

ORBit2 como middleware para una aplicación multicapa, usando el servicio de nombres CORBA

Alejandro García Castro, Juan José Sánchez Penas

Igalia Gutenberg, 34B 2º, Polígono de A Grela – 15008 A Coruña
acaastro@igalia.com, juanjo@dc.fi.udc.es

Resumen ORBit2 es el ORB utilizado en el proyecto GNOME para la comunicación de componentes en el escritorio; es la tecnología que integra las aplicaciones y les permite colaborar. **Fisterra** es un proyecto de *software libre* cuyo objetivo es la creación de una plataforma para el desarrollo de aplicaciones de gestión empresarial con tecnología GNOME. Nuestro interés por el proyecto GNOME, tanto en su vertiente tecnológica como de comunidad, para el desarrollo de soluciones, nos ha llevado a la selección de ORBit2 para la integración no únicamente del interfaz, sino también de los componentes no visuales del proyecto, planteando su uso como **middleware** de comunicaciones dentro de la arquitectura multicapa de **Fisterra**. En este artículo presentamos nuestra experiencia en el uso de ORBit2 como **middleware** de comunicaciones para la creación de una aplicación cliente/servidor, los problemas que nos hemos encontrado, y las soluciones aplicadas. El principal problema encontrado para cumplir los requisitos impuestos por un entorno como **Fisterra**, fue la ausencia en ORBit2 de un sistema para la referencia de objetos CORBA mediante una referencia fija, que haga más cómodo el uso del servicio de nombres (CORBA *Nameservice*). Con la colaboración de varios *hackers* de ORBit2 creamos un sistema que permite referenciar a un objeto por un nombre fijo y legible. Finalmente, plantearemos nuestras conclusiones tras el desarrollo de esta solución, discutiendo las posibilidades de trabajo futuro.

1. Introducción

La viabilidad de la utilización de *software libre* en el sector empresarial depende muchas veces de la disponibilidad de software de gestión de negocio. En nuestra experiencia profesional esto se ha manifestado como imprescindible para diversas empresas e incluso para instituciones públicas. Una empresa de desarrollo de *software libre* necesita generar una comunidad suficientemente grande de usuarios de las tecnologías con las que trabaja, una comunidad de clientes capaces de mantener un volumen adecuado de financiación de nuevos desarrollos. Para conseguir esta comunidad, es necesario plantear la creación de un entorno de desarrollo de software de gestión abierto e integrado con tecnologías de escritorio, que permita desarrollar este tipo de aplicaciones y mantener un repositorio de soluciones reutilizables. Por estas razones, confiando en GNOME como opción

más interesante para el escritorio, se crea el proyecto *Fisterra*¹[1,2,3] en mayo de 2003. Dentro del proyecto, la idea de crear aplicaciones de gestión usando interfaces *GNOME* evolucionó de una propuesta inicial menos ambiciosa hasta una arquitectura compleja, capaz de soportar aplicaciones muy diversas creadas por una comunidad de desarrolladores grande con requisitos heterogéneos.

La arquitectura propuesta para *Fisterra* contempla la existencia de múltiples capas, algo básico para el desarrollo de aplicaciones de este tipo, tal y como propone Martin Fowler[4]. Nuestra propuesta basa la arquitectura completa del proyecto en tecnologías *GNOME*, es decir, desde las capas de objetos de servicios, que no son visuales, hasta los interfaces y la integración con el escritorio. Las herramientas de *GNOME* eran suficientes para permitir el desarrollo, la única particularidad era realizar un uso diferente de alguna de sus partes. Se asumió el posible coste de adaptación de estas tecnologías suponiendo que las ventajas de integración a la hora de desarrollar serían mayores, habiendo antes evaluado que las dificultades de crear esta plataforma sobre esta tecnología no eran grandes.

