Institut National des Sciences
Appliquées de Lyon
&
KompleXKapharnaüM

Stage de 4ème année du département Génie Électrique
Projet Do Not Clean

Réalisation d'une carte multimédia programmable
et contrôlable via wifi

Implémentation des Machines à nombre Fini d'États

Auteur :
Olivier Radisson

Tuteur de stage :
Gilles Gallet

Chef de projet :
Pierre Hoezelle

- 6 octobre 2014 -

Dernière édition le 8 janvier 2015

Résumé

Ce document présente l'implémentation qui est faite des Machines à nombre fini d'États [1]. Cette implémentation n'est pas rigoureuse d'un point de vue mathématique mais étend son application habituelle aux problématiques rencontrées sur le projet.

Une FSM est toujours dans un état définit auparavant et possède pour chaque état un nombre fini de transitions qui seront franchies dès que leur condition sera validée. Une fois une transition franchie la FSM change d'état et attends de nouveau qu'une transition soit franchie.

Cette implémentation sert principalement à mettre en place les protocoles nécessaires au fonctionnement de notre réseau et ajoute une grand robustesse à ce niveau. De plus les FSM, comme nous les appellerons désormais dans ce document, serviront de base pour l'implémentation de la logique de scénario.

1. De l'anglais *Finite State Machine* traduit communément de façon maladroite par Machina à États Finis

Table des matières

1 Introduction **1**

2 Présentation succincte des FSM **1**

3 Implémentation des FSM dans le projet **2**
3.1 Les signaux 2
3.2 Les états 2
3.3 Les transitions et conditions 3

4 Exemple d'application d'une FSM **3**
4.1 Principe du protocole à implémenter 4
4.2 États 4
4.3 Conditions 4
4.4 Signaux 4
4.5 Implémentation du protocole 5

1 Introduction

Ce document à une visée majoritairement technique. Les points abordés peuvent intéresser les utilisateurs les plus curieux, mais sa vocation première est de présenter l'implémentation faite d'une certaine logique d'automatisme pour documenter le développement du projet.

La partie logique du projet doit gérer de nombreuses chose comme des protocoles de synchronisation du temps ou l'exécution de scénarios définis à l'avance. Ces comportements se doivent d'être le plus sûr possible, c'est à dire résistants aux imprévus comme la connexion simultanée de plusieurs cartes sur le réseau ou l'attente d'un message qui s'est perdu en route.

Pour répondre à cet objectif, le projet se basera sur une implémentation de la logique des Machines à nombre fini d'Étapes [2][1] qui sera adapté aux problématiques rencontrées.

Tout au long de ce document sera utilisé l'acronyme FSM pour *Finite State Machine*, origine anglophone du terme Machine à nombre fini d'États.

2 Présentation succincte des FSM

Une FSM peut être représentée par un graphique comme celui-ci :

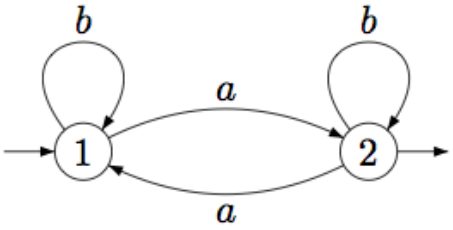

Figure 1 – Exemple de FSM

Ici la machine présentée possède deux états 1 et 2 ainsi que deux signaux a et b. Si la FSM est à l'état 1 et reçois le signal a celle-ci passera dans

2. De l'anglais *Finite State Machine* traduit communément de façon maladroite par Machina à États Finis

l'état 2.

Une machine possède un nombre fini d'états, ici 2 en l'occurrence. Un certain nombre de signaux, ici a et b. Et à chaque état est associé aucun, un ou plusieurs signaux qui peuvent permettre à la machine de changer d'état.

3 Implémentation des FSM dans le projet

La différence principale entre les FSM théorique et celle qui seront utilisées dans le projet provient des signaux qui sont plus complexe que de simple valeur.

De plus il existe des étapes conditionnelles qui ne sont pas des états pour la machine mais qui permet de traiter un signal pour activer, ou non, une transition.

3.1 Les signaux

Tout d'abord il faut définir deux notions qui sont ajoutées ici :
— **JTL** [3] : Le nombre de saut avant la mort du signal
— **TTL** [4] : Le temps de vie d'un signal

Ces notions permettent au signaux de ne pas être ignorés directement si ils ne correspondent pas à une transition. Ils sont placé en mémoire dans une pile et pourront déclencher des transitions tant que aucun de ses paramètres **TTL** et **JTL** ne sont pas expirées.

De plus les signaux peuvent transporter un certain nombre d'arguments tel que les valeurs reçus dans un message OSC, ou le résultat d'une opération précédente.
Enfin chaque message peut se voir attribuer une action lorsque celui-ci est ignoré.

3.2 Les états

Les états sont la base de la logique d'une FSM. Chaque à un identifiant unique, par exemple, *MAIN_WAIT* qui peut être l'état principal d'attente

3. *Jump To Live*
4. *Time To Live*

Signal

ID	JTL	TTL	Arguments	Ignoré
RECV_MSG	1	0.5	(/iamhere, name, *timetag*)	-

Table 1 – Exemple d'un signal pour le réception d'un message OSC

pour un protocole. De plus chaque état à une fonction qui lui est attribuée et qui est lancé lorsque la machine change d'état.

De plus chaque état à un verrou de préemptibilité permettant d'indiquer si l'action de l'étape est bien terminée et si la FSM peut en changer si un signal lui indique.

Enfin chaque état possède un certain nombre de transitions qui lui sont associées.

