Uso de interceptores CORBA para dar soporte a objetos replicados*

Francesc D. Muñoz i Escoí, Pablo Galdámez Saiz, José M. Bernabéu Aubán
Institut Tecnològic d'Informàtica
Universitat Politècnica de València
e-mail: {*fmunyoz, pgaldam, josep*}*@iti.upv.es*

Resumen

Los objetos interceptores permiten que se pueda modificar el contenido de los mensajes transmitidos entre un cliente y un objeto destino en un entorno CORBA. Utilizando sus servicios es posible proporcionar soporte para objetos replicados, de manera que su uso resulte transparente al programador.

Se analiza el uso de interceptores en Visibroker 3.2 y cómo se puede introducir parte del soporte desarrollado en nuestra arquitectura HIDRA para poder implantar objetos replicados sobre este ORB. El sistema resultante, al carecer de un monitor de pertenencia a grupo y un protocolo de cuenta de referencias, no ofrece las mismas garantías que el desarrollado sobre HIDRA, pero puede ser útil para implementar algunas aplicaciones.

1 Introducción

Uno de los objetivos principales de la arquitectura HIDRA [3] es proporcionar soporte para objetos altamente disponibles. En esta arquitectura se toma un ORB [7] como base y se modifica su propio núcleo para dar el soporte necesario a objetos replicados. Además, el ORB se integra en el sistema operativo y el resultado final permite construir un sistema distribuido en *cluster*, donde se pueden utilizar objetos altamente disponibles en la implementación de algunos componentes básicos del propio sistema.

En HIDRA se ha tomado como modelo de replicación principal al *coordinador-cohorte* [1], por introducir una carga más baja que el modelo *activo* [8] y tener un tiempo de recuperación inferior al modelo *pasivo* [2]. Para que este modelo se comporte correctamente hay que incluir soporte adicional que garantice la atomicidad y consistencia de las invocaciones sobre objetos replicados. Esto se consiguió con el diseño de un protocolo de invocación fiable (ROI [5]) complementado con un mecanismo de control de concurrencia (HCC [4]).

La experiencia obtenida en el diseño de HIDRA puede servir para implantar en otros ORB, un soporte similar al descrito. Sin embargo, en el caso de HIDRA resultó necesario modificar el núcleo del ORB y esto no va a ser posible en un ORB *comercial*, del que no se puede obtener el código fuente para efectuar las modificaciones necesarias. Aún así, la revisión 2.2 de

*Este trabajo ha sido parcialmente financiado por la Comisión Interministerial de Ciencia y Tecnología bajo el proyecto TIC 96-0729.

la especificación de CORBA, incluye por primera vez un mecanismo para modificar el comportamiento de un ORB: la utilización de *interceptores* [7, Capítulo 18]. Con un interceptor se permite modificar y filtrar las peticiones, tanto en el dominio cliente como en el dominio servidor. Utilizando interceptores, será posible construir el soporte para implementar los dos protocolos (ROI y HCC) que se necesitan para el modelo de replicación *coordinador-cohorte* tal como se está implementando en HIDRA.

El resto del artículo se ha estructurado como sigue. La sección 2 describe los interceptores con más detalle, mientras en la sección 3 se describen los interceptores en un ORB concreto: Visibroker. Seguidamente pasamos a explicar como soportar el modelo coordinador-cohorte mediante interceptores. La sección 5 da algunos detalles sobre la implementación realizada del protocolo de invocaciones fiables sobre Visibroker. Para terminar, en la sección 6 aparecen las conclusiones.

2 Interceptores CORBA

Los interceptores CORBA son objetos que el núcleo del ORB llamará cuando se realice una invocación y que van a poder filtrar de esta manera el flujo de información entre el cliente y el servidor.

