Julien KAPLAN

Léo MARACHE

Projet FSM

Implémentations du MVP :

Nous avons dans notre projet :

- Les boissons Tea/Coffee/Expresso ainsi que l'option soupe
- On ne peut pas payer par nfc si aucune boisson n'est sélectionnée
- La réduction au 11^{ième} paiement est valide, si le prix de la boisson payée est au-dessus du prix moyen des 10 derniers achats, la réduction s'appliquera au prochain paiement etc..
- Les options nuage de lait/glace vanille
- Après 45secondes sans action de la part de l'utilisateur tous les interfaces reviennent à l'état par défaut
- La gestion des sliders
- Les options sont non clicables tant qu'une boisson n'est pas sélectionnée
- L'option détection des gobelets
- L'option gestion de l'avancement de la préparation

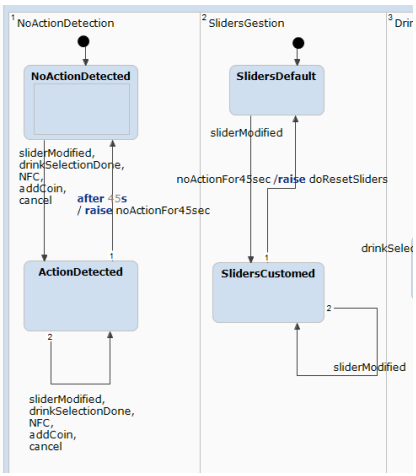
Structure de l'application :

Nos choix ont été guidés par 3 décisions :

- Nous souhaitions depuis le depuis que le code et la FSM soient étroitement liées. C'est-à-dire que l'information de l'étape en cours d'exécution soit visible à un instant t à la fois dans la FSM et dans le code.
- Nous voulions aussi qu'un maximum d'état de la machine à café soient vérifiables lors du V&V pour pouvoir retracer toutes les actions de celle-ci.
- Et enfin nous tenions à ce que rajouter des tâches ou des étapes de préparation soit possible sans trop de difficulté.

La partie création de la commande n'est pas très intéressante à développer ici car son implémentation nous est venue assez instinctivement, nous n'avons jamais retouché la structure de cette partie depuis sa première implémentation.

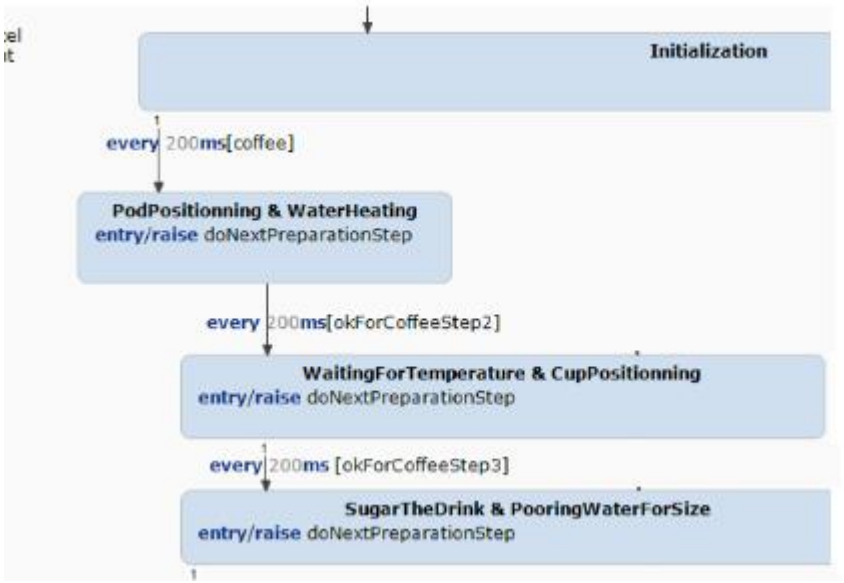
Le seul choix intéressant a été de mettre des états pour visualiser dans la FSM si un slider a été touché ou non (voir image ci-dessous). On aurait juste pu déclencher la méthode doResetSliders après les 45secondes sans action et ainsi avoir une région en moins, mais cela n'aurait pas permis de savoir lors du V&V si on a touché aux silders.



