Manuel d'utilisateur

Ce document d'écrit la manière d'utiliser et de compiler le générateur de code. Assurez-vous d'avoir installé la dernière version de java et la dernière version de maven pour que le générateur puisse s'exécuter.

Voici l'environnement utilisé pendant le développement du programme:
- macOs Sierra 10.12.4
- Java 1.8.0_121.
- Maven 10.12.4

Comment exécuter le générateur?

Le fichier generate.sh fournis à la racine du dossier, est le script pour lancer le générateur. Quelques options est fournis pour exécuter le programme. Les options fournis sont les suivantes :

- -f [file path] : cette option est **obligatoire**. Utilisez cette options pour indiquer le chemin de votre fichier scxml.

- -n [file name] : cette option n'est pas obligatoire. Utilisez cette option pour donner le nom au fichier généré.

- -w [t|f] : cette option n'est pas obligatoire. Utilisez cette option pour générer aussi le workspace.

Avant d'exécuter le script generate.sh, assurez-vous que le dossier *statemachine* est present à la racine du dossier (au meme endroit que le dossier d'entrée).

Voici un exemple d'exécution du script generate.sh.

```
Danials-MacBook-Pro:SI4-S8-FSM danial$ bash generate.sh -f resources/
door.scxml -n MyDoor -w t
Compiling the Generator ...
done
Danials-MacBook-Pro:SI4-S8-FSM danial$
```

Exemple d'execution du fichier generate.sh

Le contenu du dossier statemachine après avoir exécuter le script generate.sh précédent

Comment exécuter le compilateur?

Le fichier *run.sh* fourni à la racine du dossier, est le script pour lancer le compilateur de state machine. Lorsque le fichier généré et le fichier de “work space” sont bien present dans le dossier *statemachine*, exécutez le script *run.sh* en précisant la classe main du programme.

```
Danials-MacBook-Pro:SI4-S8-FSM danial$ bash run.sh WorkSpace
Compiling FSM ...
Executing FSM …
Open door
Close door
Danials-MacBook-Pro:SI4-S8-FSM danial$
```

Exemple d’execution du fichier run.sh

Fonctionnalités supportées

- États simple
- Transition d’un état vers un autre état lors du déclenchement d’un événement
- Support des événements du type *raise* et *send*
- Les actions *onentry* et *onexit* pour état et transition
- Gestion des états hiérarchiques
- “Compound States” : l’exécution de l’événement de l’état parent si l’événement n’est pas présent à l’état courant
- Support de la balise *log* dans les balises *onentry* et *onexit*
- Support de l’état final

Contraintes du programme

1. Fonctionnalités :

- Le programme ne support pas complètement la transition du type *internal*
- Le programme ne support pas l’état parallel
- Le programme ne gérer pas l’état historique
- Le programme ne prend en compte que un seul balise *log* dans chaque balise *onentry* et *onexit*
- Le programme ne support pas des transitions temporisées

2. Fichier SCXML :

Le fichier scxml donné doit obligatoirement suit le format donné sur le site w3.org. En plus, pour que le compilateur de state machine puisse s’exécuter, il faut que le fichier scxml indique l’état initial de la machine dans la balise scxml.

```
<scxml initial=“State1"/>
```

Lorsque vous avez des états hiérarchique dans votre state machine, assurez vous d’avoir mis l’attribut *initial* pour indiquer le premier fils de l’état courant.

```
<state id="State1" initial=“State3"/>
```

3. Squelette du “work space” du programme :

Le “work space” du programme doit suivi la squelette suivant :

- La classe doit extends la classe finit.state.machine.workspace.MyWorkSpaceImpl
- La méthode *start(this, new MyStateMachine())* doit être appelé dans le constructeur de la classe en donnant en paramètre la classe courante et la classe state machine générée.

```
import finite.state.machine.workspace.MyWorkSpaceImpl;

public class WorkSpace extends MyWorkSpaceImpl {

	public WorkSpace() {
		start(this, new MyStateMachine());
	}

	public static void main(String[] args){

	}
}
```

Exemple d’implémentation de base du programme

4. Déclaration des événements

Les événements sont représenter sous forme des méthodes dans ce programme. Les méthodes doivent être du type void et n’acceptent aucun argument. Une fois que les méthodes sont écrites, utilisez l’annotation *@FSMEvent* pour indiquer au programme que cette méthode est un événement et spécifiez le nom de l’événement dans l’argument *event* de l’annotation.

```
@FSMEvent(event="print-hello")
public void print_hello(){
    System.out.println("Hello");
}

@FSMEvent(event="print-bye")
public void print_bye(){
    System.out.println("Bye");
}
```

Exemple de déclaration d’un événement

5. Invocation des événements

Pour invoquer des transition, utilisez la méthode *submitEvent()* de la classe *MyWorkSpaceImpl* en donnant le nom de transition visée dans l’argument.

```
public static void main(String[] args){
    MyStateMachine stateMachine = new MyStateMachine();
    stateMachine.submitEvent("Button1");
    stateMachine.submitEvent("Button2");
}
```

Exemple d’invocation d’une transition

Test

Deux types des tests ont été effectués sur le programme.