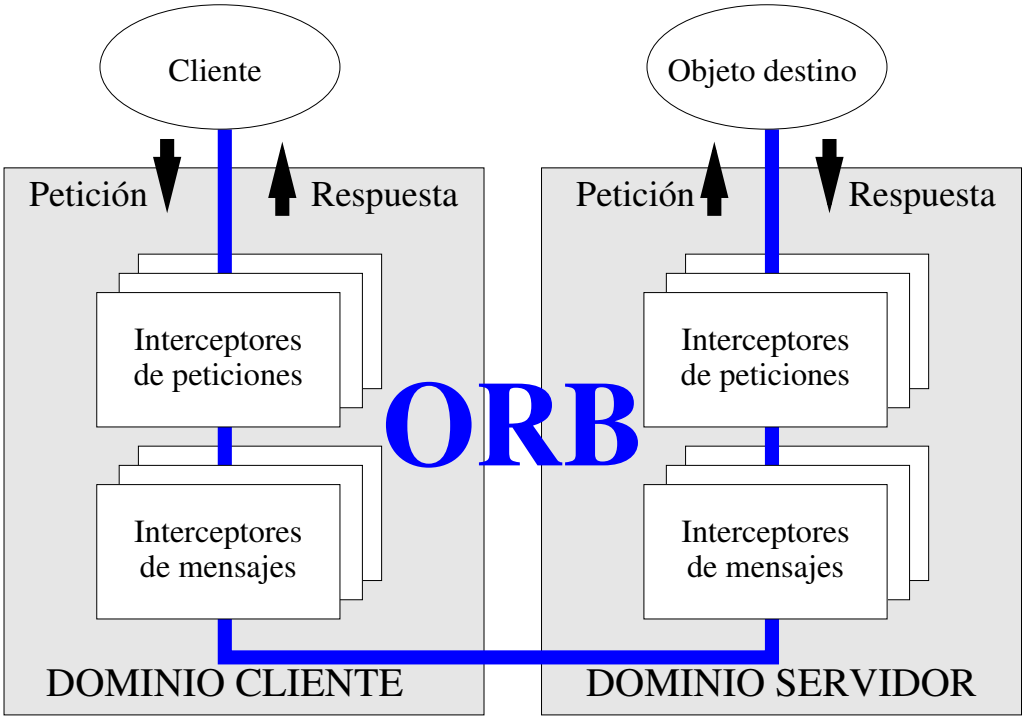

Figura 1: Interceptores llamados durante una invocación.

El estándar CORBA define dos tipos de interceptores (véase la figura 1):

- *Interceptores de peticiones*. Los interceptores de peticiones pueden ubicarse tanto en la parte cliente como en la servidora. El ORB llama a un interceptor de peticiones cliente pasándole como argumento la petición a enviar al servidor, permitiendo que la modifique. Lo mismo sucede en el dominio servidor, antes de entregar la petición a su objeto destino.

- *Interceptores de mensajes*. Los interceptores de mensajes reciben cada uno de los mensajes que emite el ORB entre los objetos cliente y servidor. El ORB indica además, cuál va a ser el objeto destino del mensaje enviado. Al igual que para las peticiones, el interceptor puede modificar el contenido del mensaje.

En la práctica, los objetos interceptores también mantendrán cierto estado. Es posible que un interceptor registre el número de invocaciones efectuadas entre un determinado par de objetos cliente-servidor, que establezca temporizadores para controlar la duración de estas invocaciones, que cifre el contenido de los mensajes en el cliente y los descifre en el servidor, que añada o elimine parte del mensaje, etc.

En nuestro caso, vamos a aprovechar la funcionalidad ofrecida por los interceptores para que éstos creen y mantengan ciertos objetos auxiliares, transfiriendo sus referencias entre las réplicas de los objetos que formen la aplicación. Esto permitirá la implementación del protocolo de invocaciones fiables y el de control de concurrencia para objetos replicados bajo el modelo coordinador-cohorte.

3 Interceptores en Visibroker

La versión 3.2 de Visibroker [9] soporta únicamente interceptores de mensajes. No existe ningún tipo de interceptor de peticiones. Para los interceptores de mensajes, en lugar de respetar las interfaces sugeridas en [7], Visibroker proporciona diferentes métodos, según el mensaje del protocolo IIOP que deba filtrarse. Además, estos interceptores se clasifican en los siguientes tipos:

- *Interceptores de enlace*. Pueden filtrar los mensajes necesarios para que un cliente obtenga una referencia a un objeto servidor, cuando utiliza el servicio de nominación.

  Usando este interceptor se puede lograr que el cliente obtenga una referencia distinta a la que ha solicitado o que utilice un servidor de nombres diferente al incluido en Visibroker.

- *Interceptores clientes*. Son los interceptores que se pueden instalar en el dominio cliente. Pueden filtrar los siguientes eventos: *preparación de envío* (para modificar la cabecera del mensaje de petición, variando su destino), *envío de petición* (para modificar el contenido del mensaje con la petición), *fallo en el envío* (cuando el ORB advierte que ha sido imposible hacer llegar el mensaje a su destino), *éxito en el envío* (cuando se ha conseguido entregar el mensaje a su destino), *recepción de respuesta* (pudiendo modificar el mensaje de respuesta antes de que lo obtenga el cliente) y *fallo en la recepción de respuesta* (cuando no se ha recibido ninguna respuesta durante el periodo de tiempo especificado por el cliente).

- *Interceptores servidores*. Son los interceptores que se pueden instalar en el dominio servidor. Pueden filtrar los siguientes eventos: *recepción de petición* (se puede modificar el contenido del mensaje y su cabecera. Modificando la cabecera se puede conseguir que el mensaje sea redirigido hacia otro objeto destino), *preparación de respuesta* (se puede modificar la cabecera del mensaje de respuesta, con lo que éste llegaría a un objeto diferente), *envío de respuesta* (se puede modificar el contenido del mensaje de respuesta), *fallo en el envío de respuesta* (si no hay comunicación con el cliente) y *petición finalizada* (cuando el ORB confirma que el cliente ya ha recibido la respuesta).

A estos métodos hay que añadir los que se pueden emplear para contestar las peticiones de enlace por parte de los clientes.