Una de las mayores ventajas de la tecnología *GNOME* de cara a afrontar nuestra propuesta era la completa integración con un *middleware* de comunicaciones *CORBA* [5], que incluía los servicios básicos y daba una implementación del estándar con un rendimiento muy elevado: *ORBit2*. *ORBit2* se utiliza principalmente en *GNOME* para la gestión de componentes², siendo la tecnología que se encarga de la integración de interfaces. En nuestro caso adaptamos este comportamiento para el acceso a objetos remotos, lo que mezclado con que *glib* permite el desarrollo de objetos sin estar atado a características de interfaz, nos permitió crear un diseño de arquitectura similar a la descrita en el libro de M. Fowler, usando *GNOME*. *Fisterra* en la actualidad incluye una serie de bibliotecas de soporte, implementadas sobre *GNOME*, que ofrecen las funcionalidades necesarias para estos objetos, como la de persistencia, los llamados *barnacles*[6].

A la hora de usar *ORBit2*, tuvimos que afrontar una serie de problemas debido a las limitaciones causadas por el desarrollo orientado únicamente a la gestión de componentes de escritorio con el que ha crecido este *ORB* (*Object Request Broker*). La principal necesidad era la utilización de *corbalocs* fijos, para permitir que los clientes se pudiesen configurar indicando la URL del objeto servidor de nombres sin tener que cogerla de un servicio externo, lo que añadiría una complejidad innecesaria al sistema. Para ello se implementó la posibilidad de registro de ese tipo de nombres en el *ORB*, mediante el uso de un mensaje de redirección de localización de *CORBA*.

En este artículo presentaremos la arquitectura propuesta usando *ORBit2* en *Fisterra* y su funcionamiento, explicando las necesidades básicas con respecto al *middleware* de comunicaciones, y expondremos las modificaciones realizadas en *ORBit2* para lograr solucionar nuestras necesidades. Además, expondremos nuestras perspectivas de futuro en el trabajo con *ORBit2*.

¹ <http://www.fisterra.org>

² <http://www.djcbssoftware.nl/projecten/nluug-bonobo/>

2. CORBA como middleware de comunicaciones

CORBA es un estándar que permite la integración de componentes software, fue creado por la *OMG(Object Management Group)*³ en 1989 y su uso es muy común en la creación de grandes sistemas.

El estándar define un sistema para que los objetos puedan enviar y recibir peticiones de otros objetos en un entorno distribuido de forma transparente. En este sentido, CORBA es un **middleware** de comunicaciones que define un protocolo utilizado para el intercambio de los mensajes entre los objetos, y que es totalmente transparente para los desarrolladores. La base de esta comunicación es el **ORB**, esta capa es la responsable de encapsular las peticiones y enviarlas al objeto que implementa el servicio deseado. Es decir, en un entorno de funcionamiento normal de CORBA, un objeto encapsula los servicios que suministra un software y que se hacen accesibles para otros objetos CORBA a través de la interfaz que publica el ORB.

Sobre el **ORB** se definen e implementan objetos para el desarrollo de software, estos objetos pueden crearse utilizando diferentes lenguajes de programación. Para la definición del interfaz de estos objetos se usa un lenguaje especial denominado *IDL(Interface Definition Language)*, y se usa además un compilador para ese lenguaje que genera el código necesario para implementar cómodamente los objetos.

ORBit2 es una implementación de CORBA (versión 2.4), que incluye un **ORB**, un compilador de interfaces (genera código para la implementación de los servicios a partir de un lenguaje de definición de interfaces) y un servicio de nombres (permite localizar objetos usando un sistema sencillo de nombrado).

3. Arquitectura multicapa, el proyecto Fistera

Como ya hemos indicado, las propuestas actuales para la solución de aplicaciones de gestión se basan en diseños de arquitectura con múltiples capas. La solución más habitual para sistemas de mediana entidad es la de tres capas. En esta propuesta se separan los interfaces para los clientes de la lógica de negocio y de la base de datos. Las implementaciones de estas arquitecturas en la actualidad utilizan un **middleware** para la comunicación entre los componentes que las forman. Estos componentes se apoyan en un sistema contenedor para poder ejecutarse y proveer los servicios que implementan. El contenedor es una aplicación que da facilidades a los objetos que se ejecutan sobre ella, suele incluir soporte para comunicaciones, gestión de información, persistencia, transacciones, etc.

