Agentes *Ping* y *Pong* con **JATLite**

Juan A. Suárez Romero

22 de febrero de 2005

Resumen

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

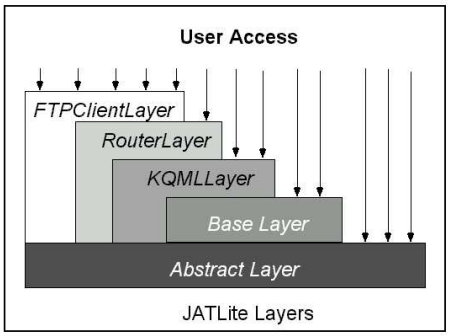

Figura 1: Estructura en capas de **JATLite**

0.1. Objetivos del proyecto

El objetivo del proyecto es medir el rendimiento de la plataforma **JATLite** 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 así hasta el final de las pruebas.

0.2. Descripción de la plataforma JATLite

JATLite[3] es una herramienta basada en Java para la comunicación de agentes empleando el lenguaje de comunicación KQML[1].

JATLite está formada por varias capas que permiten decidir en qué nivel queremos empezar a desarrollar los agentes (véase figura 1). En nuestro caso hemos decidido empezar desde la capa de Enrutamiento (*Router Layer*). En este nivel, esta capa provee de un agente simple llamado *RouterLayer.Router.RouterAction* que, mediante un sencillo fichero de configuración, permite definir en qué puerto se puede registrar un agente y cual es el puerto para comunicarse con el enrutador. De esta forma, cada vez que creemos un agente tendremos que especificarle la dirección del enrutador y la dirección del registrador (además de la dirección del propio agente que creamos).

Siguiendo en este nivel, para crear un agente que trabaje con la capa de enrutamiento creamos una clase que herede de la clase *RouterClientAction*. Entre otros métodos, en esta subclase nueva debemos redefinir un método (*Act*)que será el invocado por el enrutador cada vez que llegue un nuevo mensaje.

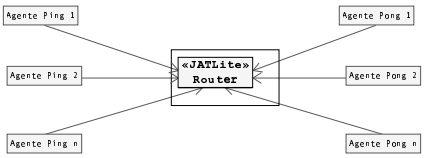

Figura 2: Relación entre los agentes y la plataforma

0.2.1. Comunicación de los agentes

Como hemos mencionado anteriormente, en **JATLite** la comunicación se realiza a través de los enrutadores. Así, primero creamos los enrutadores (en nuestro proyecto sólo hemos tenemos uno al que se conectan todos los agentes) y posteriormente creamos los distintos agentes que se conectarán al enrutador. Una de las facilidades que provee el enrutador es el almacenamiento persistente de los mensajes cuando el agente destinatario no está disponible. Así, si un cuando un agente envía un mensaje a otro agente que en ese momento no se encuentra disponible, el enrutador almacenará su mensaje para entregarlo cuando el agente se encuentre ya disponible.

Se puede ver la relación entre los agentes y la plataforma en la figura 2.

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.

Como vemos, existen claramente tres fases en los agentes: esperar a que todos los agentes estén listos, realizar las medidas, y esperar a que todos los agentes hayan finalizado sus medidas. Hemos empleado el patrón de diseño Estado[2] para crear tres comportamientos: *PingStartingState*, *PingMeasuringState* y *PingEndingState*. El funcionamiento en cada estado es el siguiente: al iniciarse el estado se envía un mensaje ping al agente correspondiente. El enrutador de la

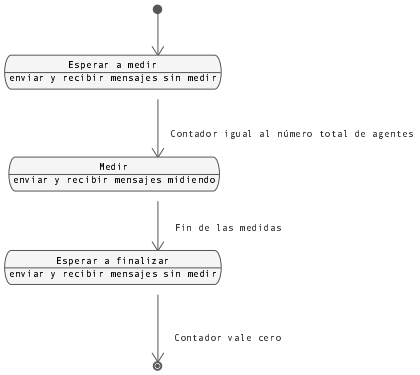

Figura 3: Fases de un agente *ping*

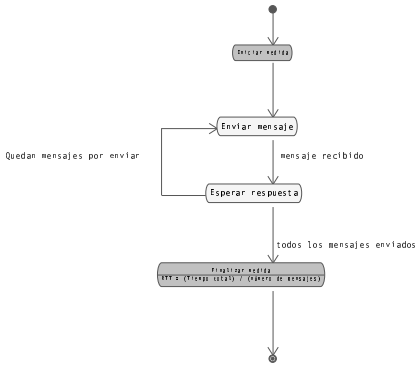

Figura 4: Fases en el envío de un mensaje

plataforma **JATLite**, al recibir la respuesta, invoca al método *Act* del agente, el cual delega en el método *Act* del estado. Este método comprueba si es necesario cambiar al siguiente estado, en cuyo caso cambiaría; en caso contrario volvería a lanzar otro ping, y así sucesivamente. Así, el primer estado sería PingStartingState, luego pasaría a PingMeasuringState y finalmente al estado PingEndingState.

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.

Figura 5: Actividad de un agente *pong*

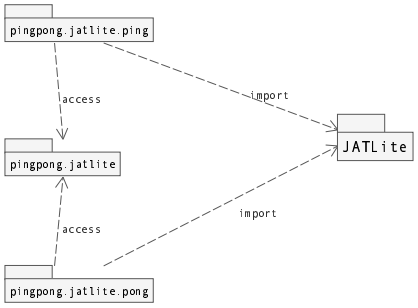

Figura 6: Paquetes que forman el proyecto

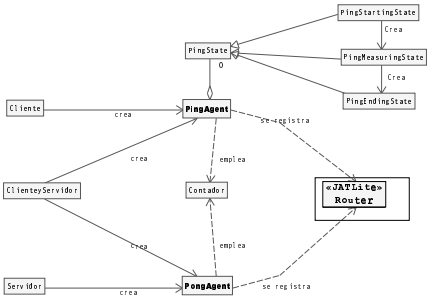

Figura 7: Diagrama de clases del proyecto

0.3.3. Funcionamiento global

El programa está formado por tres paquetes: *pingpong.jatlite.ping*, que contiene las clases que forman al agente *ping*, *pingpong.jatlite.pong*, que contiene las clases que forman al agente *pong*, y *pingpong.jatlite*, 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. 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.

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

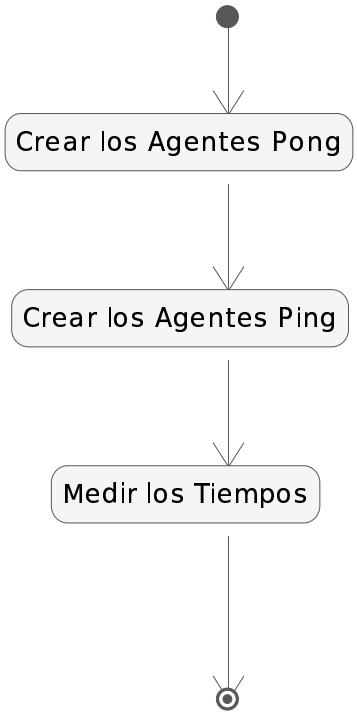

Figura 8: Esquema de funcionamiento

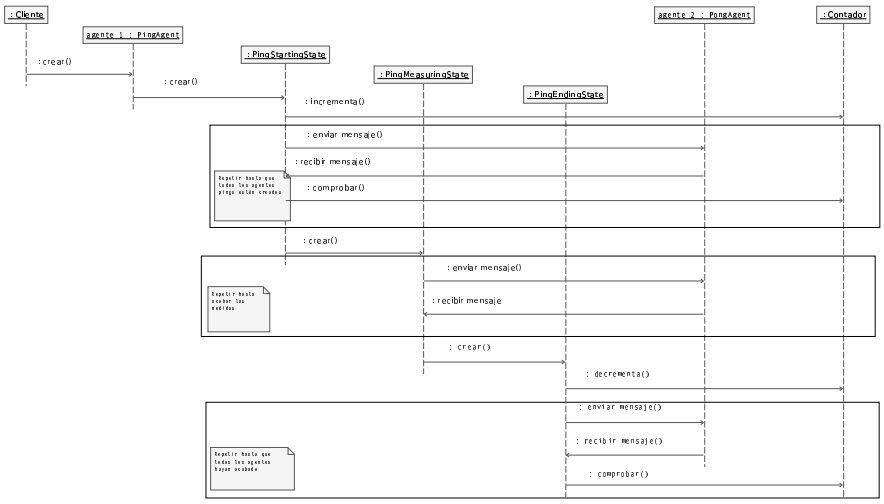

Figura 9: Funcionamiento de un agente *ping*

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 **JADE**. 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 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 nuevo, puesto que empleamos comportamientos, hemos creado los comportamientos *PingMeasuringStateLoad*, *PingStartingStateLoad* y *PingEndingStateLoad*.

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

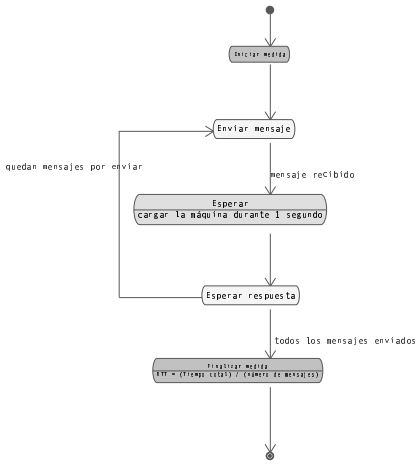

Figura 10: Tarea del agente *PingAgentLoad*

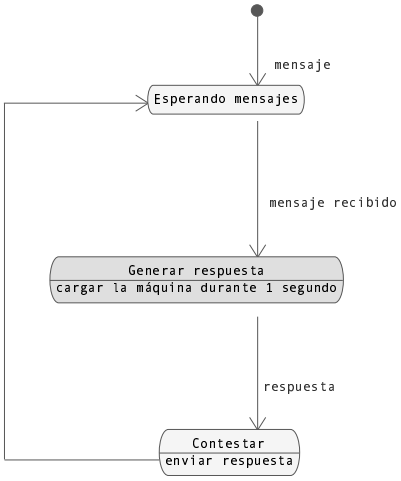

Figura 11: Tarea del agente *PongAgentLoad*

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] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[3] Heecheol Jeon, Charles Petrie, and Mark R. Cutkosky. JATLite: A java agent infrastructure with message routing. *IEEE Internet Computing*, 4(2):87–96, 2000.