Agentes *Ping* y *Pong* con **JACKAL**

Juan A. Suárez Romero

10 de febrero de 2005

Resumen

El objetivo de este proyecto es medir el rendimiento de los agentes empleando la plataforma **JACKAL**, 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*.

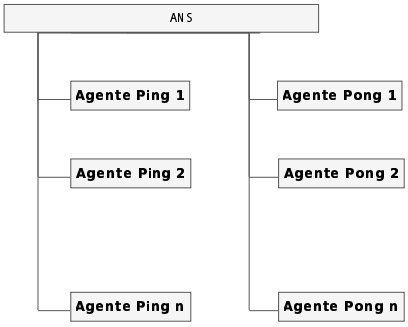

Figura 1: Organización de los agentes

0.1. Objetivos del proyecto

El objetivo del proyecto es medir el rendimiento de la plataforma **JACKAL** 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 JACKAL

JACKAL[1] es una herramienta basada en Java para la comunicación de agentes empleando el lenguaje de comunicación KQML[2].

0.2.1. Organización de los agentes

Los agentes en **JACKAL** se organizan jerárquicamente. Así, con sólo conocer los datos del agente que se encuentra en la raíz, se pueden comunicar con el resto de los agentes que forman parte del árbol. En nuestro caso, el agente raíz se conoce como *ANS*, y de él cuelgan tanto los agentes de tipo *ping* como tipo *Pong* (ver figura 1). Así el agente *Ping1* solo necesita conocer la máquina y el puerto en el que se encuentra el agente *ANS*. Si se quiere comunicar con el agente *Pong1* simplemente se indicará el nombre completo de este agente, esto es, que se quiere comunicar con el agente *Pong1.ANS*. El sistema automáticamente irá descendiendo por la jerarquía hasta llegar al agente destino.

Figura 2: Comunicación de los agentes con la plataforma **JACKAL**

0.2.2. Comunicación de los agentes

La comunicación entre los agentes se lleva a cabo a través de la plataforma. Ésta provee un intercomunicador que los agentes emplean para interaccionar con ella. Es a través de este intercomunicador como envían y reciben los mensajes del resto de los agentes. Puede verse un ejemplo en la figura 2.

El intercomunicador actúa también como un buzón de correo: los mensajes que llegan al agente quedan almacenados en un buzón hasta que el agente los atiende. En nuestro caso, los agente *ping*, una vez que han enviado un mensaje al correspondiente agente *pong*, quedarían escuchando el buzón hasta que llegase el mensaje de respuesta. Aunque **JACKAL** permite establecer filtros de los mensajes a escuchar, en nuestro caso esperamos hasta que llegue un mensaje cualquiera, y suponemos que es el mensaje pong del agente, sin comprobarlo.

Un aspecto importante es que la plataforma **JACKAL** permite que cualquier programa pueda establecer una comunicación con otro programa mediante KQML. Es por ello por lo que no proporciona una clase de tipo *Agente* de la cual construir los agentes. Por tanto tanto los agentes *ping* como los agentes *pong* derivan todos ellos de la clase de Java *Thread*.

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 3 y 4. Nótese que en esta última las fases de `Iniciar medida` y `Finalizar medida` sólo tienen sentido cuando se está midiendo los tiempos.

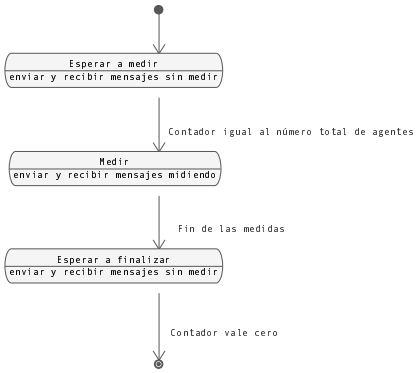

Figura 3: Fases de un agente *ping*

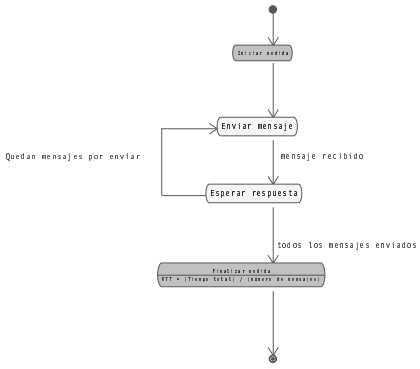

Figura 4: Fases en el envío de un mensaje

Figura 5: Actividad de los agentes *pong*

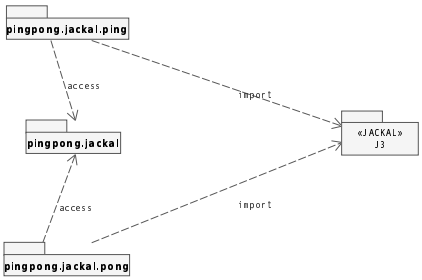

Figura 6: Paquetes que forman el proyecto

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 5.

0.3.3. Funcionamiento global

El programa está formado por tres paquetes: *pingpong.jackal.ping*, que contiene las clases que forman al agente *ping*, *pingpong.jackal.pong*, que contiene las clases que forman al agente *pong*, y *pingpong.jackal*, que contiene otra serie de clases útiles (ver figura 6).

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. Además, también hemos aportado la clase *Ans*, que constituye el agente raíz de la jerarquía, y que se puede invocar desde la línea de comandos (si separamos en distintas máquinas virtuales los agentes) o desde la propia clse *ClienteyServidor* si todo va a ejecutarse dentro de la misma máquina virtual. Se puede ver en la figura 7.

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 8.

