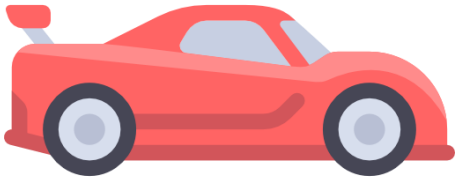
Finite State Machines

Rémy Kaloustian

SI4 – G3



1..Description du domaine possible

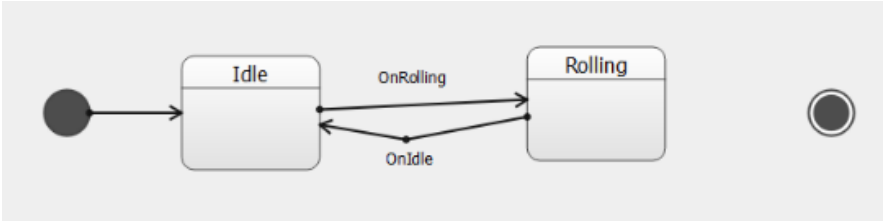
Le domaine d'application serait possiblement l'embarqué. Pour plus de précisions, nous dirons une voiture télécommandée pour enfants.

J'ai choisi ce domaine tout d'abord car il me permettait de voir comment ce que je vois à Polytech pourrait servir dans une application embarquée, mais aussi car il ne présente pas de contraintes au niveau des améliorations que je pourrai apporter à mon générateur de code (états parallèles, délai, etc…).

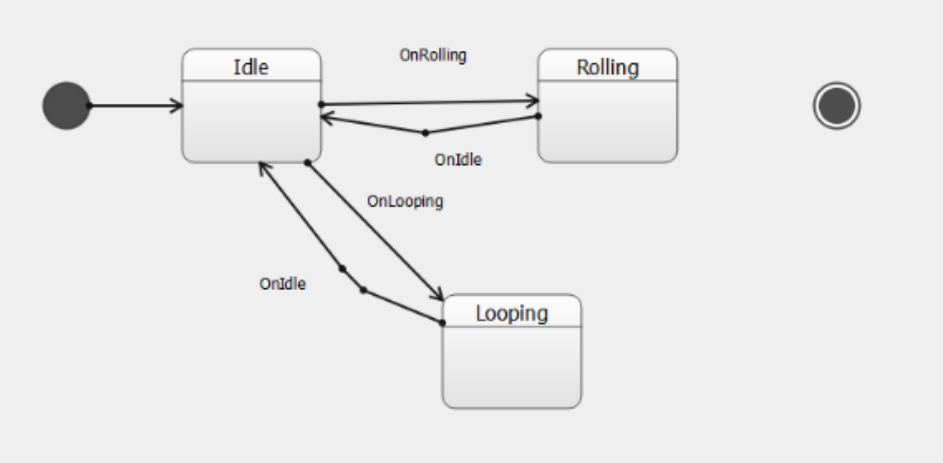
Je me suis basé sur ce cas là pour créer mon générateur de code. De ce fait, la machine à état utilisée correspond bien au domaine visé, car elle a été établie précisément pour ce cas-là.

NB : Dans les schémas, le point final n'est pas relié à la machine, tout simplement car il n'est pas pris en compte lors de la génération du code.

Nous avons tout d'abord une machine à états simple :



Puis une machine à états plus avancée :



NB : Il y a un delay à l'entrée dans looping.

2..Contraintes

L'état initial doit s'appeler Idle.

Les fonctions suivant un changement d'état ne sont exécutées qu'une fois dans l'état (c'est une machine de Moore).

La fonction exécutée dans l'état doit s'appeler InNomDeLEtat.

Le delay est ajouté sur le onentry->send de l'état correspondant.

On suppose que le fichier scxml est bien formé et respecte précisément les contraintes précédentes.

Comment utiliser le code généré ?

Faire un include classique du fichier .h généré.

Commandes à exécuter :

Une fois dans le dossier SCXMLDecoder, exécuter la commande :

. ./ compile_and_run.sh *(il y a bien un point avant ./)*

Un dossier Result est créé, il contient le code généré et les scripts de compilation.

Une fois dans Result, exécuter compile_run_basic.sh pour tester le cas basique, et compile_run_advanced.sh pour tester le cas avancé.

NB : le nom des scripts change en fonction du nom du programme précisé à la construction du CodeGenerator (dans main.cpp)

3..Amélioration = DELAY

Mon but était de proposer une machine à état concrète qui puisse être utilisable dans un vrai contexte, sans avoir à supposer certains paramètres. Cette amélioration devait avoir du sens pour moi. J'ai pensé alors à la possibilité de faire faire un looping à la voiture télécommandée. Toutefois, il se peut qu'on ait besoin d'utiliser des capteurs pour vérifier qu'aucun élément ne va bloquer le looping. Il serait donc pratique d'attendre le temps que les capteurs détectent l'environnement, avant de rentrer dans le comportement de l'état looping. D'où l'utilisation d'un delay avant d'exécuter le « vrai » code de Looping.

4.. Explication des tests

Dans chaque main_ correspondant, on teste tous les états en les activant après un passage dans Idle. Cela permet de voir si les transitions s'effectuent correctement.

5.. Conclusion

Ce projet m'a permis de me rendre compte des bénéfices que peuvent apporter les machines à états finis dans les projets informatiques.

- ➔ Tout d'abord, elles permettent une meilleure représentation du problème et/ou de l'application. Cela engendre une phase de conception plus simple et donne un meilleur contexte avant de se lancer dans l'écriture du code.

- ➔ Ensuite, une fois la machine à états finis implémentée dans l'application, son code est plus facilement compréhensible car il colle plus à la situation réelle (exemple du portail vu en cours) et permet plus facilement de penser au-delà des lignes de code.

- ➔ De plus, les machines à états finis permettent de traiter tous types de problèmes, aussi bien au niveau micro que macro.

- ➔ Enfin, on pourrait croire que cette manière de concevoir les applications soit limitée, mais bien au contraire, par exemple, Qt couplé à l'utilisation du scxml permet de créer des machines à états finis complexes avec des états hiérarchiques, parallèles, des délais, etc… Cette capacité à produire des machines complexes permet d'étendre leur utilisation, par exemple dans des jeux vidéo où la gestion d'une intelligence artificielle peut s'avérer complexe.