Con este tipo de interceptores va a resultar muy fácil controlar la comunicación entre clientes y servidores, modificándola si es preciso. Además, aunque los interceptores son llamados por el propio núcleo del ORB, en Visibroker es posible efectuar las siguientes acciones dentro de un interceptor:

A1 Crear nuevos objetos con interfaz IDL.

A2 Añadir las referencias de los objetos creados en los mensajes que se estén filtrando.

A3 Utilizar el servicio de nominación de Visibroker para obtener referencias a objetos registrados en él. Aunque hay que tener cuidado y no hay que filtrar este tipo de invocaciones.

A4 Invocar a cualquier objeto con interfaz IDL mientras se esté efectuando el filtrado del mensaje. De nuevo, hay que evitar que estas invocaciones sean también filtradas.

Estas cuatro características van a permitir que incluyamos el soporte al modelo de replicación coordinador-cohorte dentro de los interceptores.

4 Soporte para el modelo coordinador-cohorte

En el modelo de replicación coordinador-cohorte, cada cliente puede elegir una determinada réplica (la que actúa como coordinadora para esa invocación) del objeto para realizar la petición. Esta réplica la procesa localmente y, cuando lo cree necesario, realiza uno o más *checkpoints* sobre el resto de réplicas (los cohortes de esa invocación) y devuelve la respuesta al cliente.

La principal ventaja de este modelo reside en la posibilidad de que existan coordinadores diferentes para múltiples peticiones concurrentes, con lo que el objeto replicado puede servir simultáneamente varias invocaciones. Por supuesto, para evitar que se den inconsistencias, es necesario asegurar que estas múltiples invocaciones concurrentes no modifiquen la misma parte del estado del objeto. Para ello, basta con adoptar un mecanismo de control de concurrencia especial, que evite que puedan ser servidas a la vez dos invocaciones en conflicto (es decir, aquéllas que accedan a la misma parte del objeto y donde al menos una de ellas intente modificarlo).

4.1 El modelo coordinador-cohorte en HIDRA

Para poder dar soporte a este modelo de replicación, en HIDRA se han añadido algunas características al núcleo del ORB utilizado como la base de esta arquitectura. Estas características adicionales son:

C1 Inclusión de un nuevo tipo de referencia a objeto para acceder a objetos con este modelo de replicación. La referencia ha de permitir que el cliente que la utilice tenga un coordinador "preferido" y que, en caso de que éste falle, pueda ser sustituido automáticamente por otra réplica.

C2 Desarrollo de un protocolo de transferencia de estado a las nuevas réplicas que sean añadidas al objeto.

C3 Gestión de una cuenta de referencias por cada objeto existente en el sistema. Esto se da tanto para los objetos simples como para los replicados. Aunque la gestión de esta cuenta de referencias conlleva tener que realizar tareas adicionales cada vez que se transfiere una referencia a objeto, el uso de la cuenta es básico para eliminar aquellos objetos que ya no están siendo utilizados. Antes de eliminar a un objeto sin referencias cliente, nuestro ORB le envía una *notificación de no-referencia*.

El uso de la cuenta de referencias y de su notificación asociada simplifica bastante algunos de los protocolos necesarios para facilitar soporte a objetos replicados.

C4 Desarrollo de un protocolo que garantice la atomicidad y progreso de las invocaciones realizadas sobre objetos replicados. Este es el protocolo de invocaciones fiables (ROI), presentado en [5].

Con él se garantiza que una invocación iniciada por un cliente sobre un objeto replicado siempre va a terminar mientras queden réplicas del objeto invocado (progreso). Además, la invocación ha de ser atómica (es decir, o afecta a todas las réplicas o no afecta a ninguna). Para ello, únicamente se aborta una invocación cuando han fallado tanto el cliente como el coordinador y ningún cohorte ha recibido ningún *checkpoint* relacionado con ella.

C5 Desarrollo de un protocolo de control de concurrencia. Este protocolo (HCC) ya fue presentado en la pasada edición de las Jornadas [4]. Con él se evita que dos invocaciones que estén en conflicto puedan proceder a la vez sobre el mismo o sobre diferentes coordinadores.

En un ORB estándar no podremos encontrar nada de lo anterior. Por tanto, se tendrá que proporcionar el soporte a estos cinco apartados mediante el código a instalar en los interceptores.

4.2 Soporte con interceptores CORBA

Veamos de qué forma se puede proporcionar el soporte descrito en los puntos C1 a C5 utilizando interceptores. En algunos casos no será posible proporcionar un servicio exactamente igual al descrito en HIDRA (por ejemplo, para el caso de la cuenta de referencias), por lo que la implementación basada en interceptores tendrá algunas limitaciones.

4.2.1 Uso de referencias para objetos replicados

Para poder proporcionar el soporte a objetos replicados hay que poder gestionar de una manera adecuada las referencias a estos objetos. Para ello, hay que efectuar algunos cambios importantes, como son:

- Crear un servicio de nominación que extienda al que actualmente proporciona el ORB. Para ello se creará una nueva clase de contextos de nominación que darán el soporte necesario a los objetos replicados.

