Rapport Projet Domotique

Finite State Machine

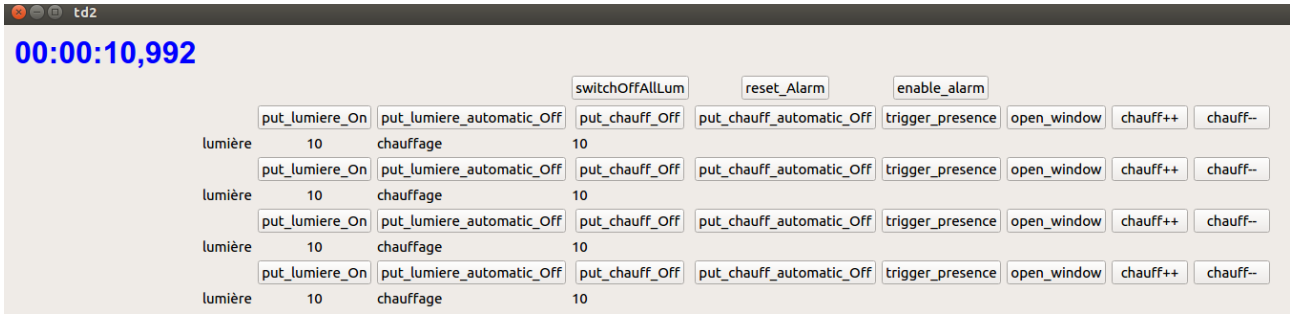
Thibaut GONNIN

Travail Réalisé :

Le contenu de ce projet contient le MVP (sécurité, lumière et chauffage) ainsi qu'un système de gestion et de détection des fenêtres ouvertes.

Explication sur l'interface d'utilisation :

Voici comment se présente mon interface.



Chaque ligne correspond à une pièce. Chaque pièce possède :
- Un indicateur sur la température de la pièce
- Un indicateur sur la luminosité dans la pièce
- Un switch pour le mode automatique de la lumière
- Un switch pour allumer la lumière ou l'éteindre
- Un switch pour le mode automatique du chauffage
- Un switch pour allumer le chauffage ou l'éteindre
- Un bouton pour augmenter la consigne du chauffage
- Un bouton pour diminuer la consigne du chauffage
- Un switch pour ouvrir ou fermer la fenêtre.
- Un bouton pour simuler une intrusion dans la pièce.

A cela s'ajoute :
- Un timer
- Un bouton pour éteindre toutes les lumières de toutes les pièces
- Un bouton pour arrêter l'alarme suite à une intrusion
- Un switch pour activer la sécurité ou non.

Commentaire Global :

J'ai décidé de créer plusieurs machines à états communiquants ensemble plutôt qu'une grosse machine. La raison principale à ce choix est majoritairement vis à vis de la modularité apporté à ma solution finale. En effet, au besoin, je peux rajouter un système d'éclairage dans l'une des chambres juste en instanciant une nouvelle machine et en la reliant au reste du système. L'intérêt de cette modularité ne vient malheureusement pas sans défaut. Les liens à créer entre les machines se font directement via le code cpp et non pas de la state machine. Dans le cas où on crée une grosse State Machine regroupant la totalité des fonctionnalités, on peut très distinctement voir les différentes relations entre les composants du système.

Pour éviter d'avoir à attendre trop longtemps et avant tout pour se rendre compte rapidement de comment fonctionne, le système j'ai fait en sorte que la mis à jour de l'affichage se fasse toutes les secondes et j'ai défini une journée comme étant 1 minute.
J'ai ainsi découpé ma minute en 4 segments égaux pour tester mon système de gestion automatique du chauffage et de la lumière.

N.B. : Si vous voulez changer les valeurs des segments dans le domo.h, faites attention à ce que les valeurs de temps en début et fin du tableau soient les mêmes pour tous les tableaux. Ayant considéré une minute = 1 journée j'ai mis une condition pour qu'à chaque journée mes compteurs retournent à 0. Dans le cas où les tableaux seraient mal remplis les gestions automatiques ne seront plus fiables.

