DERMI: Middleware para aplicaciones de trabajo en grupo descentralizadas

DERMI: Middleware for Decentralized Workgroup Applications

C. Pairot, P. García, R. Rallo et al.

Resumen

Presentamos DERMI, un middleware para el desarrollo de aplicaciones de trabajo en grupo descentralizadas construido sobre una red peer-to-peer que modela una tabla de dispersión distribuida. DERMI proporciona numerosos servicios tales como invocaciones remotas síncronas y asíncronas, localización de objetos descentralizada, movilidad de objetos, intercepción distribuida y dos nuevas abstracciones de invocación denominadas *anycall* y *manycall*.

Palabras clave: Middleware, P2P, DHT, objetos distribuidos.

Summary

In this paper we present DERMI, a middleware for the development of decentralized workgroup applications, built on top of a peer-to-peer network that models a distributed hash table. DERMI provides numerous services such as synchronous and asynchronous remote method invocations, decentralized object location, object mobility, distributed interception and two new invocation abstractions, named *anycall* and *manycall*.

Keywords: Middleware, P2P, DHT, Distributed Objects.

DERMI es un middleware para el desarrollo de aplicaciones de trabajo en grupo descentralizadas construido sobre una red peer-to-peer que modela una tabla de dispersión distribuida

1.- Introducción

Los sistemas descentralizados P2P están cobrando mayor importancia debido tanto al creciente aumento del ancho de banda disponible como al de los recursos computacionales de las microcomputadoras. Se está produciendo así una transición de modelos cliente/servidor tradicionales a modelos descentralizados que aprovechan los recursos de los clientes y reducen el clásico cuello de botella de los sistemas centralizados. Además, la aparición de las, hoy en día, ya clásicas aplicaciones P2P como Napster, SETI@Home, KaZaA o eMule ha propiciado la generalización de este paradigma.

Recientemente las arquitecturas P2P se han visto revolucionadas gracias a la aparición de las denominadas Tablas de Dispersión Distribuidas (Distributed Hash Tables o DHTs). Dichas arquitecturas descentralizadas, a diferencia de precedentes como Gnutella, definen topologías organizadas (típicamente en forma de anillo, como es el caso de Pastry [1]) que permiten mejoras sustanciales en la localización de recursos y enrutado de mensajes en la red P2P. Más concretamente, permiten localizar cualquier recurso en la red en O(log n) saltos, siendo n el número total de nodos existentes en la red.

Sobre una red P2P con arquitectura DHT (Pastry), hemos construido un middleware de objetos distribuidos denominado DERMI. Utiliza las DHTs como mecanismo de localización de recursos y un sistema de multicast a nivel de aplicación denominado Scribe para la propagación de eventos.

2.- Servicios ofrecidos por DERMI

DERMI se beneficia del sistema de publicación/suscripción proporcionado por Scribe para modelar las llamadas a métodos como eventos y suscripciones en el middleware. Existe un prototipo de implementación en http://ants.etse.urv.es/DERMI.

La intercepción distribuida es un servicio novedoso que nos permite reconectar y localizar interceptores de tipos compatibles en tiempo de ejecución en una aplicación distribuida

Nuestra aproximación ofrece los servicios tradicionales del middleware orientado a objetos como invocación síncrona y servicio de nombres, y añade servicios nuevos como la movilidad de objetos, replicación y cachés de objetos, intercepción distribuida, invocación asíncrona uno-a-muchos y un modelo de localización descentralizada. Además DERMI ofrece dos abstracciones de invocación nuevas: anycall y manycall. Dichas abstracciones son adaptaciones al middleware de objetos distribuidos de los servicios de anycast y manycast ofrecidos por Scribe. Nos centraremos en describir los servicios más importantes y novedosos proporcionados por DERMI.

2.1.- Localización descentralizada de objetos

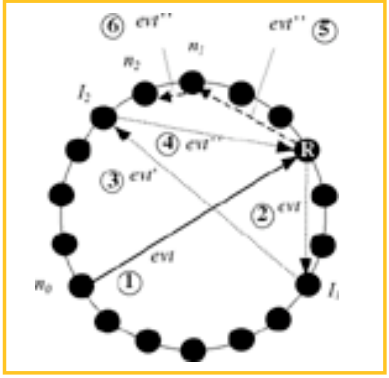
Hemos implementado un servicio de localización de objetos totalmente descentralizado que nos permite encontrar las referencias de los objetos para después ejecutar llamadas sobre ellos. Los *handles* de los objetos son insertados en el DHT haciendo un hashing del identificador del objeto. Para encontrarlo simplemente haremos un hashing de su identificador y seguidamente obtendremos su *handle* para poder efectuar llamadas a sus métodos.