  Este soporte se centra en admitir múltiples peticiones de asociación de un mismo nombre a distintos objetos. Si estos objetos cumplen la misma interfaz, entonces se aceptará el registro y pasarán a verse como réplicas del mismo objeto. En ese caso, el nuevo servicio

de nominación iniciará la ejecución del protocolo de adición de una nueva réplica para que ésta pueda recibir el estado adecuado.

Si, por el contrario, los objetos servían interfaces diferentes, la petición de asociación se interpretará como una colisión de nominación y la nueva petición será rechazada con una excepción.

Con este nuevo tipo de servicio de nominación (que presentará la misma interfaz especificada en [6, Capítulo 3], aunque con operaciones adicionales, para poder obtener también secuencias de referencias asociadas a un mismo nombre), se mantendrá un conjunto de referencias asociado a cada nombre registrado (una por cada réplica). Las aplicaciones únicamente utilizarán las operaciones presentadas en las interfaces de la especificación estándar, tanto para registrar como para resolver nombres.

- Cuando un interceptor cliente observe que la petición que deseaba efectuar no ha tenido éxito (en Visibroker, el interceptor recibe una notificación del evento *fallo en el envío*), deberá solicitar al servicio de nominación que le facilite otra referencia ligada al mismo objeto. Como el nuevo servicio de nominación puede averiguar rápidamente si existen otras réplicas, en caso de que haya alguna otra, devolverá su referencia y descartará la que ha fallado. El interceptor cliente no avisará de ninguna manera al programa cliente que ha intentado efectuar la invocación. Simplemente guardará en una tabla la equivalencia entre la referencia fallida y la que se ha obtenido para sustituirla. Cuando se vuelva a intentar una invocación sobre ese objeto replicado, el interceptor cliente consultará la tabla y sustituirá la referencia (Esto se lleva a cabo cuando el ORB notifica el evento *preparar petición*).

  Una vez el núcleo de Visibroker ha notificado el fallo en el envío al interceptor cliente, intenta de nuevo reenlazar la referencia con el objeto, por lo que aquí entra en acción el interceptor de enlace. Cuando se le notifique al interceptor de enlace que se desea reenlazar la referencia, éste podrá obtener una nueva referencia válida utilizando nuestro servidor de nombres especial. Con ello se puede acceder a otra réplica y el problema se soluciona.

  En todo este proceso, la aplicación cliente que inició la invocación no ha sido advertida para nada del fallo que ha ocurrido. Visibroker únicamente genera una excepción cuando el intento de reenlace también fracasa. En nuestro caso, esto sólo ocurrirá cuando hayan fallado todas las réplicas del objeto que se deseaba invocar.

- El servicio de nominación deberá adoptar algún criterio para devolver una referencia u otra cuando sus clientes intenten resolver el nombre de un objeto replicado. Lo ideal sería poder devolverle a cada cliente la referencia de la réplica más cercana, pero esto va a resultar difícil.

  Puede utilizarse una estrategia de turno rotatorio para devolver a cada petición una réplica distinta y repartir de forma equitativa el número de clientes que utilizarán cada una de las réplicas como coordinadoras de sus peticiones.

4.2.2 Protocolo de adición de nuevas réplicas

Para poder añadir una nueva réplica a un objeto replicado en este modelo, hay que garantizar que ésta pueda recibir una copia del estado que le permita recibir peticiones a partir de ese

momento, tanto con el papel de coordinadora como con las funciones de cohorte.

El problema principal es que si hay algunas peticiones que se están llevando a cabo en el instante en que se solicita la adición, estas no podrían completarse adecuadamente si durante su desarrollo se incluye la nueva réplica. Por tanto, la adición de una nueva réplica será vista como una operación especial de los objetos replicados que tendrá que ser serializada por el mecanismo de control de concurrencia y que será incompatible con todas las demás invocaciones. Con esto se garantiza que:

- No haya ninguna invocación en curso cuando la adición se efectúe. Las que habían sido iniciadas previamente habrán sido completadas y las que llegaron después a los objetos serializadores se suspenderán y no serán iniciadas hasta que la nueva réplica haya sido añadida.

- Aquellas invocaciones que el serializador retrase hasta después de la adición, verán todas ellas el mismo conjunto de réplicas. Se garantiza que todas estas réplicas tengan el mismo estado.

Como vemos, la adición de una nueva réplica no plantea ningún problema adicional sobre lo que ya se está efectuando en HIDRA. El bloqueo de las invocaciones iniciadas concurrentemente con la adición y la elección del instante en que se efectúa la transferencia del estado dependen principalmente del mecanismo de control de concurrencia empleado.

4.2.3 Cuenta de referencias

Con los interceptores de enlace se puede controlar cuándo se obtiene una referencia cliente para cierto objeto. Sin embargo, para llevar la cuenta no basta con controlar cuándo debe incrementarse. Para controlar los decrementos habría que detectar cuándo se descarta o libera una referencia y cuándo falla un cliente. Para estas dos tareas, los interceptores de enlace resultan inútiles.