Fistera es un proyecto diseñado según los patrones marcados por este tipo de arquitecturas, e implementado usando las tecnologías **GNOME**. Se ha intentado evitar usar tecnologías externas a **GNOME** para conseguir un desarrollo totalmente integrado.

Las partes principales de la implementación son:

³ <http://www.omg.org/>

- Cliente: es una aplicación de ventanas, desarrollada como cualquier otra aplicación de escritorio GNOME, que se encarga de interactuar con el usuario. Tiene un diseño interno basado principalmente en una estructura MVC, que dispone de una serie de utilidades para facilitar la implementación. En cuanto a las comunicaciones, debe ser capaz de buscar la localización de los componentes que implementan los servicios de Fistera. El cliente usará esta localización para hacer las llamadas a los servicios implementados en los componentes a través del *middleware* de comunicación. La sintaxis de la localización es dependiente de la tecnología utilizada por dicho *middleware*. En la arquitectura de tres capas el cliente se considera como la primera capa.
- Servidor: está formado por componentes que implementan las acciones necesarias para llevar a cabo los servicios del interfaz. Estos componentes son objetos del tipo EGB (*Enterprise Gnome Beans*), se ejecutan sobre un sistema contenedor y se diseñan para el aprovechamiento de la lógica de negocio. En cuanto a comunicaciones, deben registrarse o fijar su localización en la red para darse a conocer a los clientes. En la arquitectura de tres capas, el servidor se considera la segunda capa.

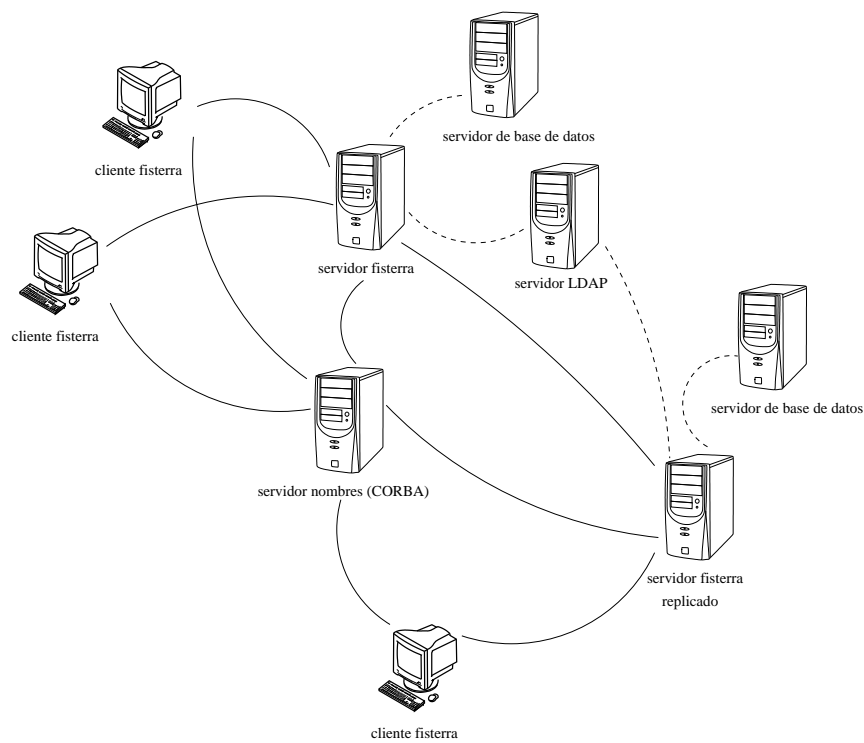


Figura 1. Ejemplo de despliegue de Fistera

Como podemos ver en el ejemplo de despliegue de *Fisterra* en la **Figura 1**, el sistema se comunica con otros servidores, que le proveen de servicios necesarios para su funcionamiento:

- Servidor de base de datos: se encarga del almacenamiento de la información. La interacción con este servidor no está basada en el uso del *middleware* de comunicaciones. Se implementa utilizando *libgda*⁴ que implementa el acceso usando las bibliotecas específicas para cada base de datos. En las arquitecturas de tres capas este servidor es el que se considera como tercera capa.
- Servidor de nombres: implementa el servicio que permite el registro de los componentes para que sean accesibles por los clientes. Los clientes deben conocer su localización para poder después pedir los objetos que necesitan. En nuestro caso este servidor es un objeto *CORBA*. Explicaremos en la siguiente sección en detalle cómo se produce esta comunicación y qué requisitos le impone nuestro sistema.
- Servidor LDAP: almacena una parte de los datos necesarios para el funcionamiento de la aplicación, la información de autenticación. Podría usarse cualquier otro tipo de sistema diferente para esta tarea ya que *Fisterra* utiliza *PAM*⁵ para realizar esta labor. Hemos puesto este servidor en el gráfico de ejemplo por considerarlo la opción más habitual a la hora de realizar un despliegue. La comunicación con este servidor se realiza utilizando el protocolo de LDAP

En el ejemplo **Figura 1** podemos ver las diferentes máquinas que podrían formar parte de un despliegue en un entorno real y las comunicaciones que establecen. Como particularidad interesante podemos ver en la figura el despliegue de un servidor de *Fisterra* replicado, con su propia base de datos en la que se replican los objetos seleccionados del servidor principal. Se permite la modificación de objetos en ambas partes, y se resuelven conflictos siguiendo criterios de temporalidad y versiones dentro del sistema. Para el tema de este artículo la parte relevante es que esos servidores se conocen unos a otros mediante el registro en el servidor de nombres. Los componentes encargados de realizar la operación se encargan de buscar los servicios necesarios en el servidor remoto y de interactuar para realizar la replicación de los objetos.

4. **ORBit2** como *middleware* de comunicaciones, el servicio de nombres

En la presentación acerca de *ORBit2* de Frank Rehberger [7] en la *GUADec*⁶ del año 2004, se propone su uso como *middleware* de comunicaciones más allá de los componentes más visuales del escritorio. En nuestro caso la propuesta particular para *Fisterra* fue el uso de *ORBit2* como *middleware* para la implementación

⁴ <http://www.gnome-db.org/>

⁵ *Pluggable Authentication Modules*

⁶ <http://2004.guadec.org/>

de las comunicaciones con la capa intermedia de una arquitectura multicapa. La principal razón para esta selección fue la integración total con el entorno GNOME, que nos permitía evitar problemas de compatibilidades tecnológicas, y trabajar con el mismo entorno para todo el desarrollo, dándonos las ventajas derivadas de ser el *middleware* de comunicaciones incrustado completamente con el escritorio. Para este tipo de aplicaciones y la integración de las mismas con GNOME, que se use un *middleware* CORBA como ORBit2 es una ventaja, ya que permite una integración muy grande y todavía sin explorar completamente con el escritorio.

Más concretamente las comunicaciones entre las partes implementadas de *Fisterra* se realizan utilizando unas capas que encapsulan todo el código dependiente de CORBA. Este código interactúa con ORBit2 y se encarga de realizar las transformaciones necesarias de datos para poder pasárselos al ORB.

En la parte cliente se usa esta capa delegada, que se inicializa cuando se arranca la aplicación. Para inicializarse lo que necesita es conseguir un objeto sesión que identifique al cliente y dé permisos ante el servidor *Fisterra*. Estos objetos sesión los asigna un componente específico que se encarga de comprobar si el usuario es quien dice ser, de crear el objeto y de devolver al cliente la referencia, una fábrica. Una vez que disponemos de este objeto sesión tenemos que pasárselo al componente de servicios generales de *Fisterra* para que lo inicialice con las ACLs⁷ de nuestro usuario en el sistema. Cuando este objeto sesión está inicializado con esta información el cliente guarda la referencia y da por finalizada la inicialización de la capa. Como podemos observar, antes de realizar cualquier comunicación con los componentes que hemos descrito, el cliente debe conocer su localización.