D'abord les tests sur le code générer pour être sur que le code générer est syntaxiquement correct et que le code générer correspond bien au state machine voulu.

Deuxièmement, les test sur le scenario principale d'un state machine. Le scenario testés sont les suivants :

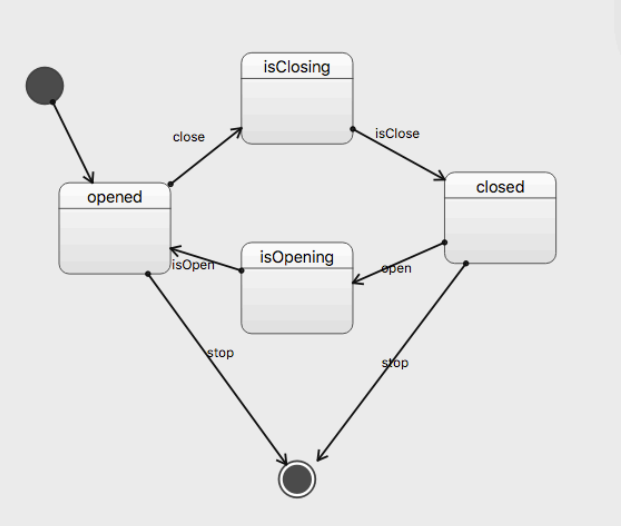
- Simple : Lorsque un événement de transition est invoqué, le test vérifie si l'état courant correspond bien à l'état destination de la transition
- Raise : Lors d'un appel d'un événement du type *raise*, le test vérifie si l'événement est effectué à la fin une fois que l'ensemble des événements courants ont été lancés
- Onenter / Onexit : Le test vérifie si l'événement *onexit* est effectue lorsque le programme sort de l'état ainsi que l'événement *onentry* lorsque le programme rentre à l'état.
- Hiérarchique : Dans un état hiérarchique ou le nom de l'événement de l'état courant est égal à celui de l'état parent, le test vérifie si le programme prend en compte d'abord l'événement à l'état courant. Si l'événement n'est pas présent à l'état courant, le programme va monter dans l'hiérarchie du l'état pour chercher l'événement correspondant.

Domain d'application

2 exemples d'applications de ce compilateur est fourni dans le dossier Application. Pour executer ces exemples, il suffit juste de compiler le fichier *run.sh* fourni sans donné d'argument.

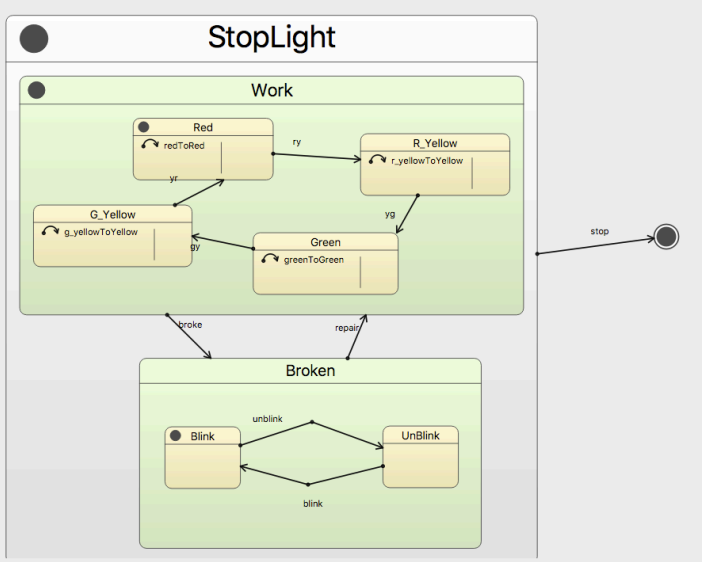
```
Danials-MacBook-Pro:Door danial$ bash run.sh
Executing FSM ...
start closing the door
stop closing the door
start opening the door
stop opening the door
Reach final
Danials-MacBook-Pro:Door danial$
```

1. Door



Comme vu dans le cours, cet exemple illustre l'opération d'une porte automatique. L'implémentation de cet exemple reste simple et ne concerne que l'utilisation des fonctionnalités comme *state*, *transition*, *send* et *final.*

2. Stop Light



Cet exemple illustre les différents étapes de fonctionnement d'un feu rouge. L'implémentation de cet exemple est beaucoup plus compliqué que le précédent. L'application se concentre surtout à l'implémentation de fonctionnalité comme état hiérarchique, gestion de onentry/onexit et gestion de raise/send.

Grâce aux onentry and raise, les transitions d'états dans cet exemple est automatisé. En appelant l'événements intern de l'état à chaque entrée de manière raise, le program peut s'exécuter en boucle. Des compteurs sont mis en place pour éviter que le programme tourne en boucle infini.

Cet application peut être améliorer en ajoutant les fonctionnalités comme la transitions temporisées (permet d'automatiser les transitions entre les états dans Work).