Jia Long Ji Qiu
Victor Teixidó López
Jiabo Wang
You Wu

PROBLEMA 1 SID

Decisiones de diseño

Termómetro

El agente termómetro se ha diseñado pensando en la arquitectura de agente reactivo con un estado interno, ya que la toma de decisiones hace referencia únicamente al presente (envía la temperatura actual en el momento que recibe una solicitud de un termostato, y actualiza su temperatura independientemente del historial de temperaturas) y teniendo en cuenta ciertas condiciones integradas en el agente (según si ha de ignorar o no la solicitud de un termostato).

La funcionalidad de actualización de la temperatura actual se ha implementado mediante un *ticker behaviour*, ya que es el más idóneo para poder hacer actualizaciones periódicamente. Consiste en asignar números aleatorios a un atributo privado del agente de acuerdo a la especificación del enunciado.

Para el tratamiento de solicitudes de temperatura por parte de un termostato, se ha utilizado un *cyclic behaviour* por ser el más adecuado para implementar un *"message handler"*. Cuando el agente recibe un mensaje con performativa "*REQUEST*", si el contenido es "*TEMPERATURA*", según si ha de ignorar al agente solicitante o no, o bien envía la temperatura actual, o bien el termómetro cambia de estado y el termostato solicitante deja de estar ignorado. Si el contenido es "*IGNORAR*", añade el agente remitente a un conjunto de agentes ignorados, con el objetivo de poder ignorar a dicho agente cuando envíe la siguiente solicitud de temperatura. El conjunto (Set de AIDs) es necesario para que múltiples termostatos puedan trabajar con los mismos termómetros, ya que cada termostato puede tener un criterio diferente y no interesa que se solapen las peticiones de ignorar solicitudes. En este punto cabe destacar que, para asegurar el correcto funcionamiento de la interacción termómetro-termostato, se ha hecho uso de un *"message template"* especificando una ontología común (en nuestro caso "P1") entre estos dos tipos de agentes. También es necesario el registro del agente en el "*DirectoryFacilitator"* para que este pueda ser encontrado por el agente termostato, además del desregistro en caso de eliminación del agente, con tal de mantener consistencia en el sistema.

El estado interno del agente queda representado por lo tanto en la funcionalidad anterior, donde debe tener en cuenta qué agentes debe ignorar cuando reciba una solicitud de temperatura.

Los behaviours citados anteriormente son *sub-behaviours* de un *parallel behaviour*, esto es así ya que nos interesa que el agente se pueda ejecutar de manera asíncrona y sin bloqueos.

Jia Long Ji Qiu
Victor Teixidó López
Jiabo Wang
You Wu

Termostato

El agente termostato se ha diseñado pensando en la arquitectura de agente reactivo con un estado interno, ya que la toma de decisiones hace referencia únicamente al presente, se recogen todas las temperaturas disponibles en el momento y se calcula la temperatura media en el sistema a partir de dichos valores. Además, en función del valor de las temperaturas recibidas, actuará de una forma u otra hacia los termómetros en cuestión.

El primero de los *behaviours*, la búsqueda de termómetros en el sistema y la solicitud de las temperaturas, se ha implementado en un *ticker behaviour* lo cuál nos permitirá pedir las temperaturas actuales de forma periódica. Además, esta implementación permite que se añadan o se retiren termómetros en el sistema de forma dinámica. El *ticker* en cuestión, se ejecuta cada 5 segundos, valor escogido arbitrariamente pero que nos parecía lo suficientemente razonable y adecuado para el buen funcionamiento en las condiciones en las que nos encontramos.

La segunda de las funcionalidades, la recogida de las temperaturas y el tratamiento de las mismas, la hemos implementado en un *cyclic behaviour* que estará diferenciado en las dos etapas comentadas anteriormente. Previamente, habremos almacenado en el *ticker* la cantidad de mensajes que el termostato va a recibir, sabiendo esto, tenemos un contador que, hasta que no reciba todas las temperaturas esperadas, no empezará a tratarlas. Si el número esperado de mensajes es 0, mostrará un aviso notificando del suceso y, si existe, imprimiendo la última temperatura calculada.

Una vez el termostato haya recibido todas las temperaturas, podrá empezar a tratarlas. Si la temperatura es notablemente superior a los valores esperados, le enviará un mensaje "*IGNORAR"* a dicho termómetro y no tendrá en cuenta su valor a la hora de calcular la media. En cualquier otro caso la suma se hará sin ningún tipo de distinciones. Después de calcular la temperatura final, si el resultado es menor que *a* o mayor que *b* se enviará un mensaje tal que *serviceDescription.type = "alarm-management"*. Adicionalmente, imprimirá el resultado por consola y lo almacenará como el último resultado correcto obtenido.

En el particular caso de que en la primera iteración, o todas las temperaturas sean inválidas o no hayan termómetros, se mostrará al usuario un aviso informando de que no dispone de los datos suficientes para saber la temperatura. Pese a esto, el termostato continuará realizando su funcionamiento habitual.

Jia Long Ji Qiu
Victor Teixidó López
Jiabo Wang
You Wu

Reparto del trabajo

A la hora de repartir el trabajo, hemos decidido dividir el entregable en dos tareas, el termómetro y el termostato. De la primera de las tareas se han encargado Jia Long y Jiabo Wang, y de la segunda, Victor Teixidó y You Wu.

De cara a realizar el código de funcionamiento del termómetro, hemos dividido el funcionamiento del agente en dos acciones, el actualizado periódico de la temperatura detectada y la espera a la solicitud de la temperatura por parte del termostato. De la primera de estas, el *behaviour ActualizarTemperatura*, se ha encargado Jia Long, y de la segunda, el *behaviour EsperarMensajeTermostato*, la ha hecho Jiabo Wang.

Para el agente del termostato, del que se encargaban Victor Teixidó y You Wu, hemos optado por una planificación, diseño y ejecución conjunta, debido a que la toma de decisiones no era trivial. A pesar de que en este este agente también tenemos dos acciones diferenciables, enviar solicitudes a los termómetros y tratar las temperaturas y computar la salida, las cargas de trabajo no estaban balanceadas, teniendo la segunda de estas una complejidad mayor. Reafirmando lo que hemos comentado, decidimos hacer conjuntamente ambos *behaviours* y asegurarnos así de obtener un buen resultado y una buena toma de decisiones respecto al tratamiento de las temperaturas.