2.2.- Intercepción distribuida

La intercepción distribuida es un servicio novedoso que nos permite reconectar y localizar interceptores de tipos compatibles en tiempo de ejecución en una aplicación distribuida. Siguiendo un modelo similar al de Java RMI, el nuestro nos permite crear *stubs* y *skeletons* propios para las clases remotas que pueden interceptar llamadas a un determinado objeto remoto en ejecución. Para no tener que cambiar las suscripciones de los *skeletons* interceptores ni de los objetos remotos interceptados cada vez que añadimos o quitamos un interceptor, hemos extendido las clases del servidor de eventos Scribe para que soporte nativamente esta característica.

La implementación de la intercepción distribuida en Scribe se ha conseguido gracias al hecho de que todos los eventos que se envían a un grupo multicast primero se enrutan a su *rendez-vous point* [2] o raíz, para después ser diseminados a lo largo de los miembros del grupo. Así pues, cada grupo multicast contiene una lista de punteros a otros objetos interceptores que se actualiza cada vez que se añade o elimina un nuevo interceptor. Para evitar perder la información de la cola de interceptores en caso de que el nodo que la alberga dejara de funcionar, se replica en los *k* nodos más cercanos utilizando un sistema de almacenaje y replicación como PAST [4].

INTERCEPCIÓN DISTRIBUIDA



Consecuentemente, cada vez que un evento se envía a un grupo multicast, la notificación primero llega al *rendez-vous point* donde comprobamos si existen interceptores; si no existen, el evento se envía directamente a todos los miembros del grupo y si existen se pasa secuencialmente por todos los interceptores que transformarán el evento para que finalmente sea enrutado de nuevo a la raíz, que lo entregará, debidamente transformado, a todos los miembros del grupo.

El objeto en el nodo n_0 envía un evento al grupo cuya raíz es R. Este evento se envía a la cola de interceptores secuencialmente (I_1 I_2), transformándolo (*evt_evt'_evt''*). Finalmente, el evento se envía de vuelta a la raíz que lo entrega al resto de suscriptores del grupo (n_1, n_2).

2.3.- Anycall/Manycall

Además de las típicas invocaciones síncronas y asíncronas, DERMI proporciona otras formas de realizar llamadas remotas a procedimientos (RPCs) utilizando notificaciones que hemos llamado *anycall* y *manycall*. Estas dos nuevos tipos de RPCs se construyen utilizando la primitiva *anycast* [3] implementada en el servicio de eventos Scribe, que permite a un nodo enviar un mensaje a un miembro de un grupo cercano a él.

Una llamada *anycall* implica enviar una notificación *anycast* a un grupo multicast. Esto hará que el miembro del grupo más cercano al cliente ejecute la llamada si se cumple una determinada condición. En este caso, al cliente le da igual qué miembro del grupo llame, sólo está interesado en encontrar alguien que pueda satisfacer la condición de la llamada. En caso de que el mensaje llegue a la raíz o rendez-vous point del grupo multicast, el anycall no ha podido ser satisfecho por nadie y se comunica al cliente mediante una excepción.

Un ejemplo útil de la utilización de una llamada *anycall* sería la obtención de datos para una simulación, donde los datos que deben ser analizados podrían estar dispersos entre diferentes servidores agrupados todos ellos en un grupo multicast. Cuando un cliente quisiera descargarse datos para analizar, simplemente haría un *anycall* al grupo para obtener un paquete de datos. En este caso vemos que al cliente le da igual de qué servidor descargue los datos, sólo le interesa que el más próximo a él sea el que se los proporcione.

