Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 302)

Reto - Evidencia 2: Intelligent Agents

Arturo Ramos Viedas	A01636133
Manuel Alejandro Ramos Valdez	A00227837
Karen Navarro Arroyo	A01641532
Juan Diego Salcedo García	A01368540
Yahir Alexandro Rivera Huerta	A00572029

01/12/2023

Agent Communication

I. Agent Communication Language (ACL):

El sistema de mensajería de nuestro modelo está basado en KQML (Knowledge Query and Manipulation Language).
Decidimos utilizar este modelo ya que se adaptaba muy bien a nuestras necesidades y al funcionamiento interno de la simulación, la cual no requiere de algo muy complejo para poder establecer la comunicación necesaria entre agentes.

II. Estructura del mensaje

La estructura del mensaje es muy simple; para crear un mensaje, la clase que maneja el ACL (*class kqml_query*) pide varios parámetros, especificados en el siguiente punto, y después crea el formato convirtiendo estos parámetros en *strings* propios de un elemento de esta clase para después concatenarlos y devolverlo.

Parámetros:

- type
- sender
- receiver
- content
- in_reply_to

Cada parámetro puede ser cualquier *string*, solo con la excepción de *type*, pues solo puede ser "tell" y "reply".
sender debe ser el nombre de la clase del agente que está enviando el mensaje (GoodTruckAgent, BadTruckAgent o CameraAgent y su respectivo ID.
receiver es el nombre de la clase del agente al que va dirigido el mensaje. Si no está dirigido a ninguno en particular, puede dejarse como '-'.
content es el contenido del mensaje. Puede ser solo un valor o una cadena completa de texto.
in_reply_to se utiliza sólo si el agente está respondiendo a otro mensaje en particular. Si no es respuesta a ningún mensaje, puede dejarse como '-'.

III. Estructura de contenido (*content*) de los mensajes

En la simulación se manejan cuatro tipos de mensajes:

1. **Coordenadas enviadas por los agentes Truck**

```
f"Estoy en " + str(self.thisTruck.isInX) + ", " + str(self.thisTruck.isInY)
```

Para este tipo de mensajes, se toman dos parámetros propios del agente; sus

coordenadas en X y Y (*self.thisTruck.isInX*, *self.thisTruck.isInY*).
De tal forma que, por ejemplo, si el agente de la clase GoodTruckAgent con id 0 está en la posición (2, 3) del plano, el contenido del mensaje se escribirá como "Estoy en 2, 3"

2. **Aviso de cargar material**

```
"Estoy cargando material."
```

No utiliza información adicional. Este mensaje la envían sólo los agentes de tipo Truck.

3. **Pagado**

```
"Pagado"
```

Este mensaje sólo puede ser enviado por agentes de la clase GoodTruck. Es un aviso de que ya pagaron en la caseta. Como el punto anterior, tampoco tiene información adicional.

4. **Reporte de camiones vistos**

```
f"Camiones vistos: {self.see(e)}\nCamiones vistos en total: {self.counter}"
```

Este mensaje solo puede ser enviado por agentes de la clase CameraAgent. Reporta qué camiones ha visto en ese momento si han entrado en su rango de visión ({self.see(e)}) y cuántos ha visto en total ({self.counter}).

IV. Descripción y justificación de las performativas

A grandes rasgos, la simulación tiene varios camiones, unos buenos que hacen su trabajo debidamente y otros que roban material, moviéndose por un terreno, el cual, en esta simulación, funciona a manera de un plano coordenado de 20x20. Estos camiones son vistos por una cámara, la cual cuenta la cantidad de camiones que ha visto en total y cuáles ha visto (con el tipo de agente y id).

Los camiones, buenos o malos, reportan cuando están cargando material y cuando se mueven a una nueva coordenada del plano, mientras que la cámara reporta qué camiones ha visto y la cantidad total que han entrado a su rango de visión.