Attention à bien d'abord passer en manuel pour ensuite utiliser les boutons spécifiques à ce mode (e.g. putLumiereOn …) .

I – Lumière

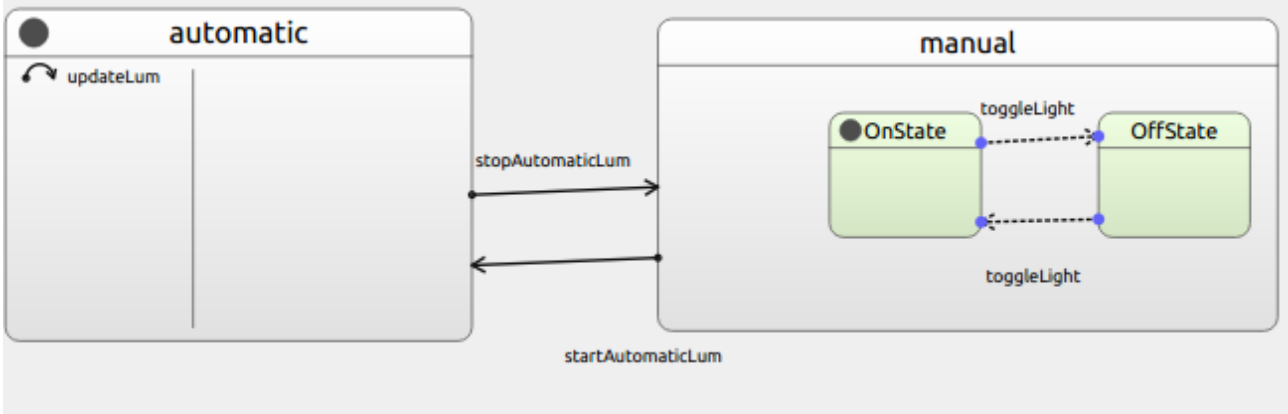


fig.1 : FSM lumManager

Pour la lumière, il fallait différencier le mode automatique du mode manuel.
Dans le mode Automatique, on a besoin de mettre à jour le modèle assez souvent pour pouvoir changer la valeur de la luminosité de la lampe. J'ai donc mis en place un Qtimer qui va envoyer un événement « updateLum » toutes les secondes à mon « LumManager » pour pouvoir dans un premier lieu mettre à jour mon modèle (la valeur de consigne change en fonction du temps dans le mode automatique) puis faire augmenter la valeur de la lampe dans un deuxième temps.

Pour augmenter la valeur de la lampe, on va tout d'abord vérifier si la valeur actuelle est inférieure ou supérieure à la valeur de consigne. Si ce n'est pas le cas, c'est que les 2 valeurs sont égales et qu'il n'y a plus rien à faire. Dans le cas inverse, on augmente ou décrémente d 'une unité la valeur associée à la lampe pour ensuite réinjecter dans la StateMachine un événement « updateLum » qui va permettre d'appeler récursivement la fonction jusqu'à ce que la valeur de la lampe et sa consigne soient strictement égale.

Pour la partie Manuel, il reste assez simple niveau fonctionnement. Il n'existe plus de transition prenant en compte « updateLum » et, de ce fait, il n'y a plus de mis à jour de modèle. Par contre, à l'arrivée dans le manuel la valeur maximale de lumière est donné en tant que consigne à l'ampoule et lorsque la transition vers l'état « offState » est engagé alors c'est la valeur minimale qui lui est donné.
De ce fait, on recrée le comportement simple d'une lampe basique.

Pour ce qui est de la fonctionnalité du bouton pour éteindre tout l'appartement en un clic, il suffit d'envoyer à toutes les lumManager l'événement pour passer en manuel puis pour éteindre les lampes.

