IA01 — TD n°12
Système multi-agent

L'objectif de ce TD est de créer un certain nombre de fonctions qui permette de développer une plate-forme rustique d'agents (version 0). Pour ce faire, il est nécessaire de créer des agents, de les doter de compétences (skills), de les agiter et de visualiser les résultats.
Ce TD est la première partie d'un travail plus important qui consistera à réaliser l'exemple de vente aux enchères. La première partie consiste à développer les mécanismes de base (création d'agents, définition de la structure des messages, contrôle de fonctionnement, gestion de process séparés). La deuxième partie consistera à mettre en place les différents agents, leurs compétences, et un mécanisme de trace des messages échangés.
NOTE : Ce TD est sous-spécifié. Il vous faudra donc faire des hypothèses sur les éléments qui manquent. Il est conseillé d'essayer de proposer des choses simples, car le problème peut devenir rapidement très compliqué.

Modèle d'agent Un agent est composé de plusieurs parties :
– une partie pour la communication (input, output)
– une partie contenant les compétences (skills)
– une partie contenant les données propres de l'agent (memory)
– une partie contrôle (control)

---

1 Création d'agents et mise en place du mécanisme de contrôle

1.1 Création d'agents

Un agent sera représenté par un frame contenant un certain nombre de slots avec une seule facette ($valeur, implicite) multivaluée (note : ce n'est rien de plus qu'une a-list).

```
((name V-1)                          ; nom de l'agent
 (skills (send-catalog send-offer))  ; compétences sous forme d'une
```

```
liste de fonctions, chaque fonction réalisant une compétence
 (memory ...)                          ; structure à définir pour y
ranger les données nécessaires au fonctionnement de l'agent
 (inbox M-2 M-8)                       ; boite aux lettres en entrée
 (process <identifiant du process>)  ; process en attente sur la
boîte aux  lettres
 (comments "Agent vendeur : propose des lots selon la méthode des
enchères à la chandelle. Il offre un lot à un prix donné, si personne
n'est intéressé, au bout de 2 secondes, il diminue le prix. Le premier
acheteur qui fait une offre remporte le lot. Si personne n'a fait
d'offre et qu'un prix plancher est atteint le vendeur retire le lot
de la vente."))
```

Un agent sera créé par la fonction make-agent qui devra préciser le nom de l'agent, l'identifieur choisi et remplir les commentaires de la zone de commentaires. La syntaxe sera la suivante :

```
(make-agent :name 'V-1
     :comments "Agent vendeur : propose des lots selon la méthode des
     enchères à la chandelle. Il offre un lot à un prix donné, si
     personne n'est intéressé, au bout de 2 secondes, il diminue le
     prix. Le premier acheteur qui fait une offre remporte le lot.
     Si personne n'a fait d'offre et qu'un prix plancher est atteint
     le vendeur retire le lot de la vente.")
```

Les arguments étant repérés par des mots clés (:name, :comments), l'ordre des arguments peut être quelconque.
Cette fonction devra créer l'agent, initialiser les slots même vides, et ajouter l'identifiant de l'agent à la liste globale des agents du système *agents*, et retourner l'identifiant de l'agent. Cette fonction devra également créer un process qui se mettra en attente sur la boîte à lettre d'entrée, inbox, de l'agent (voir section contrôle). Donc, dès que la fonction sera exécutée, l'agent sera actif et prêt à recevoir des messages.

1.2 Contrôle

Le contrôle se fait de la façon suivante :
Dès que l'agent est créé un process est attaché à la boîte aux lettres d'entrée de l'agent, inbox.
- Tant que inbox reste vide, le process reste en attente.
- Dès qu'un message arrive dans la boîte, le process est réveillé et exécute le controleur : fonction control.

Comportement de la fonction control dépend du type de message reçu :
- si le message est du type "info" ou "request" la compétence correspondante devra être exécutée (à noter que rien n'oblige un agent à exécuter une compétence s'il n'en à pas envie, contrairement à un système de programmation orientée objets).
- si le message est de type "answer", l'agent devra exécuter une fonction traitant la réponse correspondant à la sous-tâche demandée,
- si le message est de type "cancel", l'agent supprimera la tâche correspondante.