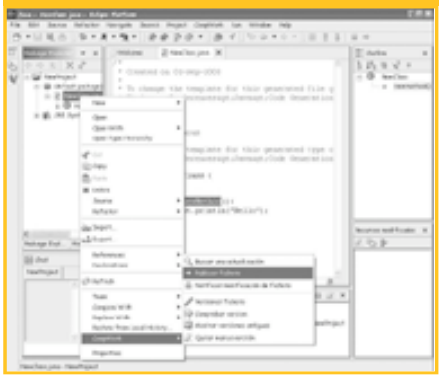
El caso de una llamada *manycall* es una variación del *anycall*. Ahora, la llamada se ejecuta cuando *n* miembros del grupo satisfacen una condición global. Es similar a un *anycall* en el sentido de que cuando un objeto recibe un mensaje *manycall*, primero comprueba si satisface una condición local y después de llamar a su procedimiento remoto comprueba si satisface la global. Si es así, la llamada *manycall* ha tenido éxito. En caso contrario, el mensaje se va enrutando a través de los restantes miembros del grupo hasta que se satisface la condición o hasta que se llega a la raíz del árbol multicast.

El desarrollo de software en grupo ha seguido tradicionalmente un enfoque cliente/servidor y nosotros planteamos aquí un modelo más flexible y descentralizado

3.- Prueba de concepto: CoopWork

Uno de los objetivos claves de nuestra investigación es diseñar el middleware necesario para construir aplicaciones colaborativas multiusuario sobre arquitecturas descentralizadas. Así, hemos escogido como prueba de concepto el desarrollo de software colaborativo descentralizado. El desarrollo de software en grupo ha seguido tradicionalmente un enfoque cliente/servidor (CVS-Concurrent Versioning System, SourceForge) y nosotros planteamos aquí un modelo más flexible y descentralizado.

ASPECTO DEL ENTORNO COOPWORK



Hemos desarrollado el entorno CoopWork como un plugin del entorno de desarrollo Eclipse [5]. CoopWork se construye sobre el middleware DERMI y se beneficia de sus servicios de localización de recursos, transmisión de eventos, invocación uno-a-muchos e invocación *anycall* y *manycall*.

Estamos trabajando en el diseño y desarrollo del middleware necesario para la creación de sistemas de colaboración y trabajo en grupo sobre redes descentralizadas

CoopWork integra una serie de herramientas necesarias para poder trabajar conjuntamente de manera más efectiva como: servicios de presencia, un sistema de chat, una lista de recursos modificados, un historial de versiones publicadas en el grupo y un sistema de avisos, y mecanismos para compartir el código que se está escribiendo.

4.- Conclusiones y trabajo futuro

Actualmente estamos trabajando en el diseño y desarrollo del middleware necesario para la creación de sistemas de colaboración y trabajo en grupo sobre redes descentralizadas. Los mecanismos de tablas de dispersión distribuidas y overlay multicast nos están permitiendo ofrecer servicios interesantes para la creación de aplicaciones. Además, estamos en vías de crear un anillo español Pastry entre diferentes universidades españolas (Rovira i Virgili, Murcia, Valladolid, UPC) para experimentar y desarrollar sistemas de trabajo en grupo y plataformas software descentralizadas basadas en nuestro middleware DERMI.

Referencias

[1] A. Rowstron y P. Druschel, "Pastry: Scalable, Decentralized Object Location and Routing for Large-Scale Peer-to-Peer Systems". *Proceedings IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pp. 329-350, Noviembre 2001.

[2] M. Castro, P. Druschel, A. M. Kermarrec y A. Rowstron, "SCRIBE: A Large-Scale and Decentralized Application-Level Multicast Infrastructure", *IEEE Journal on Selected Areas in Communications*, Vol. 20, No. 8, Octubre 2002.

[3] M. Castro, P. Druschel, A. M. Kermarrec y A. Rowstron, "Scalable Application-Level Anycast for Highly Dynamic Groups", NGC 2003, Munich, Alemania, Septiembre 2003.

[4] A. Rowstron y P. Druschel, "Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility", *ACM Symposium on Operating Systems Principles*, Octubre 2001.

[5] Eclipse IDE, http://www.eclipse.org

[6] DERMI Website, http://ants.etse.urv.es/DERMI.

Este trabajo ha sido parcialmente financiado por el proyecto MCyT TIC2003-09288-C02-00.

Carles Pairot, Pedro García
(cpairot@etse.urv.es), (pgarcia@etse.urv.es),
Robert Rallo, Rubén Mondéjar
(rrallo@etse.urv.es)
Dpto. d'Enginyeria Informàtica i Matemàtiques
ETSE - Universitat Rovira i Virgili
Antonio F. Gómez Skarmeta
(skarmeta@dif.um.es)
Dpto. de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia