ÉCOLE POLYTECH'NICE SOPHIA
SI4
2021-2022

Julien Deantoni

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 Chart. 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. Notez que si vous ne demandez pas la license aujourd'hui vous risquez de ne plus pouvoir travailler dans quelques semaines (et donc vous risquez de ne pas pouvoir faire un rendu en fin de séance...) Pendant le téléchargement, commencez 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'appuie sur le bouton. Le bouton start/stop sert alors à remettre l'affichage à 0.
Une squelette simple d'implémentation en Java 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. Verifiez bien que vous avez le comportement souhaité, **que vous me ferez valider avant de continuer**.

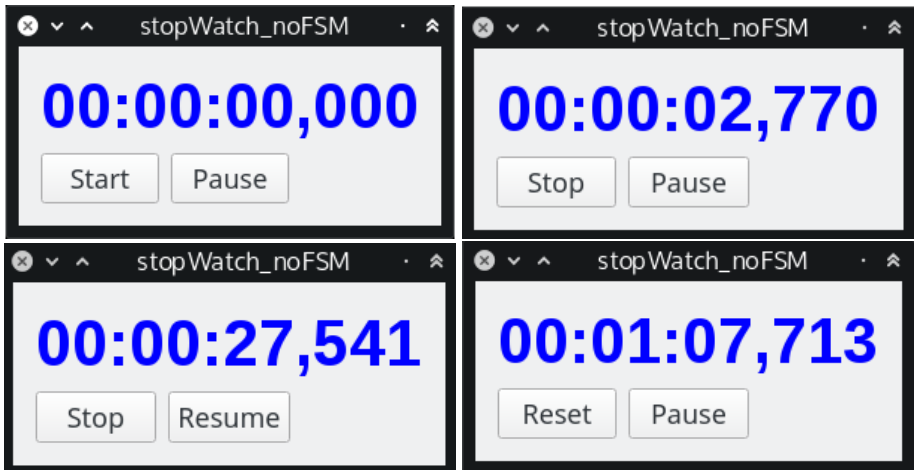

Figure 1: Différentes vues d'une application chronomètre simple

Version 2

Yakindu fourni un éditeur graphique pour l'édition de State charts ainsi qu'un simulateur et un générateur de code. Vous repartirez pour cette version de la version initiale qui vous a été fournie. Vous pourrai 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 a état soit correctement compilée (clic droit sur le fichier .sct → new code generator model. Utilisez *yakindu::java*. Puis src-gen doit être défini comme un dossier contenant des sources java. ;
- ajoutez un attribut de type "leNomDeVotreStateMachine" à la classe *StopWatchGui*. Enfin vous pouvez initialiser la machine a états comme suit:

  ```
  theFSM = new StopWatchStatemachine(); //for instance
  theFSM.enter();
  ```

  ou comme ceci si vous utilisez des transitions temporisées:

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

  Dans tous les cas, pensez à modifier votre fichier *sgen* en rajoutant ceci:

  ```
  feature OutEventAPI {
     observables =  true
     getters =  false
  }
  ```

```
feature Naming {
   libraryPackage =  "com.yakindu.core"
   basePackage =  "fr.unice.polytech.si4.fsm.stopwatch"
   typeName =  "StopWatchStatemachine"
}

feature GeneralFeatures {
   RuntimeService = false
   TimerService = true
   synchronized = true
   runnable = false
}
```

- connectez des "bouts de code" à l'évolution de la machine à état. Pour ce faire vous avez deux cas de figure. Le premier pour envoyer des événements vers la state machine:
  - `theFSM.raiseNameOfAnInputEvent(); //send the ``NameOfAnInputEvent'' //input event to the State Machine`
  - Le deuxième pour réagir aux événements générés par votre state machine (par un raise myEvent dans la state machine). Dans ce cas vous devez implémenter l'interface observer et l'ajouter aux observers de votre state machine; par exemple:

    ```
    package yourpackage;

    import com.yakindu.core.rx.Observer;
    //observer dedicate to the XX event
    public class MyObserver implements Observer<Void>{

        @Override
         public void next(Void value) {
           System.out.println("this is a reaction to a XX event");
         }

    }
    ```

    puis vous ajouter cette classe comme observer de l'événement désiré (ici XX):

    ```
    theFSM.getXX().subscribe(new MyObserver());
    ```

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 !
Continuez et implémentez maintenant la version 3 puis la 4, puis la "7"(combinant les fonctionnalités de la 3 et la 4).

What Next ?

Les années précédentes j'ai laissé 4 séances pour réaliser un système un peu conséquent impliquant principalement du contrôle en utilisant des State Machine. J'avais demandé par exemple le contrôle d'une machine à café ou encore le contrôle d'un robot (avec le simulateur VREP). Des envies ? objections ? remarques ?