Pour ce qui est de la préparation nous avons eu à changer plusieurs fois de construction.

1ière version :

Au tout début nous souhaitions que les tâches exécutées en parallèles soient traitées dans le code et que la FSM servent à passer d'une étape à l'autre (voir image ci-dessous). L'avantage est que la FSM est très lisible, et qu'on peut tout de même vérifier à quelle étape on est lors de l'étape de V&V du projet.



Cette implémentation pose néanmoins des problèmes critiques :

- Manque de précision sur quelles étapes sont réalisés, si on bloque dans WatingForTemperature & CupPositionning on ne saura pas laquelle de ces 2 tâches a coincé.

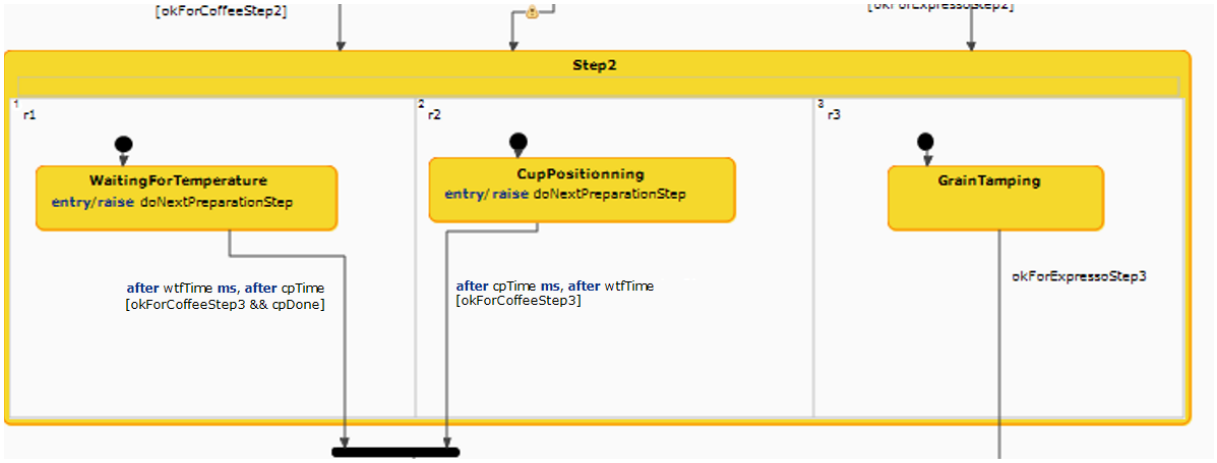
- On ne profite pas de la propriété de parallélisme des FSM ce qui est dommage lorsu'on veut travailler avec cette technologie.

La solution ?

Mettre tous les états possibles en parallèles par étape.

2ième version :

Dans cette version nous appliquons la solution de la version précédente (voir image ci-dessous). Puisque les temps des tâches peuvent varier, nous avons décidé de donner des variables de temps pour chaque tâche qui seraient établies dans le code, à chaque étape de la préparation la FSM demande au code d'établir les temps de l'étape suivante via la méthode doNextPreparationStep (nous conservons donc ce lien étroit entre code et FSM).



Une fois de plus nous nous rendons compte que cette structure comporte des problèmes :

- Alors qu'on a demandé un café on passe par l'état Grain Tamping ce qui n'est pas très logique, même si rien ne sortira de cet état lorsqu'on demande un café ce n'est pas une bonne pratique et lors de la V&V on se saura pas si on a exécuté ou non cette tâche.