Para aquellos objetos que necesiten cuenta de referencias, se puede implementar una solución dentro del propio código de estos objetos replicados, en una interfaz de la que ellos hereden.

Para efectuar este control:

- Cada réplica del objeto debería mantener una referencia para cada una de las otras réplicas existentes, junto al valor de la cuenta de referencias clientes. De hecho, cada réplica podrá verse también como un cliente de las restantes.

- Cada vez que un cliente obtenga una referencia para un objeto de este tipo, efectuará un registro de su referencia en el objeto al que se refiere. Este registro ha de renovarse periódicamente invocando a cierta operación de "refresco" para indicar que la referencia va a seguir utilizándose. Aquellas referencias para las que no se ha efectuado la invocación de esta operación especial dentro del periodo fijado, son eliminadas de la cuenta.

  Además de esto, existirá otra operación para efectuar la liberación explícita de la referencia.

Con este soporte, el fallo de un dominio cliente se detectará fácilmente, ya que en el dominio servidor se dejarán de recibir las invocaciones de renovación de la referencia y, por tanto, se podrá actualizar adecuadamente la cuenta.

Cuando una réplica observa que su cuenta ha bajado a cero, llama a todas las demás para obligar a que se dé una notificación de no-referencia. Cuando las demás reciben esta llamada, se anotan que su cuenta es cero y que la notificación se ha enviado. A continuación se envía la notificación y se contesta a la réplica que inició la llamada. Por último, esta réplica envía también la notificación. Esto puede que notifique incorrectamente la ausencia de referencias, pero garantiza que todas las réplicas adopten la misma decisión (excepto cuando algunas hayan quedado aisladas debido a una partición de la red).

Para solucionar estos problemas, sería necesario incluir un monitor de pertenencia a grupo. Sin embargo, ya que no se tiene el control para decidir qué referencias son válidas o no, y no se puede modificar el núcleo del ORB que da el soporte a interceptores para poder incorporar la información obtenida por este monitor, habrá que descartar esta solución.

Por tanto, con la solución basada en mensajes periódicos de renovación de referencias, se tiene una cuenta aproximada. Los defectos de esta solución son:

- Se necesitan mensajes por cada objeto replicado que exista y por cada cliente que tenga acceso a ellos. Con un monitor de pertenencia que controle las máquinas en funcionamiento, el número de mensajes es mucho menor y la información obtenida puede emplearla el ORB para saber qué objetos servidores están todavía en funcionamiento y cuáles fallan cuando un nodo cae.
- Se pueden tomar decisiones incorrectas en casos de fallos múltiples.
- La reconfiguración del estado de un objeto replicado se complica un poco al no tener protocolos de reconfiguración dirigidos por el monitor de pertenencia.

Como consecuencia, el soporte ofrecido para objetos replicados utilizando interceptores únicamente será útil si las aplicaciones que lo utilizan no tienen un elevado número de objetos replicados (porque en ese caso, la cuenta de referencias exigiría un alto número de mensajes) y el estado de éstos es fácilmente transmisible (porque de ese modo la reconfiguración de los objetos replicados en casos de fallos y adiciones será sencilla).

4.2.4 Protocolo de invocación fiable

La implementación de este protocolo mediante interceptores CORBA se describe en la sección 5.

4.2.5 Protocolo de control de concurrencia

El protocolo de control de concurrencia no exige ningún soporte especial que deba ser proporcionado por los interceptores. El único problema a resolver es que habrá un conjunto de objetos (el serializador y sus agentes) que deben ser invocados antes de que una invocación llegue a su destino. Estas invocaciones a los serializadores serán efectuadas desde los interceptores servidores. Como ya hemos comentado en la sección 3, desde un interceptor va a ser posible invocar a un objeto con interfaz IDL, con lo que no se van a tener dificultades para implementar este punto. Además, el propio interceptor puede bloquear la invocación suspendiendo al hilo que lo esté ejecutando, con lo que será posible sincronizar adecuadamente a las invocaciones fiables que se estén controlando.

La información adicional necesaria para contruir el CCS (la matriz de compatibilidad de operaciones) puede ser obtenida por el nuevo servicio de nominación cuando se registre un

objeto replicado. El servicio de nominación sería entonces el encargado de crear un serializador inicial y de registrar en él, y en sus agentes, todos los CCS a medida que sea necesario.

4.2.6 Otras consideraciones

Para poder instalar los interceptores, independientemente de su tipo, se necesita implementar objetos "factorías". Visibroker facilita llamadas para poder registrar estos objetos factorías, de manera que cuando se obtenga una referencia o se registre un objeto en el ORB se puedan crear los interceptores de enlace, clientes o servidores correspondientes.