II - Chauffage

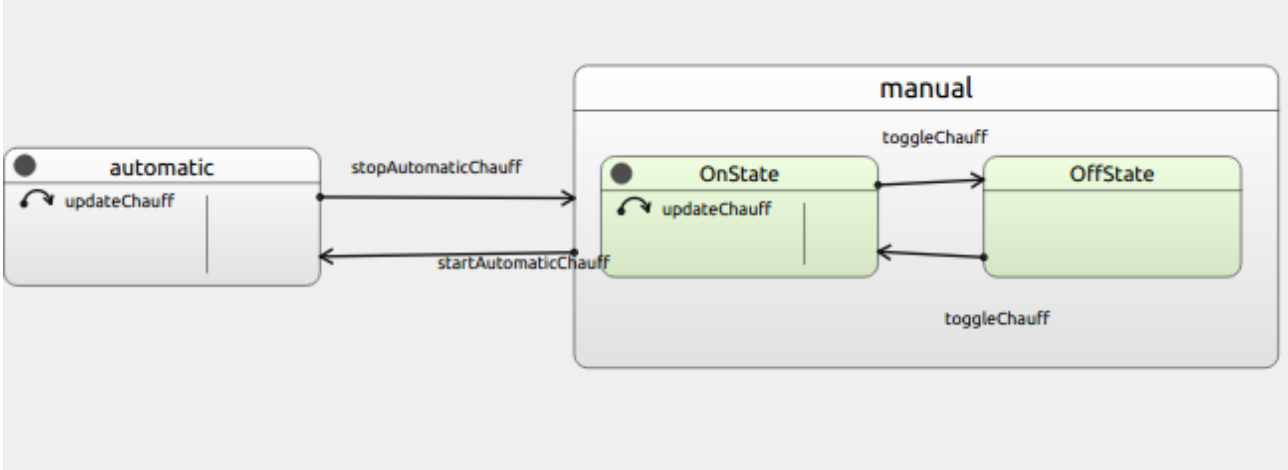


fig.2: FSM chauffManager

Le fonctionnement du chauffage ressemble beaucoup à celui précédemment décrit pour la lumière. La différence majeure vient au niveau du « OnState » où la valeur de la consigne peut changer. Dans le cas de la lumière on passait directement, la consigne de la lumière à sa valeur maximale pour pouvoir allumer la lampe à la luminosité maximale. Dans le cas du chauffage, on va juste rajouter une nouvelle transition sur l'état « OnState » pour permettre de changer la valeur de la température. À notre bon vouloir.
L'utilisateur va donc dans cet état pouvoir changer la température de consigne grâce aux boutons « chauff++ » et « chauf-- ».