En la parte servidora los componentes cuando se cargan en el ORB se registran en el servidor de nombres para darse a conocer a los clientes. Por un lado se carga el servicio de sesiones, que se encarga de la gestión de los objetos sesión de *Fisterra*, y por otro el componente que representa los servicios de negocios.

Como podemos observar en la *Figura 2* la secuencia de mensajes es la siguiente:

1. Registro del objeto servidor de sesiones en el servidor de nombres CORBA.
2. Registro del objeto servidor de *Fisterra* en el servidor de nombres CORBA.
3. Resolución del nombre del servidor de sesiones en el servidor de nombres.
4. Resolución del nombre del servidor de *Fisterra* en el servidor de nombres.
5. Petición de una sesión al servidor de sesiones. Este servidor es una fábrica (*factory*) de objetos, que se encarga de crearlos y devuelve la referencia. Además tiene la responsabilidad de destruirlas en el caso de que se dejen de utilizar.
6. Envío de la referencia a la sesión al servidor de *Fisterra* para inicializar la lista de ACLs.
7. Acceso a la base de datos para recuperar las ACLs.
8. Envío de las ACLs al objeto sesión para dejarlas registradas durante el tiempo que estemos conectados.

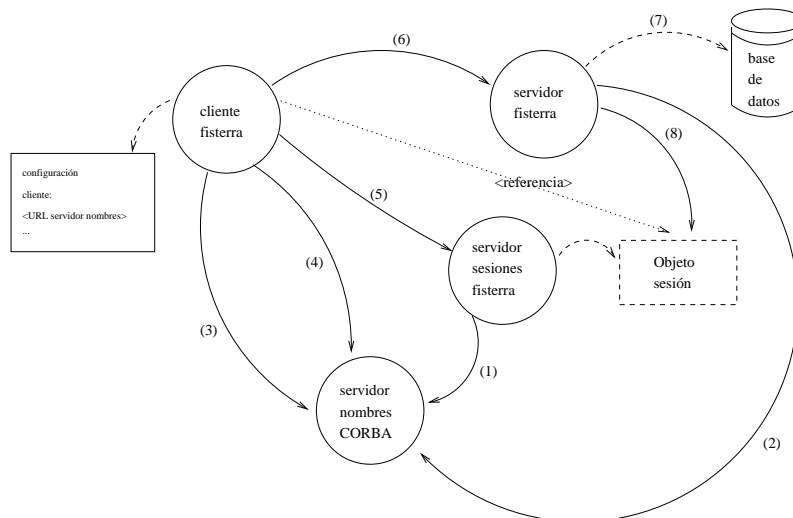


Figura 2. Comunicaciones dentro de Fistera

Para conocer las localizaciones de objetos CORBA se utilizan los IORs⁸. Un IOR es una referencia a un objeto CORBA, que identifica el protocolo utilizado para comunicarse e información relevante para usar ese protocolo. En el caso de usar como protocolo IIOP⁹, la información incluye el nombre de la máquina, el número de puerto TCP/IP y la *object key*. La *object key* identifica a un objeto de forma unívoca dentro de un ORB. Para la identificación del objeto se incluye información del adaptador (POA¹⁰) que está ejecutando ese objeto dentro del ORB y de su nombre dentro de ese adaptador.

Cuando es necesario tratar con los IORs se pueden convertir en cadenas de texto mediante funciones del ORB. Estas cadenas son de la forma:

IOR:01000002100000049444c3a426f6e6f62 ...

Como estas cadenas son engorrosas de manejar, ya que no son legibles, se utiliza otra forma de representación en la que se intenta convertir en caracteres las partes de la cadena hexadecimal del IOR que tienen significado, dando lugar a algo como:

corbaloc:iiop:1.2@myurl.com:1234/01000000937b18f ...