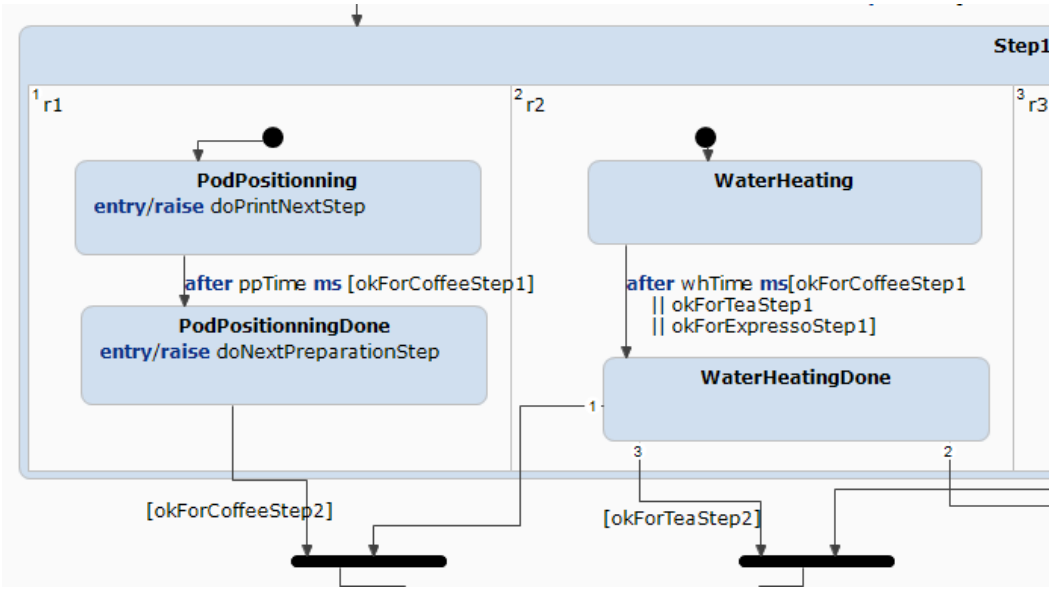
- De plus avec une telle implémentation nous devons gérer le temps des tâches à la fois dans le code et dans la FSM ce qui n'a pas d'intérêt, autant tout traiter dans l'un ou dans l'autre. Notre application devient assez compliquée à étendre, car on doit modifier beaucoup de code juste pour rajouter 1 tâche.

La solution ?

Ajouter 1 état transitoire pour chaque région

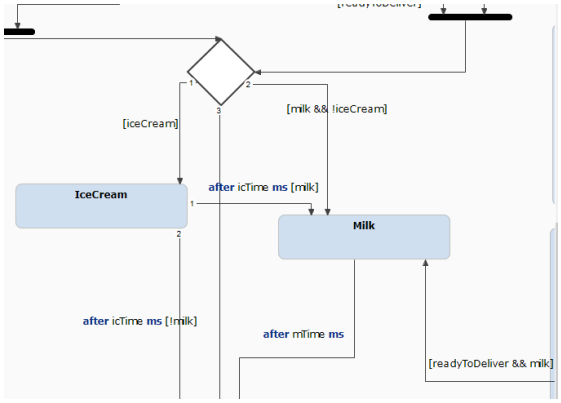
3^{ième} version (version choisie) :

Nous décidons donc d'ajouter des états transitoires à notre version 2 (voir image ci-dessous). On entre toujours dans toutes les tâches d'une étape mais désormais on n'exécute que celles de la boisson à réaliser, vérifier cela est simple puisqu'il suffit de voir dans quel état « Done » on est passé. Une fois que toutes les tâches sont effectuées et que le code java a mis à jour les temps pour l'étape suivante, on peut changer d'étape de la préparation.



On a besoin de mettre [okForDrink] entre les 2 états d'une même région afin d'être sûr de ne pas lancer un mauvais état, on sait donc en regardant les booléens quelles boissons vont exécuter la tâche associée. Une autre solution aurait été de mettre les temps à 999 pour les états non utilisés, mais il aurait fallu remettre la valeur des temps à jour après chaque utilisation et ce n'est pas très intuitif non plus.

