Projet Robot - format des messages inter agents

Novembre 2016

Résumé

Le but des ce document est de définir le format des messages qui seront échangés entre les différents acteurs (dénommés ici agent) logiciel concourant au fonctionnement du robot. Un agent est donc une entité fonctionnelle qui ne dépend pas de la façon dont elle est a priori implémentée, par exemple un même processus (thread) peut implémenter une ou plusieurs entités logicielles.
Le but de format est de décorrélé autant que faire de peut le mécanisme d'échange de message avec la ou les couches transport présentent sur le robot (par exemple la couche transport entre une entité externe au réseau, IA déportée par exemple, sera sans doute implémentée sur IP via UDP et TCP alors que la couche de communication avec l'arduino sera effectué via la liaison série via USB).
Enfin ce format unique permettra de logger de façon uniforme les échanges internes entre les différents acteurs et ainsi permettre le débogage en cas de problèmes liés aux interfonctionnements.

1 Identification des agents

Chaque agent doit être identifié par une chaîne de caractères unique dans le système.
Pour assurer l'unicité des identifications dans le système, il serait préférable de disposé d'une plat forme (librairie) qui permette a chacun des agents de se déclarer ainsi que ses méthodes d'accès. Par exemple si le protocole de transport est IP basé sur UDP et que l'on désire que chaque agent gère lui même son socket, la déclaration consistera a fournir à la plate forme l'association <nom,@IP,port UDP>). La plate forme sera alors à même de fournir une interface qui fournira pour chaque destinataire enregistré ses caractéristiques (dans l'exemple IP/UDP l'adresse et le port a utiliser).

2 type des messages

Dans le cadre du robot, 3 types de messages semblent adéquat : Les Actions qui ne nécessitent par d'acquittements, les Questions qui provoquent une Réponse de l'agent destinataire et enfin les Status qui des messages spontanés d'un agent a priori esclave afin de remonter des événements ou alarmes mais pas forcément destinées à un agent particulier.

2.1 Actions

Une action est un message destiné a un agent donné et peut donc se contenter, outre l'action elle même, d'indiquer un destinataire, l'émetteur pouvant resté anonyme (car généralement implicite).

Cependant pour des question de régularité et de débogage, il semble judicieux que l'information de émetteur soit systématique présente dans ce type de messages (ce dernier mécanisme est optionnel).

2.2 Questions/Réponses

Dans ce cas, les deux informations destinataire et l'émetteur sont nécessaire. (on pourrait envisager un mode où l'on connait implicitement l'émetteur mais cela ferait perdre en souplesse le mécanisme de communication).
Une information additionnelle qui permet de relié la question et la réponse est optionnelle, la référence. Si une référence est passée par l'émetteur alors le destinataire doit insérer cette référence dans son message de réponse.

2.3 Status

Dans ce cas de figure, le message ne doit pas comporter de destinataire car le message sera envoyé a tous les agents qui ont souscrit aux status en provenance de cet émetteur

3 encodage des message

Pour des raison de simplicité de mise en œuvre, un encodage 'textuel' des messages pourra être implémenté dans une première étape. Cette méthode est celle qui permet le plus haut degré d'inter opérabilité entre des système hétérogènes.
Un message est donc constitué des champs suivant qui sont tous des chaîne de caractères (l'ensemble des caractères autorisés étant un sous ensemble de caractère 'ascii') :

- O : émetteur
- D : destinataire
- X : référence
- A : action
- Q : question
- R : réponse
- S : status

Les caractères "{","}" et ";" seront réservés pour encapsuler le message.

3.1 exemple1 : envoi d'une action

dans cet exemple l'agent "MAITRE" envoie la commande "COM" vers l'agent "ESCLAVE". Le message sera donc

{O :MAITRE ;D :ESCLAVE ;A :COM}

Cependant rien n'interdit à l'émetteur d'insérer une référence, celle ci pouvant être ignorée par le destinataire.

{O :MAITRE ;D :ESCLAVE ;X=1 ;A :COM}

3.2 exemple2 : envoi d'une question/réponse

dans cet exemple l'agent "MAITRE" à l'agent "ESCLAVE" de lui retourner une réponse à la question "QUESTION".

{O :MAITRE ;D :ESCLAVE ;X=3 ;Q :QUESTION}
ce qui provoque la réponse suivante :
{O :ESCLAVE ;D :MAITRE ;X=3 ;R :REPONSE}

3.3 exemple3 : envoi d'une Alarme

Un agent a priori esclave ESCLAVE veut remonter un événement vers éventuellement plusieurs autres agents.

{O :ESCLAVE ;S :PANNE}

4 extension possible

On pourrait éventuellement ajouter un champ optionnel Urgence pour hiérarchiser au sens du destinataire le traitement des messages en cas de congestion.

5 implémentation possibles

5.1 plateforme

La création d'une classe plateforme (singleton) peut être envisager pour faciliter les mécanismes d'échanges. On peut opter pour différents degrés d'abstraction :

- L'agent gère de façon autonome son socket ou ses sockets de communication, il s'attribue lui même une ou des adresses IP et des ports UDP
- L'agent gère lui même son/ses sockets mais les attributs de ceux ci sont fournis par la plateforme (Intérêt de la méthode, on n'a plus de problème de double utilisation de port UDP, la gestion de ceux ci étant centralisée, la connaissance de l'adresse IP étant masquée)
- tout est géré par la plateforme qui offre une méthode de réception de message et masque complétement l'aspect socket.

Le cas qui semble le plus adapté est le second dans ce cas la plateforme offrirait est service suivant :

- declare(Nom de l'agent) retourne @IP,port à utiliser lors de la création du socket de communication de l'agent.
- declare(Nom de l'agent,IP,port) retourne void et permet d'ajouter un autre agent sur le même socket.
- get(Nom de l'agent) retourne @IP,port à utiliser pour joindre le destinataire

5.2 Parser

On peut créer une classe qui sérialise/de-sérialise les massage qui offrirait quelques constructeurs du style MSG(ORIG,DEST) ou MSG() et des méthode d'accès aux attributs :

- getDest et setDest
- getOrig et setOrig
- getAction et setAction
- ....

et deux méthodes

— Parse que prends en entrée une chaine de caractère et l'interprete
— serialize qui a partir des attributs qui ont été postionnés construit le message à envoyer.