En esta segunda forma podemos identificar a simple vista: el nombre del protocolo usado, la versión, el nombre de la máquina y el puerto. Aun así tenemos

⁷ Access Control Lists
⁸ Interoperable Object References
⁹ Internet Inter-ORB Protocol
¹⁰ Portable Object Adapter

el *corba key* al final que sigue siendo incomprensible y que, además, como incluye la información del adaptador (POA) en el que se ejecuta el objeto, varía cada vez que arrancamos el ORB. Eso implica que si queremos identificar un objeto cada vez que se vuelve a arrancar el ORB en el que se ejecuta, tenemos que recargar el IOR. Aunque la mayoría de ORBs actuales incluyen una forma de poder sustituir el *corba key* por una cadena fija, ORBit2 no tiene esa posibilidad. Por lo tanto, la única forma de poder indicar una referencia fija para el objeto sería tener un servicio externo accesible con un localizador fijo en el que se guardase la referencia al objeto, por ejemplo HTTP. De esa forma, el cliente podría bajar el IOR, que se guardaría cada vez que se arranca el ORB en la *URL*:

```
http://myurl.com/fisterra.ior
```

Como hemos indicado, en Fistera hemos planteado el uso de un servidor de nombres CORBA, este servidor traduce nombres fijos de objetos, referenciados mediante una estructura de directorios, del tipo: *myorb/mysite/fisterra.service*. Es decir, que podemos localizar cualquier objeto dentro del servidor de nombres, con una referencia fija, todos excepto la referencia al propio servidor de nombres. Para solucionar este problema es para lo que hemos añadido a ORBit2 un sistema para poder registrar y acceder a objetos que nos permita sustituir el *corba key* por un nombre fijo, como por ejemplo:

```
corbaloc:iiop:1.2@myurl.com:1234/NameService
```

Como podemos observar en la Figura 2 la localización del servidor de nombres la obtiene el cliente de un parámetro de configuración que indicará un localizador fijo como el anterior. Gracias a nuestra modificación de ORBit2 podremos indicar ese nombre fijo en la configuración y usarlo de forma transparente como un IOR CORBA.

En la siguiente sección explicaremos en qué consiste de forma más detallada la modificación y a qué ha afectado. Además, señalaremos los cambios más importantes que se han realizado en el código de ORBit2.

5. Implementación de localizadores fijos en ORBit2

El diseño de la propuesta para la inclusión de esta característica dentro de ORBit2 fue realizada por Frank Rehberger. La solución se basa en el uso del mensaje *LocateResponse* del protocolo GIOP¹¹. Este mensaje se usa como respuesta al mensaje *LocateRequest* para indicar la localización de un objeto, proporcionando su IOR. Siguiendo el estándar, el cliente CORBA realiza automáticamente la petición a la nueva localización que se le indica con el mensaje *LocateResponse*.

Gracias a este comportamiento, cuando el cliente hace una petición utilizando los localizadores fijos explicados en el apartado anterior, podemos responder desde el servidor con un mensaje *LocateResponse* en el que indicamos el IOR

¹¹ *General Inter ORB Protocol*

a utilizar. Este nuevo IOR se crea tras transformar el nombre recibido en el localizador fijo por el identificador correspondiente.

El cliente realiza de nuevo la petición al IOR que le enviamos de vuelta y, de forma transparente, se lleva a cabo la petición al objeto. Como podemos observar, al meter esta redirección estamos metiendo un retraso en la petición al tener que enviar un par de mensajes extra, con lo que deberíamos usar este tipo de localizadores solo para servicios puntuales, como por ejemplo el servicio de nombres de CORBA.

Para implementar el registro de los nombres debemos introducir una nueva estructura en el ORB, en la que especificamos la relación entre un nombre fijo y el IOR del objeto que deseamos registrar. Usamos una tabla *hash*, utilizamos las cadenas que representan los objetos como claves y para indexar las cadenas de los IOR que se corresponden con ellas.

```
struct CORBA_ORB_type {
    struct ORBit_RootObject_struct  root_object;

    ...

    GSList                          *servers;
    GSList                          *profiles;
    GPtrArray                       *adaptors;

    /* Esta es la tabla hash que utilizaremos */
    GHashTable                      *forw_binds;

    GSList                          *current_invocations;
    gpointer                        default_ctx;
    GHashTable                      *initial_refs;
    guint                           life_flags;

    ...
};
```