État

ID	Action	Transition	Préemptible
MAIN_WAIT	*_pass*	RECV_MSG : step_iamhere	True

Table 2 – Exemple d'état : Attente de message d'un protocole

3.3 Les transitions et conditions

Les transitions sont de simples listes de doublet entre un signal et, soit un autre état, soit une condition.

Les conditions sont de simples fonctions qui prennent en paramètre le signal les ayant déclenchées et retournant, soit un état vers le quel transité, soit une valeur nulle pour signifier que le signal ne doit pas déclencher de changement d'état.

4 Exemple d'application d'une FSM

Cet exemple est une partie de la FSM qui gère le protocole de synchronisation dans le réseau. L'exemple concerne la partie synchronisation

du temps côté référence.

4.1 Principe du protocole à implémenter

Depuis l'état principal d'attente, si la FSM reçoit un message OSC */rtp/asktime* celle-ci va démarrer une synchronisation en envoyant une série de **ping** et attendant pour chaqu'un d'eux une réponse **pong** correspondante. Pour chaque échange il va chercher le temps de trajet et tenter, avec un échantillon assez représentatif [5], d'identifier le temps de parcours moyen pour ensuite envoyer son temps plus le correctif à appliquer.

4.2 États

Les états de fonctionnements sont les suivants :

- **MAIN_WAIT** : État d'attente principale du protocole.
- **START_SYNC** : État initialisant les variables nécessaires à la synchronisation et un timer pour éviter de rester bloquer lors de la synchronisation.
- **SEND_PING** : État envoyant un **ping** et notant la date d'envoi dans une variable.
- **WAIT_PONG** : État d'attente après l'envoie d'un **ping**
- **RECV_PONG** : État de réception d'un **pong** et de calcul du temps de parcours puis de l'éventuelle synchronisation
- **SEND_SYNC** : État envoyant un message de synchronisation

4.3 Conditions

Dans ce fonctionnement il y a une transition conditionnelle qui se situe après la réception d'un **pong**. Si les valeurs de temps de parcours sont cohérentes et que la synchronisation est possible cette condition renvoie vers l'état **SEND_SYNC** sinon elle retourne l'état **SEND_PING** pour que cela continue.

4.4 Signaux

Les principaux signaux sont les suivants :

5. La procédure exacte n'est pas traité ici car ce n'est pas le sujet principal

Signaux

ID	JTL	TTL	Arguments	Ignoré
RECV_MSG	1	0.5	(/rtp/asktime, ...)	-
RECV_MSG	1	-	(/rtp/pong, ...)	-
TIME_OUT	3	-	-	-

Table 3 – Signaux présents dans l'exemple

On peut voir ici qu'il y a deux signaux avec le même identifiant. En réalité une petit fonction permet de créer facilement et de manière transparente des conditions lors de la réception d'un message OSC en fonction de l'adresse de celui-ci. Un signal **RECV_MSG** ne déclenchera la passage à **START_SYNC** que si son adresse est */rtp/asktime*.

4.5 Implémentation du protocole

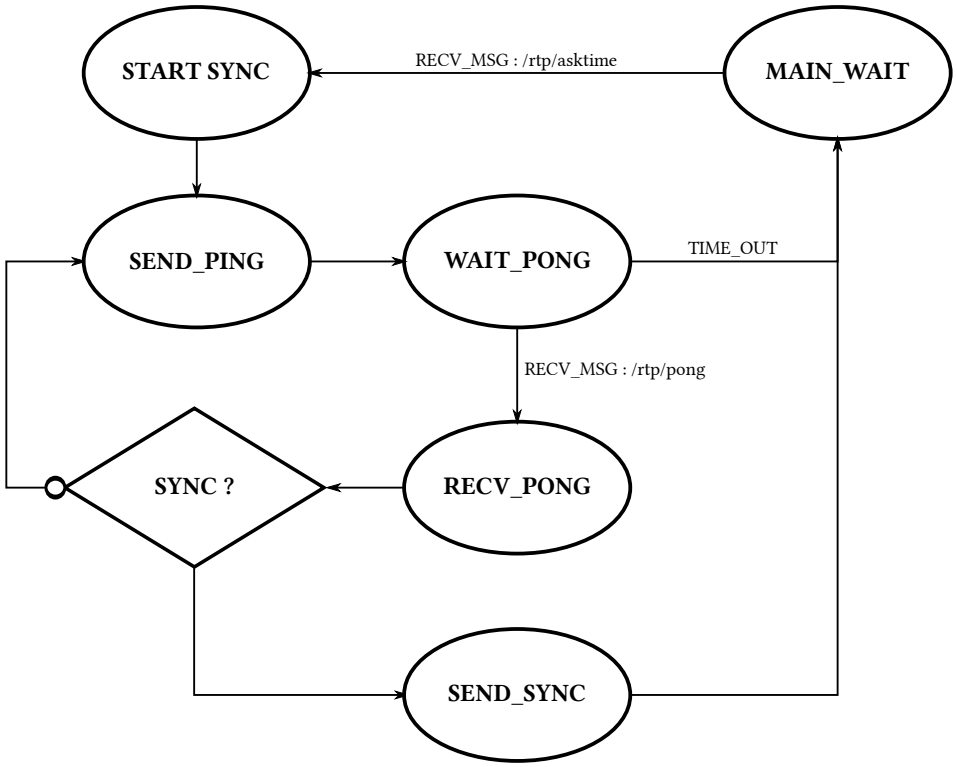

Figure 2 – Implémentation du protocole avec une FSM

Les transitions représentées par une flèche sans signal sont des transitions directes, franchies dès que l'état est préemptible.

Références

[1] *Automate fini*. Déc. 2014. URL : http://fr.wikipedia.org/wiki/Automate_fini (version du 08/01/2015).

[2] (en) Jan Daciuk. *Finite State Automata*. URL : http://galaxy.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/thesis/node12.html (version du 08/01/2015).