Gestion des Navettes :

Afin de gérer au mieux les navettes, nous avons choisi de créer un nœud “shuttles” indépendant. L’idée de ce nœud est venue du fait que la solution que nous avons conçue présente de nombreux nœuds tels qu’aiguillages ou postes qui doivent interagir avec les navettes. La centralisation des informations sur les navettes au niveau d’un nœud unique nous a permis d’en faciliter la gestion.

L’idée est que ce nœud soit informé par les autres nœuds de la simulation des changements opérés sur les navettes. Il va posséder en interne une collection d’objets « Shuttle » présentant de nombreux attributs présentés ci-dessous :

class Shuttle

{

public:

// un nom unique est attribué à chaque navette

    std::string name;

// L’attribut destination permet de connaitre la destination de la navette

    int destination;

// L’attribut product permet de connaitre le produit transporter par la navette et l’étape de la gamme dans lequel il se trouve.

int product;

// Les attributs handle et handlePlatform sont des identifiants uniques attribuer par Vrep au navette, ils nous permettent d’identifier les navettes et également de changer la couleur de la platform lorsque le produit transporter évolue.

    int handle;

    int handlePlatform;

// L’attribut zone pourrait permettre de connaitre en permanence la position de la navette mais cette fonctionnalité reste à implanter.

    int zone;

    […]

};

Le nœud connaît donc en permanence l’état des navettes dans le circuit. Grâce à cela il va offrir des services permettant la récupération des informations sur les navettes ou encore la recherche des navettes ne transportant pas de produit. Pour l’informer des changements les autres nœuds n’ont cas publier des messages sur des topics dédiés. Des topics sont aussi disponible pour la création et la destruction des navettes.

    La classe majeure du package *shuttles* associé à notre noeud est le *ShuttleManager*, une explication de cette classe est disponible ci-dessous. L’ensemble des fonctionnalités est enfaite réalisé par le code contenu dans cette classe, le *mainShuttles* se contente de crée le nœud et l’objet puis tourne à l’infini sur une boucle d’écoute ROS classique.

/\*\*\*\* Projet Long N7 2017 \*\*\*\*/

#ifndef SHUTTLEMANAGER

#define SHUTTLEMANAGER

#include <ros/ros.h>

#include "shuttles/srvGetShuttleStatus.h"

#include "shuttles/srvGetEmptyShuttles.h"

#include "shuttle.h"

#include "string.h"

#include <std\_msgs/Int32.h>

#include "shuttles/msgShuttleChange.h"

#include "shuttles/msgShuttleCreate.h"

#include <cstdlib>

#include <map>

#include <vector>

class ShuttleManager

{

private:

    std::map<int,Shuttle\*> ShuttlesMap;

    /\*

    Collection "MAP" <key,Object\*>, key = handle, Object\* = pointer to shuttle object

    Cette Collection permet le stockage des navettes et de les identifier à l'aide de leur handle qui est un entier unique pour chacune d'entre elle.

    Vous remarquerez que ceux qui est vraiment stocker dans la collection est le pointer vers l'objet pour des question de simplicité.

    \*/

    ros::ServiceServer serviceGetShuttleStatus;

    /\*

    Le service "shuttles/srvGetShuttleStatus.h" disponible dans le package shuttles permet de récupérer l'ensemble des informations disponibles sur un shuttle.

    Pour l'utiliser il suffit de lui transmettre l'handle de la navette.

    il retourne true si la navette a bien été renseigner, false si le handle n'existe pas

    \*/

    ros::ServiceServer serviceGetEmptyShuttles;

    /\*

    Ce service permet de récuperer la liste des navettes vides. Il retourne 2 vector handles et zones. Il ne nécessite pas d'argument en entrée. Il retourne false si aucune navette n'est vide.

    \*/

    ros::Subscriber subProductChange, subDestinationChange, subZoneChange, subAddShuttle, subDelShuttle;

    /\*

    Les subscribers permettent la gestion de la collection map fonction des msg recu par les autres noeuds

    subAddShuttle : un msg de type "shuttles/msgShuttleCreate.h" publié sur le topic "" permet l'ajout d'une nouvelle navette

    subDelShuttle : un msg de type "std\_msgs/Int32.h"  publié sur le topic "" permet la suppression d'une navette

    subProductChange : un msg de type ""shuttles/msgShuttleChange.h""  publié sur le topic "" permet la maj du produit transporter par la navette

    subDestinationChange : un msg de type ""shuttles/msgShuttleChange.h""  publié sur le topic "" permet la maj de la destination de la navette

    subZoneChange : un msg de type ""shuttles/msgShuttleChange.h""  publié sur le topic "" permet la maj de la zone où se trouve la navette

    \*/

public:

    ShuttleManager();

    ~ShuttleManager();

    //Function to initialize subscribers and services

    void init(ros::NodeHandle nh);

    //Callback Function for Services

    bool getShuttleStatus(shuttles::srvGetShuttleStatus::Request &req, shuttles::srvGetShuttleStatus::Response &res);

    bool getEmptyShuttles(shuttles::srvGetEmptyShuttles::Request &req, shuttles::srvGetEmptyShuttles::Response &res);

    //Callback Function for Subscribers

    void productChangeCallBack(const shuttles::msgShuttleChange::ConstPtr& msg);

    void destinationChangeCallBack(const shuttles::msgShuttleChange::ConstPtr& msg);

    void zoneChangeCallBack(const shuttles::msgShuttleChange::ConstPtr& msg);

    void addShuttleCallBack(const shuttles::msgShuttleCreate::ConstPtr& msg);

    void delShuttleCallBack(const std\_msgs::Int32::ConstPtr& msg);

};

#endif