Agentes *Ping* y *Pong* con **Mozart/Oz**

Juan A. Suárez Romero

11 de febrero de 2005

Resumen

El objetivo de este proyecto es medir el rendimiento de los agentes empleando la plataforma de comunicación desarrollada para el sistema **Mozart/Oz**, que permite comunicar agentes mediante KQML. Se presenta la filosofía de funcionamiento de dicha plataforma, y cómo se ha medido su rendimiento en base a unos agentes de tipo *ping* y *pong*.

0.1. Objetivos del proyecto

El objetivo del proyecto es medir el rendimiento de la plataforma desarrollada para **Mozart/Oz** en función del número de agentes que contiene. La idea consiste en crear un agente que haga un ping y otro agente que conteste con un pong a dicho agente, y realizar varios ping y pong para medir el tiempo medio de ida y vuelta (RTT, *Round Trip Time*) de los mensajes. Se irá incrementando el número de parejas para ver cómo afecta al RTT de los mensajes. Así, para n parejas el RTT será la media de los RTTs de los mensajes de cada pareja.

En orden a que las medidas sean correctas, cada pareja sólo comenzará a tomar las medidas cuando detecte que todos las parejas han sido creadas. No obstante, aunque no comiencen las medidas, si comenzarán a transmitir y recibir los mensajes tan pronto como se vayan creando, y continuarán asi hasta el final de las pruebas.

0.2. Descripción de la plataforma para Mozart/Oz

La plataforma que hemos empleado permite que agentes desarrollados en el lenguaje **Mozart/Oz**[2] puedan comunicarse entre sí utilizando el lenguaje de comunicación KQML[1].

0.2.1. Comunicación de los agentes

La comunicación entre los agentes se lleva a cabo a través de la plataforma. Para poder usar este y otros servicios de la plataforma un agente debe registrarse con un nombre. Este nombre deberá ser único para cada agente registrado a la plataforma. Una vez registrado, se le proveerá de una *fachada* a través de la cual el agente se comunicará con la plataforma. Así, para enviar un mensaje a otro agente deberá indicar el nombre de registro del agente destino y el mensaje a enviar, haciéndolo a través de dicha fachada.

Además de los nombres de los agentes registrados en una plataforma, la propia plataforma tiene un nombre que se le asigna al lanzarla. Este nombre sólo es necesario cuando un agente desea enviar un mensaje a otro agente que está en una plataforma distinta de la suya. Así, si tenemos dos agentes registrados como *ping1* y *pong1* en la plataforma *plataforma1* y un agente registrado como *pong1* en la plataforma *plataforma2*, si el agente *ping1* quiere enviar un mensaje a éste último agente deberá indicar como destinatario el agente *pong1@plataforma2*, mientras que si quiere enviarlo al agente de su propia plataforma sólo tendría que indicar *pong1*.

Otra de las características en la comunicación en esta plataforma es que para poder enviar un mensaje a una agente primero hay que establecer una conversación con él; posteriormente todos los mensajes irán dentro de dicha conversación (o de una nueva si se desea).

Un aspecto importante es que la plataforma no provee de ninguna clase a partir de la cual heredar nuestros agentes. Más bien, en el registro es necesario indicarle una función que la plataforma invocará cada vez que llegue un mensaje.

Se puede ver la relación entre los agentes y la plataforma en la figura 1.

Figura 1: Relación entre los agentes y la plataforma

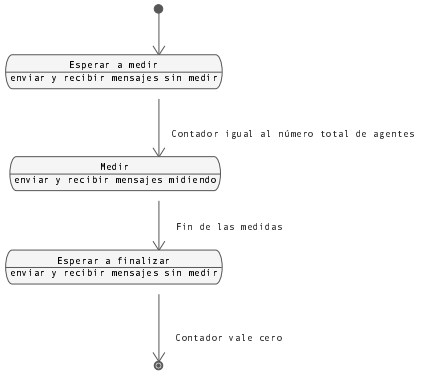

Figura 2: Fases de un agente *ping*

0.3. Diseño del programa

0.3.1. Agentes *Ping*

Como hemos comentado al principio, los agentes *ping* son los encargados de medir el rendimiento del sistema enviando y recibiendo los mensajes ping y pong. Sin embargo, no comienzan sus medidas hasta que todos ellos estén listos. Para ello los agentes comparten una clase, *Contador*, que permite su sincronización. Así, cuando un agente está listo para realizar sus medidas, incrementa este contador y comienzan a enviar y recibir los mensajes pong. Cada cierto número de envíos, comprueban si el Contador ha llegado al tope, esto es, al número total de agentes que tiene que haber. Si este es el caso, entonces es cuando comienza la medida. Una vez que ha finalizado, y para evitar que la máquina quede descargarda y desvirtuar así la medida, cada agente decrementa el contador y sigue enviando y recibiendo mensajes sin medirlos, y comprobando de vez en cuando si el contador ha llegado a cero. Este caso ocurrirá cuando todos los agentes hayan decrementado respectivamente una vez el contador, esto es, cuando hayan finalizado sus medidas. En este momento es cuando el programa finalizará. Este proceso se encuentra esquematizado en las figuras 2 y 3. Nótese que en esta última las fases de `Iniciar medida` y `Finalizar medida` sólo tienen sentido cuando se está midiendo los tiempos.