III – Sécurité

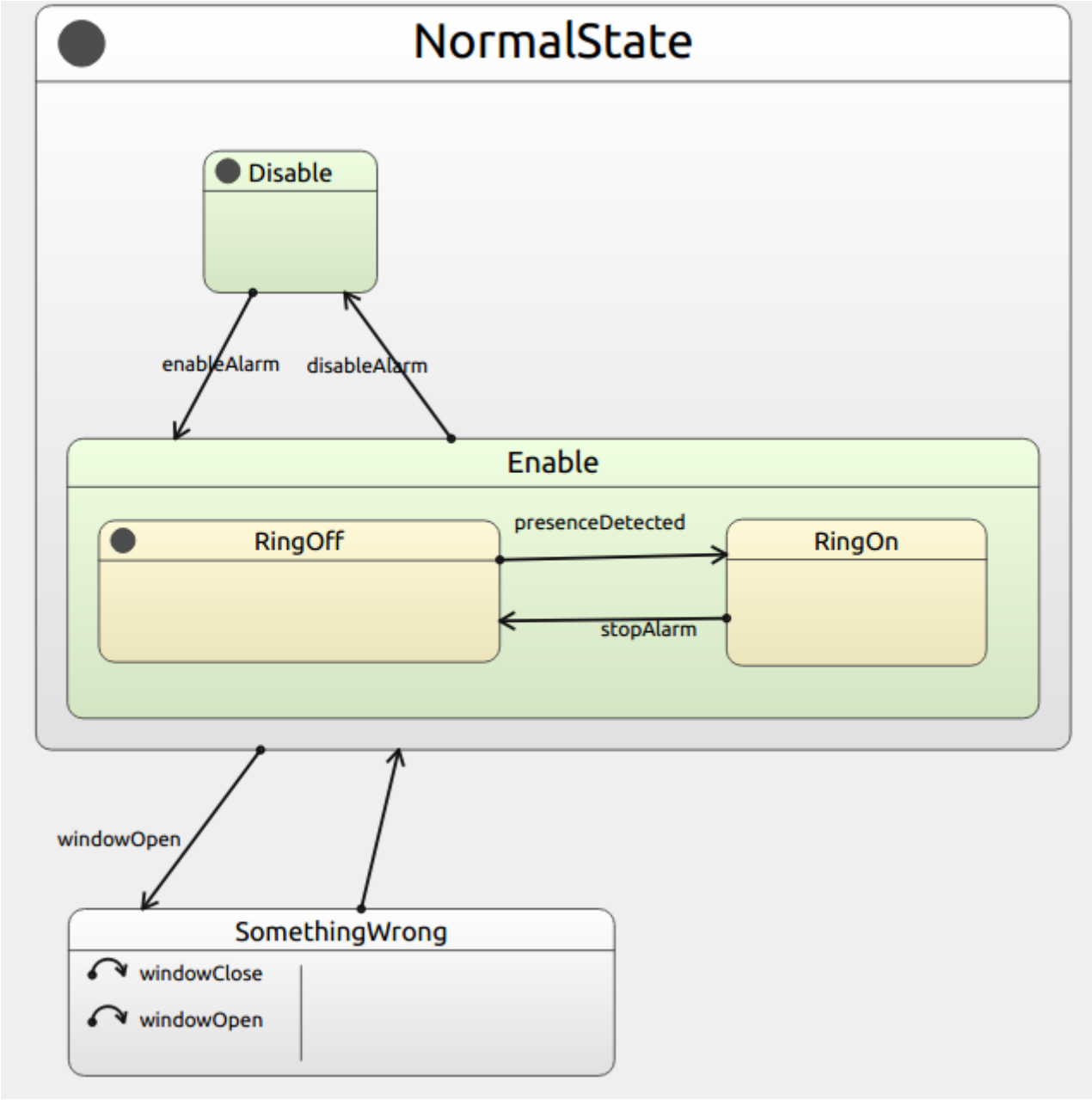


fig.3 : FSM sécu

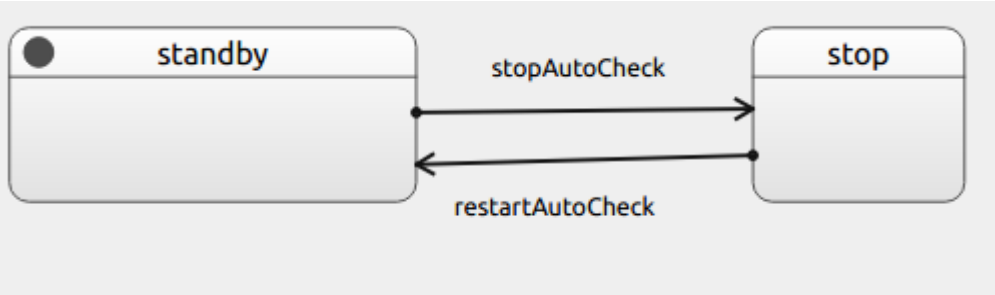


fig.4 : FSM alarm

Pour le système d'alarme, j'ai décidé de représenter la centrale de commande de sécurité par une State Machine « sécu » unique à l'appartement et de modéliser les détecteurs de mouvements grâce à une State Machine « alarm » par pièce.
Le but de sécu est de gérer à lui seul la totalité de la sécurité de l'appartement. C'est d'ailleurs pour cette raison que l'on peut voir apparaître certaines transitions en rapport avec la fenêtre (voir chapitre suivant).
Une « alarm » peut être dans seulement 2 états différents. L'un où le capteur attend de capter un mouvement et l'autre où le système attend la confirmation de l'instante « sécu » pour pouvoir recommencer son attente. A noter que le capteur ne va pas envoyer d'événements intéressants à sécu lorsque ce dernier est dans l'état « Disable ».
L'instance sécurité commence dans le stade inactif pour ensuite prendre son sens dans le stade actif. Dans le cas où on se trouverait dans l'état « RingOff » le seul moyen de passer dans l'état « RingOn » est d'attendre l'envoi du bon événement par l'une des alarmes. Une fois qu'une alarm notifie la sécu d'une intrusion, ce dernier se charge de passer tous les détecteurs de chaque pièce à l'état « stop » pour qu'ils arrêtent de rechercher une intrusion potentielle.