Decidimos hacerlo así ya que decidimos apegarnos a hacerlo como sería en un escenario real en la medida de lo posible; Las cámaras ven a los camiones (si entran en su campo de visión) y se reportan a manera de mensaje, mientras que los camiones pueden moverse por el plano y cargar en el área de carga. Los camiones buenos (GoodTruck), a diferencia de los camiones malos, pagan en caseta una vez hayan cargado y se encuentren en el límite del mapa, donde se supone que está la caseta (coordenada 20, 21). Al llegar ahí envían el mensaje de "Pagado" para informar que su trabajo está terminado.

V. Implementación

Declaración de la estructura KQML como clase

```
class kqml_query:
    def __init__(self, type, sender, receiver, content,
in_reply_to):
        types = ["tell", "reply"]
        if type not in types:
            raise ValueError(f"Type {type} is not valid.")

        self.type = type
        self.sender = sender
        self.reciever = receiver
        self.content = content
        self.in_reply_to = in_reply_to

    def __str__(self):
        res = f"({self.type}\n"
        res += f" :sender {self.sender}\n"
        res += f" :receiver {self.receiver}\n"
        res += f" :content {self.content}\n"
        if self.in_reply_to != "":
            res += f" :in-reply-to {self.in_reply_to}\n"
        res += ")"
        return res
```

Variable global para registrar los mensajes

```
Broadcast = []
```

Envío de mensajes dentro de la clase de los agentes (Cada uno de estos bloques de código está en sus respectivas clases)

```
Broadcast.append(kqml_query("tell", f"GoodTruckAgent,
ID:{self.id}", "-", "Estoy cargando material.", "-"))

Broadcast.append(kqml_query("tell", f"GoodTruckAgent,
ID:{self.id}", "-", "Estoy en " + str(self.thisTruck.isInX) + ", "
+ str(self.thisTruck.isInY), "-"))
```

```
Broadcast.append(kqml_query("tell", "Camera Agent, ID: 0",
"TruckModel", f"Camiones vistos: {self.see(e)}\nCamiones vistos en
total: {self.counter}", "-"))
```

Registrar los mensajes de Broadcast en el archivo de texto *'queries.txt'* (Al final, cuando la simulación termine de ejecutarse)

```
with open('queries.txt', 'w') as archivo:
    for m in Broadcast:
        archivo.write(str(m) + '\n')
```

Agents Interaction

I. Situaciones en las que interactúan los agentes

Ya que nuestro modelo busca apegarse a un escenario real de la mejor manera posible, la comunicación entre los agentes en este caso particular es relativamente limitada, pues no existe tal cosa como tal entre los agentes de tipo Truck.

El ejemplo más destacable de interacción entre agentes en nuestra simulación ocurre cuando un agente de clase Truck entra en el campo de visión de uno de clase Camera. Lo que ocurre en estas situaciones es que la cámara toma el id y tipo del agente que entra en su rango de visión, después lo agrega a la variable global *ts* para imprimir este registro después de terminar la simulación.

II. Tipo de interacción

El tipo de interacción más parecido a lo que estamos implementando en la simulación es de **negociación**, pues, a pesar de la limitada interacción directa que existe entre los agentes, estos pueden realizar ciertas acciones con su entorno y con otros agentes. Si bien no hay un intercambio de tipo 'entregar una cosa y obtener otra', los agentes sí 'entregan' u 'obtienen' algo en la simulación.

A pesar de esto, hay algunos casos en los que los agentes pueden hacer una acción pero que no implica la interacción con otro agente, por ejemplo, la acción de "pagar", única de GoodTruck, en la que no hay un agente en particular a quien le esté entregando el pago, sino solo cambia su estado para poder ser registrado en la variable global 'pagado', que es una lista en la que aparecen todos los camiones que han realizado esta acción, es decir, al final de la simulación todos los agentes de clase GoodTruck deben aparecer en esta lista. Otro ejemplo sería la acción de 'cargar' de los agentes de clase Truck, en los que no pagan pero sí 'reciben' la carga.