Le  noeud commande est à la fois utilisé comme une interface entre les noeuds aiguillages, postes, robots et tâches et le noeud commande\_locale lui même servant d’interface entre la commande et la simulation. Mais le noeud commande sert également de superviseur. C’est dans ce noeud que nous avons codé le réseau de Pétri répondant au cahier des charges.

Interfaces entre les noeuds aiguillages et postes et la simulation

Lorsqu’un noeud aiguillage ou poste souhaite modifier l’état d’un des actionneurs comme les aiguillages, les stops et les ergots, il publie dans le topic correspondant à l’action le numéro de l’actionneur. La commande met alors à jour le tableau correspondant puis le publie dans un topic à destination de commande\_locale qui est retransmis à V-Rep. Par exemple, supposons que le noeud associé à l’aiguillage 11 souhaite le faire tourner à gauche, il publiera tout d'abord 11 dans le topic /commande/DeverouilleAiguillage, le noeud commande mettra alors à jour les tableau Dx et Vx : Dx recevra 1 pour l’aiguillage 11 et Vx recevra 0. Ces variables ne conduisent pas à une action de V-Rep et le topic utilisé pour les envoyer est le même que celui utilisé pour envoyer un ordre de rotation. Par conséquent, il n’est pas nécessaire de les envoyer tout de suite à commande\_locale. Le noeud aiguillage 11 va ensuite écrire 11 sur le topic /commande/AiguillageGauche. La commande met à jour le tableau RxG puis publie sur le topic commande/Simulation/Actionneurs\_aiguillages les tableaux RxG, RxD et LOCK(qui dépend de Vx et Dx), qui seront transmis à V-Rep par le noeud commande\_locale. Il remettra alors la valeur de RxG associée à cet aiguillage à zéro pour éviter que vrep reçoive de nouveau l’ordre de faire tourner l’aiguillage à gauche lors d’une autre publication pour un autre aiguillage par exemple. Lorsque le noeud aiguillage voudra verrouiller l’aiguillage, il transmettra 11 à commande pour mettre à jour les tableaux Dx et Vx.

Ces fonctions sont gérés par une instance de la classe Commande;

Les noeuds aiguillages et postes n’utilisent pas le noeud commande pour avoir l’état des capteurs dans V-Rep, il souscrit directement aux topics de V-Reps.

Superviseur

Cahier des charges :

Le cahier des charges général qui doit être respecté par le système est le suivant :

Des navettes arrivent entre les aiguillages 9 et 10 transportant un certain produit. Le produit transporté doit suivre une certaine gamme : il doit passer par différents postes de travail dans un ordre prédéfini. Par exemple, le produit A devra passer par le poste 1, puis les postes 2, 3 et enfin le poste 4. Une fois la gamme terminée, la navette transportant le produit fini devra aller dans la section entre les aiguillages 3 et 4 où elle sera supprimée.

Lorsque le produit transporté par la navette doit aller au poste de travail i (tâche i), la navette doit aller au poste i. Une fois arrivé au poste, le robot doit prendre le produit, et le déposer au niveau du poste de travail. Une fois le produit pris par le robot, la navette est considérée comme vide et repart avec la destination 0. Cette étape est déjà gérée par les noeuds postes. Une fois que le poste de travail i a terminé son traitement sur le produit, une navette vide doit être ramenée au poste de travail pour récupérer le produit traité et doit l’emmener au poste de travail suivant.

Pour réaliser le réseau de Petri, nous avons considéré quelques hypothèses supplémentaires :

    -    Les navettes qui arrivent ne sont jamais vides.

    -    Une navette vide ne doit jamais sortir.

    -    Bien que physiquement possible, on suppose qu’on ne peut pas prendre un produit au niveau du poste 1 pour le déposer au niveau du poste de travail 2 et inversement. Idem pour les postes 3 et 4.

Fonctions utilisées par le réseau de Pétri

Le noeud commande va également permettre de commander les robots, le redémarrage des navettes au niveau des postes et la destination des navettes. Pour cela, le programme main\_commande dans lequel le réseau de Petri sera implémenté contient des instances des classes Commande (vu précédemment) et Robots. La classe Commande va permettre au niveau de la supervision de créer un certain nombre de fonctions qui utiliseront des services et contiendra des attributs qui seront mis à jour par des fonctions callbacks qui permettront de connaître l’état du système, notamment la présence de navettes vides ou occupées au niveau des postes (et leurs handles).

La classe Robots a le même objectif mais uniquement en ce qui concerne les robots. Elle possède par exemple une fonction ControlerRobot() qui prend en paramètre le numéro du robot concerné, la position souhaitée (dans le plan), en position haute ou basse et pince ouverte ou fermée. Elle contient également des retours sur les mouvements des robots : la fonction RobotEnPosition() renvoie 1 si le robot a atteint la position demandé, la fonction BrasEnPosition() renvoie 1 si le bras est en position haute, -1 sinon, et la fonction  PinceEnPosition() renvoie -1 si la pince est ouverte, 1 sinon.

Réseau de Petri :

Le réseau de Petri que nous avons réalisé se compose de quatre parties principales associées à chaque poste du système. Il s’agit d’un réseau de Petri coloré dont le marquage de certaines places correspondra à un handle de navette. Il contient un certain nombre de places communes aux différentes parties : la place 0 correspond aux navettes dont la destination est d0 (tourne au milieu du circuit). La place 50 (resp. 250) correspond à la disponibilité du robot 1 (resp. robot 2) (partage de ressource). Ces places contiennent un jeton au marquage initial. Toutes les autres places correspondent à une phase du traitement du produit.

Le réseau de Petri précédemment présenté répond bien au cahier des charges mais peut être amélioré. Dans cette solution, une tâche n’est pas représentée dans le réseau de Pétri comme une ressource partagée. En effet, imaginons qu’une pièce doive aller au poste de travail 2 alors que celui-ci est déjà occupé. Lorsque la navette transportant cette pièce arrivera au niveau du poste, celui-ci le fera repartir automatiquement. Le réseau de Petri n’a donc pas besoin de le considérer comme une ressource partagée. Cependant, on pourrait quand même le prendre en compte pour éviter que la navette aille à chaque fois au niveau du poste alors que celui-ci ne sera pas disponible. Pour ce faire, il faut utiliser une place par tâche pour modéliser la ressource partagé.

Une deuxième amélioration consiste à faire passer directement un produit entre deux postes de travail voisins sans avoir besoin d’utiliser une navette pour passer d’un poste à l’autre. Cette amélioration dépend du cahier des charges : il devra être spécifié si ce mouvement est autorisé ou non. Pour mettre en place cette amélioration, il serait préférable que l’amélioration précédente le soit également puisque les tâches devront être considérées comme une ressource partagée. En effet, puisque le produit ne passera pas par le poste, il n’y aura pas de contrôle pour savoir si le poste de travail est occupé ou pas. La mise en place de cette amélioration est cependant beaucoup plus compliquée parce qu’il faut modifier le noeud tâche. En effet, pour le moment, lorsqu’une tâche est terminée, le produit traité est envoyé au poste correspondant. La possibilité de l’envoyer à une autre tâche n’a pas été mise en place.