0.3.2. Agentes *Pong*

Estos agentes son más sencillos de de diseñar. Su único objetivo es permanecer a la escucha de mensajes, y cuando se reciba uno se conteste al agente emisor. Se puede ver la actividad de estos agentes en la figura 4.

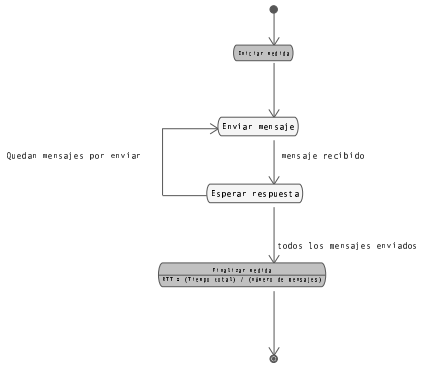

Figura 3: Fases en el envío de un mensaje

Figura 4: Actividad de un agente *pong*

0.3.3. Funcionamiento global

Cada agente *ping* recibe un identificador numérico, que va desde 1 hasta el número total de agentes *ping* que se creen. De igual forma se hace con los agentes *pong*, de forma que el agente *ping i* sólo se comunicará con el agente *pong i* correspondiente. Así, primero se crean los agentes *pong* y posteriormente los agentes *ping*.

A la hora de hacer las medidas, hemos posibilitado que o bien todos los agentes residan dentro de la misma máquina virtual, como que estén en máquinas separadas: la plataforma en una máquina virtual, todos los agentes *ping* en otra y todos los agentes *pong* en otra. Por ello se han creado tres clases nuevas: *Cliente*, que lanza tantos agentes *ping* como se indique, *Servidor*, que lanza tantos agentes *pong* como se indique, y *ClienteyServidor*, que lanza tantas parejas *ping* y *pong* como se indique dentro de la misma máquina virtual. Se puede ver en la figura 5.

Independientemente de si se ejecutan las pruebas en una o en más máquinas virtuales, el proceso siempre es el mismo: lanzar primero los agentes *pong*, luego los agentes *ping* y realizar las medidas, tal y como se ve en la figura 6.

