant faire naturellement, la seconde en vous forçant à utiliser une machine à état et la troisième en vous faisant coder la machine à état.

Installation de Yakindu

Toutes les informations se trouvent ici: <https://www.itemis.com/en/yakindu/state-machine/>. Il va falloir répondre à quelques questions et éventuellement créer un compte avec votre adresse *univ-cotedazur* afin d'avoir une license académique. Pendant le téléchargement, commencer à lire et réfléchir au sujet.

Développement d'une application simple: Chronomètre

Votre but est de développer une application représentant un chronomètre simple. Un chronomètre est utilisé pour mesurer une durée. Il possède un bouton pour démarrer et arrêter le compteur et un autre pour mettre en pause le défilement du temps sur l'affichage (aussi connu sous le nom de "split", où le compteur continue de garder le temps alors que l'affichage est arrêté sur le temps lors de l'appui sur pause. Cela sert, par exemple à mesurer le temps de chaque tour lors d'une course). Lorsque le chronomètre a été arrêté, l'affichage reste figé sur le temps lors de l'appui sur le bouton. Le bouton start/stop sert alors à remettre l'affichage à 0.

Une squelette simple d'implémentation en JavaFx vous est fourni sur ma page web. Vous pouvez modifier le code comme bon vous semble mais le fonctionnement doit être celui décrit ci dessus. Pour l'importer, le plus simple est de démarrer sur un nouveau workspace puis de faire clic droit -> import -> General -> Existing project into workspace.

Version 1

Dans la première version je vous laisse implémenter le comportement simple de l'application chronomètre par vous même. Vous avez le droit de modifier tous les fichiers, de rajouter des attributs, des méthodes, etc à la classe existante. Vérifiez bien que vous avez le comportement souhaité, **que vous me ferez valider avant de continuer.**

¹Dans mon travail en tant que chercheur, je ne travaille ni dans les interfaces graphiques ni avec JavaFx, comprenez donc que le but n'est ni de devenir pro de JavaFx ni de faire de jolies interfaces



Figure 1: Différentes vues d'une application chronomètre simple

Version 2

Yakindu fournit un éditeur graphique pour l'édition de State charts ainsi qu'un simulateur et un générateur de code. J'ai modifié le générateur de code pour corriger certains problèmes. Afin d'installer la version modifiée, il faut dézipper le zip nommée 'dropins.zip' qui se trouve sur ma page web, dans le répertoire d'installation de Yakindu. Vous repartirez pour cette version de la version initiale qui vous a été fournie. Vous pourrez ensuite faire clic droit → new → StateChart Model.

Vous pouvez maintenant définir une machine à état permettant de définir le comportement souhaité de l'interface. Ceci se fera en deux temps: Premièrement, ne vous préoccupez pas du code et définissez uniquement la machine à état. Une fois que vous pensez que c'est correct, faites moi valider.

Notez que vous pouvez simuler votre state Machine avec un retour graphique. Ecosia, Qwant ou Google sont vos amis si vous ne trouvez pas comment faire...

Ensuite, vous lierez cette machine à état à du comportement dans votre code. Pour ce faire, voici la marche à suivre:

- assurez vous que la machine à état soit correctement compilée (clic droit sur le fichier .sct → new code generator model. Au lieu de *yakindu::java* utilisez *yakindu::deantoni::java*. Puis src-gen doit être défini comme un dossier contenant des sources java. ;
- ajoutez un attribut de type "leNomDeVotreStateMachine"+"StateMachine" à la classe *StopWatchGui*. Enfin vous pouvez initialiser la machine à état comme suit:

```
theFSM = new StopWatchStatemachine();
theFSM.init();
theFSM.enter();
```

ou comme ceci si vous utilisez des transitions temporisées:

```
theFSM = new StopWatchStatemachine();
TimerService timer = new TimerService();
theFSM.setTimer(timer);
theFSM.init();
theFSM.enter();
```

- connectez des "bouts de code" à l'évolution de la machine à état. Pour ce faire vous avez deux primitives:

- `theFSM.getSCInterface().raiseNameOfAnInputEvent();` //send the ``NameOfAnInputEvent``
//input event to the State Machine

- Afin de réagir aux événements générés par votre state machine, vous pouvez avoir une méthode comme celle ci:

```
SCInterface.start.subscribe(event -> doStart())
```

à noter, pour être propre, ajoutez toutes les souscriptions à une collection (e.g., *allSubscriptions* = *new ArrayList<>()*; *allSubscriptions.add(SCInterface.start.subscribe(event -> doStart()));*) et dans la méthode stop de l'interface (appelée par le framework javafx), faites:

```
@Override
public void stop() throws Exception {
    allSubscriptions.forEach(s -> s.unsubscribe());
    theFSM.exit();
    super.stop();
}
```

Encore une fois vous pouvez (devez) ajouter ce que vous voulez au code existant du moment qu'il a le comportement souhaité.

Version 2: Mealy ou Moore ?

Réfléchissez. Avez vous fait une machine de Mealy, de Moore ou un mixte des deux ? Faites une deuxième version distincte de votre state machine afin d'avoir une version de Mealy et une de Moore. Que pouvez vous en dire ?

Version 3

Vous allez ajouter du comportement à la version 2. Il s'agit ici d'ajouter un troisième bouton "mode". Ce bouton permet d'afficher l'heure ou la date pendant 1 seconde. Au premier appuie sur le bouton, l'affichage devra montrer l'heure courante. Au bout d'une seconde, si le bouton n'a pas été appuyé à nouveau, le chronomètre retourne dans l'état où il était. Si le bouton est appuyé à nouveau avant 1 seconde, l'affichage montrera la date. De même, si aucun bouton n'est appuyé pendant 1 seconde on retourne au comportement initial, sinon on remontre l'heure courante. Le fonctionnement décrit sera actif peu importe l'état dans lequel se trouve le chronomètre (arrêté, démarré, en pause, etc).

Version 4

Vous allez ajouter du comportement à la version 2 (peu importe que celui-ci soit très pertinent, c'est pour l'exemple). Il s'agit ici d'ajouter un troisième bouton et 5 labels. Ce bouton permettra de gérer les nouveaux labels. Le fonctionnement suivant sera actif peu importe l'état dans lequel se trouve le chronomètre (arrêté, démarré, en pause, etc). Lors d'un appuie sur le bouton, le temps courant sera stocké dans le prochain label libre, ou s'il n'y en a plus, dans le premier à avoir été utilisé. Si le bouton est appuyé deux fois de suite dans un intervalle de 1 seconde alors tous les labels sont réinitialisés.

Bien sûr l'implémentation utilisera la machine à état et mettra en œuvre les constructions vues en cours.

Version 5

Vous allez repartir du squelette de code qui vous a été fourni initialement. Maintenant que vous avez décrit la machine à état (de la version 2), vous allez l'implémenter manuellement. Essayez de structurer votre code au mieux !

Qu'en est-il si vous deviez implémenter manuellement la version 3 ? La 4 ?

What Next ?

L'année dernière j'ai laissé 4 séances pour réaliser un système un peu conséquent impliquant principalement du contrôle en utilisant des State Machine. J'avais demandé le contrôle d'un robot (avec le simulateur VREP). Des objections ? remarques ?