En nuestro caso, únicamente nos interesa tener interceptores para tratar las invocaciones de los objetos replicados, pero no querremos asociarlos con objetos simples. Para ello, la factoría de interceptores clientes facilita en su método de creación la referencia a la que puede asociarse el interceptor. Se exigirá que esa referencia esté asociada a un objeto que satisfaga cierta interfaz para llegar a asociarle el interceptor cliente correspondiente. La jerarquía de clases utilizada se presenta en la figura 2. En ella se observa que únicamente se utilizan tres interfaces (*Replicated*, *ReplicatedRC* e *InterReplicated*) para dar soporte a replicación. *Replicated* es la que utiliza internamente el nuevo servicio de nominación para averiguar si el objeto que se está registrando es replicado o no, *ReplicatedRC* es la que incorpora la gestión de la cuenta de referencias, mientras que *InterReplicated* es la que se utiliza como marcador para decidir qué tipo de interceptor ha de asociarse al objeto. En caso de que un objeto herede de la interfaz *InterReplicated*, se asume que los interceptores que se le asociarán llevan incluido el soporte para el protocolo de invocaciones fiables (ROI), mientras que si no se hereda de ella, el soporte para este protocolo no es necesario. Esto se debe a que para implementar el protocolo ROI resulta necesario emplear algunos objetos auxiliares replicados, pero para dar soporte a sus invocaciones no podemos emplear el propio protocolo de invocación fiable.

Por tanto, podemos tener objetos que hereden de *ReplicatedRC*, como es el caso de los objetos replicados auxiliares que se utilizan en el protocolo ROI, donde hace falta llevar una gestión de su cuenta de referencias. También habrá objetos que hereden únicamente de *InterReplicated* si son objetos replicados con garantías de invocación fiable pero sin necesidad de cuenta de referencias. Por último, tendremos objetos que hereden de ambas interfaces; estos serán los que necesiten cuenta de referencias y las garantías ofrecidas por el protocolo de invocación fiable.

Por último, los interceptores servidores siempre son instanciados por su factoría cuando se crea un nuevo objeto y se registra en el ORB. Lo único que podemos hacer, a medida que llegan invocaciones sobre ese objeto, es comprobar qué interfaces satisface el objeto asociado. Así, si el objeto ni siquiera satisface la interfaz *Replicated*, el interceptor servidor no toma ninguna acción especial y devuelve en seguida el control al ORB. Por contra, si satisface la interfaz *Replicated* el interceptor efectuará alguna labor que dependerá de si también satisface o no la interfaz *InterReplicated*. En el primer caso, se proporciona el soporte asociado al protocolo ROI, en el segundo no es necesario.

5 Implementación del protocolo de invocación fiable

El protocolo de invocación fiable a objetos de HIDRA se describe en detalle en [5]. En esta sección únicamente se presentarán los objetos necesarios para su implementación y cómo ha podido llevarse ésta a cabo utilizando interceptores.

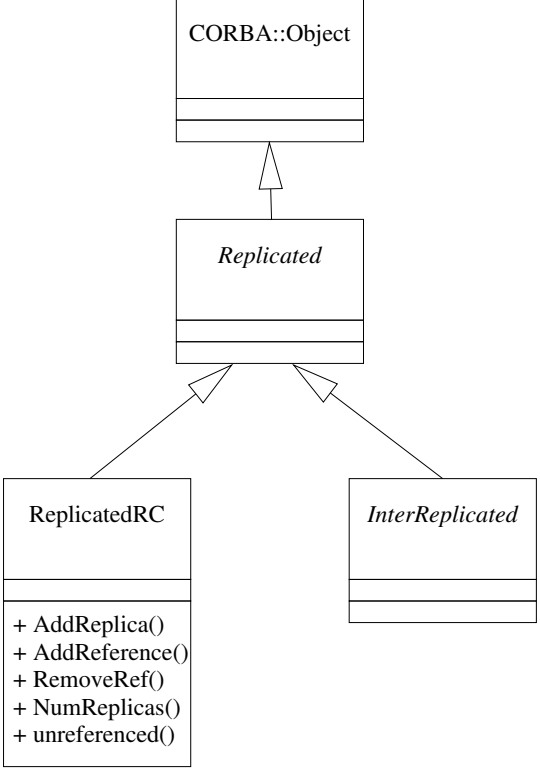

Figura 2: Jerarquía de clases para dar soporte a objetos replicados.

5.1 Objetos auxiliares

Para poder implantar este protocolo se necesitan los siguientes objetos auxiliares:

- **RoiID**. Identificación de invocación fiable. Permite identificar una invocación, de manera que resulte sencillo detectar reintentos de una misma invocación cuando se ha producido el fallo del objeto coordinador utilizado en el intento anterior.

  Este objeto únicamente se replica cuando el cliente que inicia la invocación también es un objeto replicado. De esta manera, si falla la réplica cliente que inició la invocación, otras réplicas podrán tomar su papel y concluir su trabajo.

- **TObj**. Objeto de terminación. Este objeto es creado inicialmente por el interceptor servidor asociado a la réplica coordinadora para esta invocación. Sirve para identificar cuándo la invocación ha concluido su trabajo en todas las réplicas del objeto invocado. Cuando reciben el primer *checkpoint* asociado a esta invocación, los interceptores servidores de las réplicas cohortes crean réplicas de este objeto. Además, el objeto serializador utilizado por el protocolo HCC también mantiene una de sus réplicas.