Squelette de la boucle de contrôle

```
(defun scan-messages (agent &aux message)
  "the function is associated with the input mailbox of the agent
```

```
  and waits for incoming messages.
   When a message is received the scanner processes it.
Arguments:
   agent: agent-id"
  (loop
    ;; wait if no message in the mailbox
    (mp:process-wait "idle: waiting for incoming message"
      #'input-message? agent)
    ;; remove one message from the mailbox
    (setq message (get-input-messages agent))
    ;(agent-trace agent "just received: ~s"
      (message-format message))
    ;; process message
    (process-message agent message)
    ))

(defun input-message? (agent)
  "the function returns the list of input messages contained in the
  agent inbox."
   ...
   )

(defun get-input-message (agent)
  "the function returns a message taken from the list of input messages,
   and updates the list accordingly."
   ...
   )
```

La création du process en attente sur la boîte aux lettres d'entrée se fait en mettant dans la fonction make-agent l'instruction suivante :

```
 ;; launch then scanner process on input-messages queue
 (setq process-id
      (mp:process-run-function
      ; fonction de la bibliothèque ACL
       (concatenate 'string (symbol-name agent) "-scan")
       ; nom qui sera associé au process
       #'scan-messages
       ; fonction exécutée par le process
       agent))
       ; argument de la fonction précédente
```

1.3 Tests

On testera le système de la façon suivante :
- créer un agent vendeur (A-1)
- vérifier que le process associé a été lancé
- modifier manuellement l'agent A-1 en mettant une valeur (liste quelconque) dans sa boîte aux lettres
- le traitement du message consistera à l'imprimer (fonction process-message)

2 Messages

2.1 Format des messages

Les messages seront eux aussi représentés par des frames et auront un identifiant (ex : M-3). Les slots correspondants devront être au moins :
– date (temps correspondant à l'heure de création du message)
– type (request, answer, info, cancel)
– from
– to
– action (ex : for-sale, offer)
– args (ex : télé, canapé, chaises, lit, VTT, PC, vaisselle, tableau)
Exemple :

```
((:type request)(:time "23/11/01-16:12:32")(:from V1)(:to :ALL)
(:action offer)(:args (lot1 2800)))
((:type answer)(:time "23/11/01-16:14:25")(:from A1)(:to V1)(:args VTT))
```

Un message pourra être envoyé à un seul agent, ou à tous les agents du système (mot-clé :ALL dans le slot :to). On écrira une fonction de création de message

```
(defun make-message (&key type from to action args)
   "comments..."
   ...)
```

Exemple d'utilisation

```
? (make-message :type 'inform :from 'user :to 'a-1 :action 'send-catalog
:args '(catalog))
M-0
? M-0
((DATE 3213697740) (TYPE INFORM) (FROM USER) (TO A-1) (ACTION SEND-CATALOG)
 (ARGS (CATALOG)))
```

On notera que date n'est pas un paramètre du message mais correspond à l'heure à laquelle le message a été créé. Il existe une fonction qui permet d'obtenir l'heure exacte dans le système. A vous de la trouver.

2.2 Envoi des messages

On écrira une fonction d'envoi de message qui mettra le message directement dans l'inbox de chaque agent concerné.

```
(defun send-message (message)
   "comments..."
   ...)
```

Exemple d'utilisation

```
? (send-message 'm-0)
"*done*
```

2.3 Tests

On testera de la façon suivante :
– création de plusieurs agents
– création d'un message quelconque destiné à 1 agent
– envoi du message
– création d'un message destiné à tous les agents
– envoi du message (chaque agent recepteur devrait imprimer un message...)