Je me suis permis de simplifier le système décrit dans le sujet en modélisant juste un simple bouton pour remettre à l'état initial les capteurs et « sécu ».

IV – Fenêtre

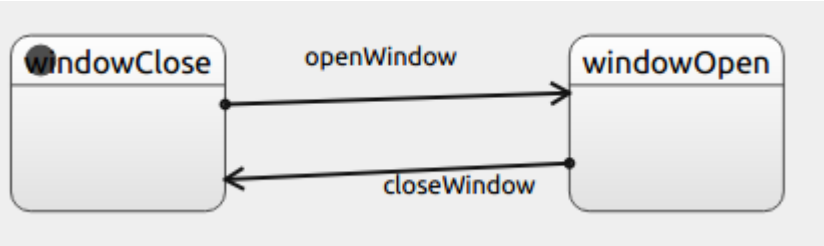


fig.5: FSM fenetre

Avec le rajout de la gestion de la fenêtre j'ai rajouté 3 interactions principales avec ce qui était déjà présent dans le système.

-Arrêter le chauffage quand on ouvre la fenêtre.
-Détecter une intrusion dans le domicile si une fenêtre s'ouvre alors que la sécurité est activé.
-Empêcher l'allumage de la sécurité dans le cas où la fenêtre est ouverte.

Pour l'arrêt du chauffage, je me suis contenté d'envoyer au chauffageManager les événements permettant d'arrêter le chauffage dès qu'une ouverture de fenêtre est enclenché.
Pour la détection d'intrus, je me suis encore une fois contenté d'envoyer à l' « alarm » une intrusion. A noter que la différence avec une intrusion classique vient du fait qu'il n'est pas possible d'arrêter l'alarme tant que toutes les fenêtres ne sont pas fermés.
Pour empêcher la mis en route de la sécurité quand une fenêtre est ouverte j'ai rajouté une nouvelle fonctionnalité à la State Machine « sécu ». Cette dernière utilise un datamodel pour pouvoir être utilisé potentiellement dans le futur. Si vous reprenez la figure 3, vous pouvez vous rendre compte qu'un état SomethingWrong apparaît en dehors du NormalState. Pour y rentrer vous devez avoir un événement « windowOpen » qui arrive. La State Machine à partir de ce moment compte le nombre de windowOpen que le système reçoit. L'intérêt à ceci réside dans le fait que l'on pourrait oublier de fermer certaines fenêtres avant d'allumer la sécurité.
Par exemple si vous ouvrez la fenêtre de la chambre 1 puis celle de la chambre 2 puis que vous refermez celle de la chambre 1. il reste toujours la chambre 2 qui n'est pas fermé. L'intérêt du rajout d'un datamodel vient essentiellement de ce cas de figure. Cette variable interne permet véritablement de compter le nombre de fenêtres ouvertes globalement dans l'appartement. C'est lorsque cette variable est à 0 que nous sortons du « SomethingWrong » pour rentrer dans le normalState.
L'intérêt de cette technique est qu'elle est très adapté à ce que l'on souhaite. Savoir le nombre de fenêtres ouvertes est typiquement intéressant pour le contrôleur et en tant qu'utilisateur, on n'a pas trop besoin d'avoir cette information.
Qui plus est on peut très bien imaginer que l'on puisse rajouter le même système avec les portes et dans ce cas l'adaptation se fait très rapidement puisqu'il suffit globalement de rajouter une State Machine et une variable pour compter le nombre de porte ouverte.

Le désavantage pourrait potentiellement venir plus tard si l'on décide de faire du Model Checking sur notre State Machine. En effet, rajouter des variables internes peut rajouter de la complexité pour ce genre de pratique, mais j'ai considéré que cela restait quand même peu complexe vu le peu de pièce que l'on doit gérer.