  Este objeto recibe la notificación de no-referencia cuando la invocación ha terminado en todos los dominios servidores. Con ello, el serializador sabe que ya puede dar paso a la siguiente invocación que esperase su terminación.

- **CObj**. Objeto de confirmación. También es creado por el interceptor servidor asociado a la réplica coordinadora de la invocación. En este caso, únicamente los cohortes crean réplicas de este objeto, pero el cliente ha de recibir también una de sus referencias.

Se utiliza para comprobar cuándo el cliente ha obtenido el resultado final de la invocación. Al detectarse esta situación, los interceptores servidores asociados a las réplicas del objeto invocado ya pueden desechar la copia de los resultados que mantienen para evitar repeticiones del servicio de la invocación en caso de fallo.

5.2 Implementación

En los siguientes puntos se pasa a describir el protocolo de invocaciones fiables a objeto, así como los problemas que ha sido necesario resolver para implantarlo utilizando interceptores CORBA:

1. *Creación del RoiID*. El interceptor cliente comprueba si el objeto a invocar satisface la interfaz *InterReplicated*. En caso de ser así, creará una instancia de la clase RoiID (gracias a las características A1 y A2 citadas en la sección 3) e incluirá su referencia en el mensaje enviado hacia el objeto destino.

   Nótese que la clase RoiID hereda únicamente de *ReplicatedRC* puesto que necesita la gestión de la cuenta de referencias para saber cuando puede eliminarse este objeto. Sin embargo, no puede heredar de *InterReplicated* porque de ser así necesitaría a su vez otros objetos RoiID para controlar sus invocaciones.

2. *Creación de los CObj y TObj*. Una vez la invocación llega al dominio servidor, el ORB invoca las operaciones correspondientes del interceptor servidor. Éste verifica que el objeto destino de la invocación satisfaga la interfaz *InterReplicated*. De ser así, se crean las primeras réplicas de los objetos CObj y TObj. Una vez creadas, se envía una referencia del CObj al RoiID, utilizando su operación destinada a tal efecto. También se llama al objeto serializador y éste suspende la llamada hasta que la invocación original pueda proceder al no haber ninguna otra invocación activa en conflicto con ella.

   Esto es factible gracias a la característica A4 citada anteriormente. Para ello, los objetos serializadores no exigen que se emplee el protocolo ROI para poderlos invocar.

3. *Inicio de checkpoints*. Cuando el interceptor servidor retoma el control, tras haber consultado al serializador, devuelve el mensaje sin su referencia al RoiID (que ha sido extraída) al ORB, quien lo hará llegar hasta su objeto destino.

   Eventualmente, el objeto destino inicia un *checkpoint* hacia las réplicas cohortes para esta invocación (podemos utilizar el interceptor cliente para difundir la petición a todos los cohortes). En este *checkpoint* hay que incluir una referencia al RoiID, al CObj y al TObj, pero esto debe hacerlo el interceptor cliente.

   Nótese que esto sucede en el dominio de la réplica coordinadora, pero la creación de estos objetos sucedió en el interceptor servidor, mientras que ahora su inclusión en el cliente debe llevarla a cabo el interceptor cliente. Para lograr esta transmisión, los objetos RoiID, TObj y CObj fueron registrados por el interceptor servidor en el nuevo servicio de nominación, dándoles un nombre único que puede construirse concatenando al nombre del hilo de ejecución utilizado un sufijo formado por el PID de la máquina virtual Java y el nombre de la clase de objeto.

   El interceptor cliente utiliza ese mismo nombre para obtener las referencias que incluirá en el mensaje a enviar a los cohortes.

Una vez los cohortes reciben este mensaje, sus interceptores servidores recogen las referencias transmitidas y crean a partir de ellas las correspondientes réplicas de los objetos RoiID, TObj y CObj.

4. *Envío del último checkpoint*. Para enviar el último checkpoint, las labores a realizar en el coordinador coinciden con las del punto anterior. Sin embargo, en el interceptor servidor de los dominios cliente deben tomarse las siguientes acciones adicionales:

   - Guardar una copia de los resultados transmitidos en este último *checkpoint*. Esta copia se utiliza para contestar a los reintentos de la invocación que se producen en caso de fallo de la réplica coordinadora.
   - Liberar las referencias clientes mantenidas en las réplicas de los objetos CObj y TObj. Cuando todas las referencias clientes han sido liberadas, se entrega la notificación de no-referencia a estos objetos y se toman las medidas oportunas.

5. *Finalización de la invocación en el dominio coordinador*. En este caso el interceptor servidor asociado a esta réplica debe guardar también una copia de los resultados (por si falla la réplica cliente que inició la invocación) y liberará las referencias cliente de los CObj y TObj.

6. *El cliente recibe el resultado de la invocación*. Su interceptor cliente libera la referencia al CObj.

7. *El objeto TObj recibe la notificación de no-referencia*. El serializador da paso a la siguiente invocación en conflicto. En todos los dominios se elimina el objeto TObj.