Cette version 3 possède, comme toutes les solutions, des défauts :

- Sur l'implémentation des options, comme celles-ci ne sont pas obligatoire, il était trop compliqué de les inclure avant un « join » nous les avons donc traitées soit à part, lorsque l'option ne s'exécute pas en parallèle à d'autres tâches, ce qui est le cas du nuage de lait et de la glasse à la vanille (voir l'image ci-dessous). Théoriquement ces tâches devraient être exécutées après le coulage de l'eau mais puisque couler l'eau est la tâche la plus longue, nous considérons qu'elles s'exécutent après l'étape entière. Par ailleurs, les temps de ces options est définit dans la FSM qu'ils se varient pas et qu'on ne les traite pas comme les autres tâches dans des « Step ».
- 

ici les options sont traitées après la dernière étape de préparation de la boisson choisie

  Soit en mettant le temps à 0 lorsque l'option doit être exécuter en parallèle à d'autres tâches comme c'est le cas pour les croutons ou l'ajout du sucre. Donc lors de la préparation d'une boisson pouvant être sucré, on passera dans l'état « SugarTheDrinkDone » même si l'on n'a pas ajouté de sucre ce qui n'est pas optimal pour la partie du projet V&V.

- Lorsque l'on doit ajouter une étape de préparation « Step », dans la FSM il faut que l'un des états d'initialisation contienne la méthode doPrintNextStep qui va demander au code d'afficher l'étape actuelle. Ce défaut complexifie légèrement l'aspect extensible de notre implémentation.

- Lorsque l'on ajoute la préparation d'une boisson, il faut que dans chaque étape il y ait une et une seule de ses tâches qui contienne la méthode DoNextPreparationStep pour dire au code de préparer l'étape suivant. Ce défaut complexifie légèrement l'aspect extensible de notre implémentation.

- Notre classe java principale DrinkFactoryMachine est très lourd, nous ne nous sommes pas concentrés sur la déresponsabilisation de celle-ci ce qui ne facilite pas forcément la compréhension du code.

Les principaux avantages de notre structure sont les suivant :

- Si la machine bug ou s'arrête, on saura à quel état elle s'est arrêtée à la fois dans le code et dans la FSM

- La FSM est assez explicite visuellement, et rajouter une étape ou une tâche n'est pas particulièrement compliqué. Le code aussi est facilement extensible.

- On peut poser n'importe quelle question en V&V à la FSM et on obtiendra une réponse cohérente (à part pour les options en parallèles comme l'ajout du sucre, dont on a parlé précédemment)

- Le principe de parallélisme de la FSM a été très utilisé pour gérer les tâches en parallèles, nous avons toute la gestion du temps dans la FSM mise à part pour la barre de progression qui elle est mise à jour grâce à un compteur de milliseconde dans le code. Le code donne les temps à attendre mais c'est à la FSM de les appliquer.

V&V :

La prop1 permet de vérifier que si on a choisi une boisson, on aura toujours un moyen de payer c'est une property de Liveness et elle permet une Verification

La prop2 permet de vérifier que si on a payé, on aura toujours un moyen de choisir une boisson c'est une property de Liveness et elle permet une Verification

Une property de Safety qui permettrait une Validation serait la suivante :

Si on ne fait ni un addCoin ni un nfc on ne peut pas obtenir de paymentChecked

Rétrospective :

Nous aurions dû consacrer plus de temps à la partie V&V et surtout à la compréhension du logiciel LTSA car finalement nous testons une partie assez fiable de la FSM et de manière très succincte.

Nous trouvons que nous avons bien fait évoluer notre projet petit à petit ce qui nous a permis de ne pas s'engouffrer dans un tunnel. Lorsqu'on a remarqué que notre architecture n'était pas viable, nous avons pu revenir en arrière assez facilement.

Nous aurions peut-être dû réfléchir plus en orienté objet pour la partie code, car à force de rajouter des méthodes dans la classe principale, ça devenait compliqué vers la fin du projet de retrouver des fonctionnalités à modifier.

Nous nous sommes bien répartis les tâches de travail à 2, chacun à une vue sur l'ensemble du projet ce qui a permis d'éviter des erreurs d'inattention notamment.