En la figura 7 podemos ver una secuencia de funcionamiento de un agente

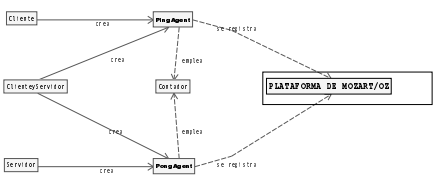

Figura 5: Diagrama de clases del proyecto

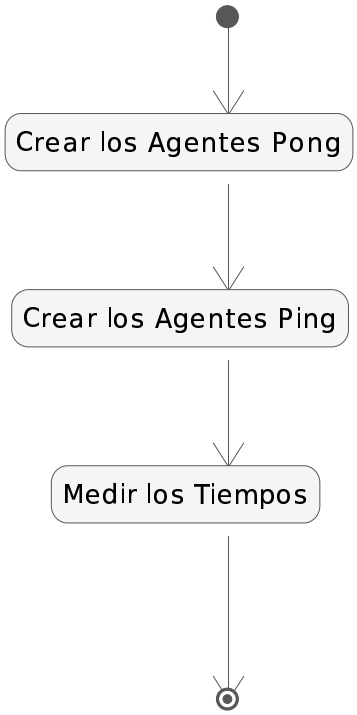

Figura 6: Fases en la ejecución del programa

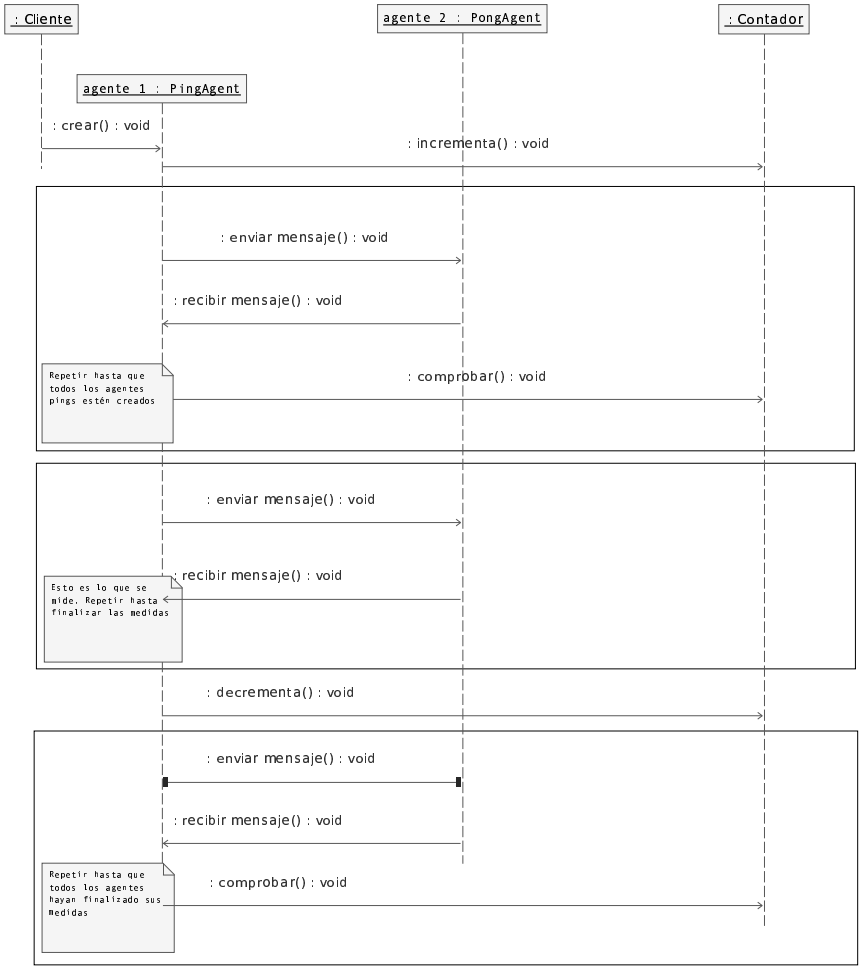

Figura 7: Funcionamiento de un agente *ping*

ping en conjunción con un agente *pong*.

0.4. Carga de la máquina

En la sección anterior hemos descrito los agentes que hemos empleado para medir el rendimiento de la plataforma desarrollada para **Mozart/Oz**. En la descripción hemos indicado que los agentes *ping* envían un mensaje y quedan esperando la respuesta; de igual forma los agentes *pong* esperan un mensaje y contestan con una respuesta. En ambos casos el proceso se repite continuamente.

En un sistema real, los agentes no quedan simplemente esperando por la respuesta a un mensaje, ni la respuesta es un simple mensaje, sino que mientras esperan realizan alguna tarea, y la respuesta suele ser fruto de alguna operación que se realiza.

Por ello se ha creado paralelamente a las clases *PingAgent* y *PongAgent* (que

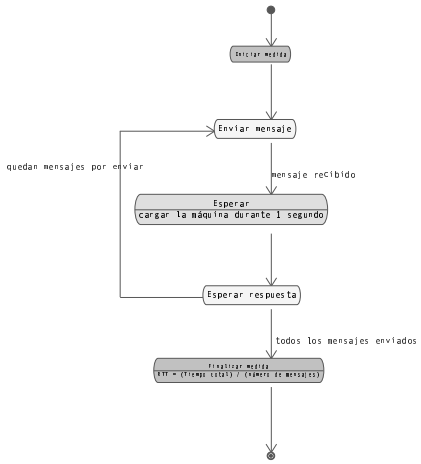

Figura 8: Tarea del agente *PingAgentLoad*

se corresponde con los agentes mencionados antes) las clases *PingAgentLoad* y *PongAgentLoad*. La diferencia entre los agentes *PingAgent* y *PingAgentLoad* es que éste último ejecuta una tarea entre el envío del mensaje y la espera por una respuesta (ver figura 8), mientras que la diferencia entre los agentes *PongAgent* y *PongAgentLoad* es que éste último calcula la respuesta en función de una tarea (ver figura 9). En ambos casos se ha aportado un módulo denominado *FullLoad* con un procedimiento que carga la máquina durante el tiempo estipulado (en nuestro caso durante 1 segundo).

De igual forma, para poder ejecutar estos nuevos agentes se han creado, a semejanza de antes, las clases *ClienteLoad*, *ServidorLoad* y *ClienteyServidorLoad*.

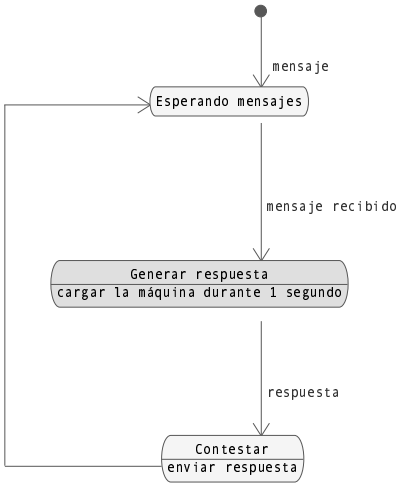

Figura 9: Tarea del agente *PongAgentLoad*

Bibliografía

[1] Tim Finin, Jay Weber, Gio Wiederhold, Mike Genesereth, Don McKay, Rich Fritzson, Stu Shapiro, Richard Pelavin, and Jim McGuire. Specification of the KQML Agent-Communication Language – plus example agent policies and architectures, 1993.

[2] Peter Van Roy and Seif Haridi. Mozart: A programming system for agent applications. In *International Workshop on Distributed and Internet Programming with Logic and Constraint Languages*, 1999.