3 Compétences

3.1 Spécification des compétences

Une compétence peut être simple (envoi d'un message de type :inform ou :cancel, ou plus compliquée comme l'envoi d'une requête avec attente d'une réponse. Dans tous les cas il faut écrire une fonction pour réaliser la fonctionnalité demandée, mais dans le deuxième cas il faut en plus prévoir une fonction qui s'exécutera au retour de la réponse. La première fonction est dite statique, la deuxième dynamique.
Une compétence (skill) sera créée par la fonction make-skill qui devra préciser le nom de la compétence, celui de l'agent, la fonction d'exécution statique et éventuellement la fonction d'exécution dynamique. La syntaxe sera la suivante :

```
(make-skill 'advertise-catalog 'V-1
    :static-fcn 'advertise-catalog)
(make-skill 'send-offer 'V-1
    :static-fcn 'send-offer
    :dynamic-fcn 'process-bid)
```

Le résultat de l'exécution de cette fonction sera de doter l'agent d'une compétence sous forme d'une sous-liste associée à la propriété skill. Cette sous-liste comprendra le nom de la compétence et les fonctions statique et dynamique. L'exécution de cette fonction remplacera les valeurs précédentes s'il y en avait. Pour le format exact, voir l'exemple. Exemple :

```
? (make-skill 'advertize-catalog 'A-1 :static-fcn 'advertize-catalog)
(ADVERTIZE-CATALOG A-1)
? A-1
((NAME A-1) (SKILLS) (MEMORY) (CURRENT-TASK) (SUBTASKS) (INBOX)
 (PROCESS #<PROCESS A-1-scan [idle: waiting for incoming message] #xBC4CEAE>)
 (COMMENTS "agent test")
 (SKILL
  (ADVERTIZE-CATALOG (STATIC ADVERTIZE-CATALOG) (DYNAMIC NIL))))
? (make-skill 'tell-price 'A-1 :static-fcn 'tell-price :dynamic-fcn
'process-new-offer)
(TELL-PRICE A-1)
? A-1
((NAME A-1) (SKILLS) (MEMORY) (CURRENT-TASK) (SUBTASKS) (INBOX)
 (PROCESS #<PROCESS A-1-scan [idle: waiting for incoming message] #xBC4CEAE>)
 (COMMENTS "agent test")
 (SKILL (TELL-PRICE (STATIC TELL-PRICE) (DYNAMIC PROCESS-NEW-OFFER))
  (ADVERTIZE-CATALOG (STATIC ADVERTIZE-CATALOG) (DYNAMIC NIL))))
```

3.2 Modification de la fonction process-message

La fonction process-message doit être modifiée pour au lieu d'imprimer le message reçu, déclencher la fonction correspondant à la compétence demandée dans le message. On distinguera les cas suivants en fonction du type de message :
– :cancel, on ne fait rien
– :answer, on applique la fonction dynamique
– :request ou :inform, on applique la fonstion statique.
On pourra utiliser la primitive case sur le type de message. Pour appliquer une fonction dont on connaît le nom on utilise la primitive apply comme suit :

```
(apply <nom de la fonction><liste d'arguments>)
```

On pourra intercaler un argument supplémentaire : le nom de l'agent comme suit :

```
(apply <nom de la fonction><nom de l'agent><liste d'arguments>)
```

La liste des arguments est celle qui se trouve dans le message (propriété args). La valeur associée à la propriété action donne la compétence requise, ce qui permet de récupérer le nom de la fonction correspondante dans la liste des compétences, selon le type de message.
Pour la mise au point de cette fonction il sera prudent de mettre un certain nombre de prints...

3.3 Tests

On créera un agent vendeur V-1, et 3 agent acheteurs A-1 A-2, et A-3. L'agent V-1 sera doté d'une compétence advertize-catalog qui consistera à envoyer la liste des objets à vendre à tous les agents acheteurs (message de type :inform). Les agents acheteurs auront une compétence de type catalog qui leur permettra de mémoriser (imprimer) les objets à vendre. L'utilisateur enverra à l'agent vendeur un message de type inform avec l'action advertize-catalog pour lancer la diffusion du catalogue.

```
? (setq mm (make-message :type :inform :from 'user :to 'v-1 :action
'advertize-catalog))
M-0
? (send-message mm)
"*done*"
V-1: advertizing-catalog
A-1 got the goodie list : (TÉLÉ VTT CANAPÉ TABLE)
A-2 got the goodie list : (TÉLÉ VTT CANAPÉ TABLE)
A-3 got the goodie list : (TÉLÉ VTT CANAPÉ TABLE)
?
```