8. *El objeto CObj recibe la notificación de no-referencia*. Todos los dominios con una réplica de este objeto liberan su referencia cliente al RoiID, destruyen el objeto CObj y descartan la copia de los resultados que habían mantenido hasta el momento.

9. *El objeto RoiID recibe la notificación de no-referencia*. El objeto RoiID se destruye en el interceptor cliente del dominio que inició la invocación. Con esto se da por terminada definitivamente la invocación.

Los puntos anteriores describen el funcionamiento del protocolo en ausencia de fallos. En caso de que alguno de los participantes en el protocolo llegase a fallar, la evolución del protocolo puede encontrarse en [5]. Cuando se usan interceptores, apenas habrá diferencias. La ausencia de un protocolo de cuenta de referencias integrado en el ORB hará más lenta la detección de algunos casos de fallo y provocará que, en caso de particiones, el sistema no se comporte correctamente (En Hidra, estas situaciones son controladas por el ORB).

Sin embargo, si no se dan particiones, el protocolo funcionará correctamente. Si se desea tener una rápida detección de los fallos para reajustar inmediatamente los valores de las cuentas, la solución descrita previamente necesitará un elevado número de mensajes, ya que las invocaciones para refresco de referencias deberán ser muy frecuentes. Además, estas invocaciones dependen del número de clientes para este tipo de objetos replicados (en nuestro caso, los propios objetos auxiliares RoiID, CObj y TObj), por lo que si existen muchos objetos replicados con un elevado número de invocaciones entre ellos, la carga introducida por el protocolo de cuenta de referencias puede ser excesivamente alta. Como conclusión, para no necesitar un elevado número de invocaciones de refresco de referencias, se deberá incrementar el periodo de refresco con lo que se incrementará el tiempo necesario para detectar estas situaciones de fallo.

6 Conclusiones

En este artículo se ha presentado el soporte implementado mediante interceptores CORBA para objetos replicados bajo el modelo coordinador-cohorte. Se ha tomado como soporte original el diseñado en el sistema HIDRA y que actualmente se está implementando.

El soporte presentado se ha utilizado para desarrollar una versión inicial del protocolo de invocaciones fiables a objeto, cuya implementación también ha sido descrita. Para completar este diseño será necesario terminar la implementación del servicio de nominación para objetos replicados, así como los interceptores de enlace necesarios para obtener nuevas referencias clientes en caso de fallo de alguna réplica.

El uso de interceptores plantea algunas limitaciones importantes cuando se compara con el modelo de HIDRA. La ausencia de un monitor de pertenencia a grupo y la consiguiente ausencia de un protocolo de cuenta de referencias integrado en el ORB, complican el desarrollo del soporte necesario. La simulación de la cuenta de referencias en una clase de la que heredarán los objetos que la necesiten no siempre será aceptable, ya que introduce una sobrecarga importante en el número de mensajes necesarios para mantener esta cuenta y los resultados obtenidos no son correctos en casos de fallos múltiples y presencia de particiones en la red.

Sin embargo, esta solución puede utilizarse en entornos en los que haya baja probabilidad de múltiples fallos y en los que el número de objetos replicados no sea excesivamente alto.

Referencias

[1] K. P. Birman, T. Joseph, T. Raeuchle, and A. El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Trans. on SW Eng.*, 11(6):502–508, June 1985.

[2] N. Budhiraja, K. Marzullo, F. B. Schneider, and S. Toueg. The primary-backup approach. In S. J. Mullender, editor, *Distributed Systems (2nd ed.)*, pages 199–216. Addison-Wesley, Wokingham, UK, 1993.

[3] P. Galdámez, F. D. Muñoz-Escoí, and J. M. Bernabéu-Aubán. High availability support in CORBA environments. In F. Plášil and K. G. Jeffery, editors, *24th Seminar on Current Trends in Theory and Practice of Informatics, Milovy, Czech Republic*, volume 1338 of *LNCS*, pages 407–414. Springer Verlag, November 1997.

[4] F. D. Muñoz-Escoí, P. Galdámez, and J. M. Bernabéu-Aubán. HCC: A concurrency control mechanism for replicated objects. In *Actas de las VI Jornadas de Concurrencia, Pamplona, España*, pages 189–204, July 1998.

[5] F. D. Muñoz-Escoí, P. Galdámez, and J. M. Bernabéu-Aubán. ROI: An invocation mechanism for replicated objects. In *Proc. of the 17th IEEE Symposium on Reliable Distributed Systems, West Lafayette, IN, USA*, pages 29–35, October 1998.

[6] OMG. *CORBAservices: Common Object Services Specification*. Object Management Group, November 1997.

[7] OMG. *The Common Object Request Broker: Architecture and Specification*. Object Management Group, February 1998. Revision 2.2.

[8] F. B. Schneider. Replication management using the state-machine approach. In S. J. Mullender, editor, *Distributed Systems (2nd ed.)*, pages 166–197. Addison-Wesley, Wokingham, UK, 1993.

[9] Visigenic Software. *Visibroker for Java v3.2 Programmer's Guide*. Visigenic Software, Inc., San Mateo, CA, USA, February 1998.