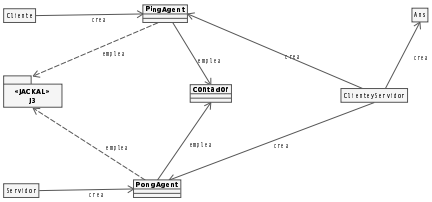

Figura 7: Diagrama de clases del proyecto

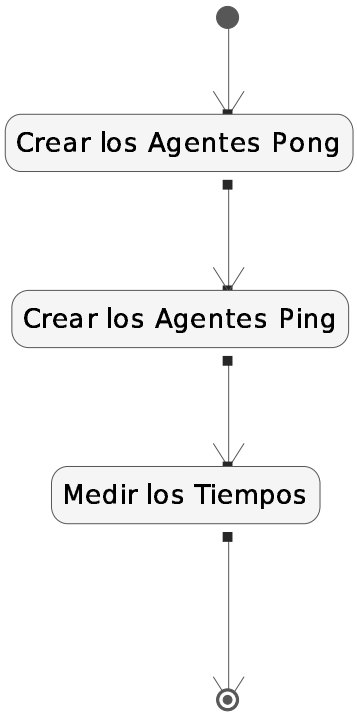

Figura 8: Esquema de funcionamiento

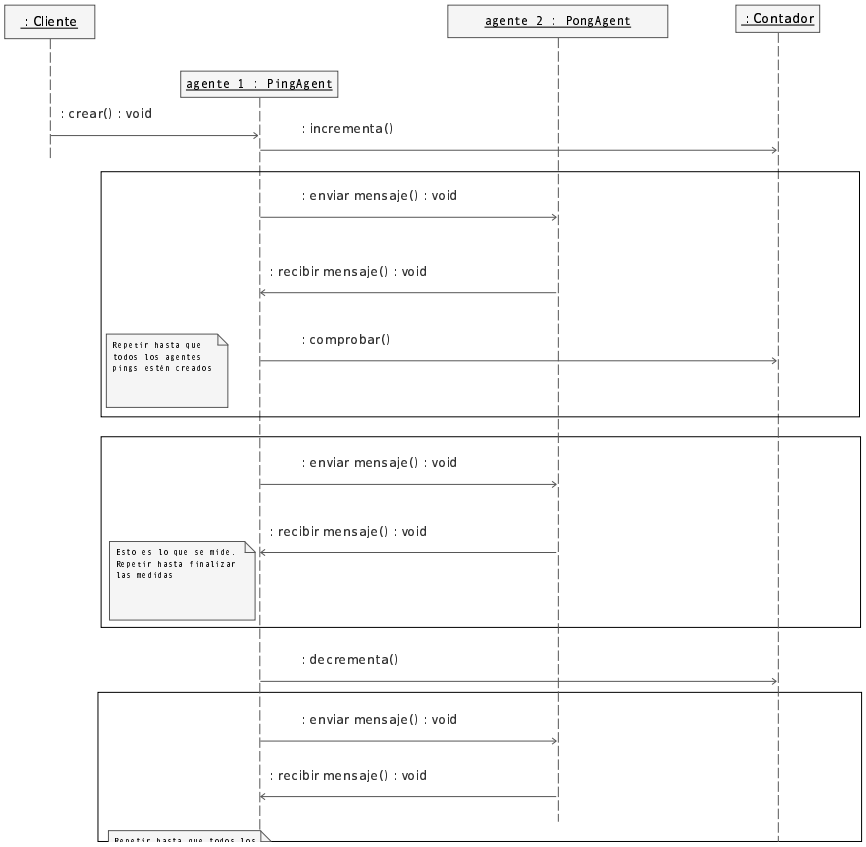

Figura 9: Funcionamiento de un agente *ping*

En la figura 9 podemos ver una secuencia de funcionamiento de un agente *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 **JACKAL**. 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

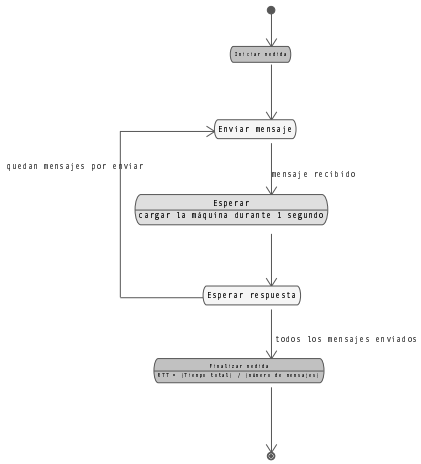

Figura 10: Tarea del agente *PingAgentLoad*

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 10), mientras que la diferencia entre los agentes *PongAgent* y *PongAgentLoad* es que éste último calcula la respuesta en función de una tarea (ver figura 11). En ambos casos se ha aportado una clase denominada *FullLoad* con un método 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*.

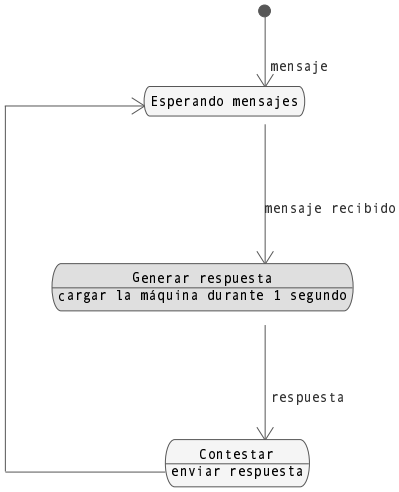

Figura 11: Tarea del agente *PongAgentLoad*

Bibliografía

[1] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughanam. Jackal: A Java-based Tool for Agent Development. In *AAAI-98, Workshop on Tools for Agent Development*, Madison, WI, 1998.

[2] 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.