Por otro lado, añadimos el registro de los objetos y la comprobación de la tabla *hash*. La búsqueda en esta tabla se realiza únicamente en el caso en el que no se haya encontrado la clave del objeto en el sistema, con lo que le no se añade ningún retraso a las peticiones normales de ORBit2.

```
...

if (!adaptor || !objkey) {
    CORBA_Object forw_obj = ORBit_forw_bind_find (orb, objkey);
    if (forw_obj) {
        GIOPSendBuffer *send_buffer =
```

```

        giop_send_buffer_use_reply
        (recv_buffer->giop_version,
         giop_recv_buffer_get_request_id (recv_buffer),
         GIOP_LOCATION_FORWARD);

        ORBit_marshal_object(send_buffer, forw_obj);

        giop_send_buffer_write (send_buffer,
                                recv_buffer->connection,
                                FALSE);
        giop_send_buffer_unuse (send_buffer);

        giop_recv_buffer_unuse (recv_buffer);
    }
else {
...

```

Como podemos observar en este trozo de código, una vez que la referencia al objeto no se encuentra entre las *object keys* del ORB, lo que se hace es buscar primero en la tabla que hemos creado, si localizamos un objeto al que redirigir la petición entonces creamos el mensaje `LocateResponse` con la cadena del IOR del objeto representado por la petición, y en caso contrario se devuelve un error.

Con estas modificaciones, el servidor de nombres, que ya realizaba la llamada a la función `ORBit_ORB_forw_bind`, se registra en la nueva tabla *hash* con un nombre que se le pasa en la línea de comando usando el parámetro: `-key`. Antes de introducir nuestro cambio en `ORBit2`, esta función devolvía un mensaje diciendo que todavía no estaba implementada.

```

$ orbit-name-server-2 --help
Usage: orbit-name-server-2 [OPTION...]
  --key=string      Respond to corbaloc requests
                   with this object key

Help options:
  -?, --help       Show this help message
  --usage          Display brief usage message

```

6. Ejemplos de uso

Expondremos en este punto la forma más sencilla de probar el funcionamiento del ORB con estas modificaciones. Una vez parcheado he instalado el código debemos ejecutar el servidor de nombres.

```

$ orbit-name-server-2 --key=NameService --ORBIIOPv4=1

```

```
-ORBIIOPIPName=localhost -ORBIIOPIPSock=1234
-ORBIIOPUNIX=0 -ORBCorbaloc=1
```

Después podemos ejecutar el ejemplo de resolución de nombres que podemos encontrar en el *CVS* de ORBit2 para comprobar que el servidor se registra correctamente y el cliente consigue localizarlo.

```
$ name-resolve-server
    -ORBNamingIOR=corbaloc:iiop:1.2@localhost:4444/NameService
$ name-resolve-client
    -ORBNamingIOR=corbaloc:iiop:1.2@localhost:4444/NameService
```

7. Conclusiones y trabajo futuro

Las modificaciones descritas en este artículo llevan utilizándose ya cierto tiempo dentro del proyecto *Fisterra* para la localización de los componentes. El funcionamiento de ORBit2 como *middleware* de comunicaciones para este tipo de arquitecturas nos ha permitido desarrollar *Fisterra* como una aplicación completamente integrada con las tecnologías GNOME y con su escritorio. ORBit2 es una tecnología madura, y una implementación eficiente de un ORB; aun así todavía tiene algunas limitaciones que afectan a sistemas como *Fisterra*. La posibilidad de usar un protocolo seguro como SSL sería algo necesario para poder utilizar el *middleware* en entornos inseguros. Además, pensando también en la seguridad, sería importante revisar los problemas a la hora de realizar registros de nombres en el servidor, y habría que evitar la posibilidad de registrar objetos que falsifiquen a otros.

Las modificaciones realizadas en el ORB nos han permitido utilizar el servicio de nombres de ORBit2 de forma sencilla, simplificando la gestión de comunicaciones del sistema. Además nos han permitido algunas funcionalidades interesantes como la implementación sencilla de la tolerancia a fallos; para ello, el servidor comprueba, en el caso de no poder registrarse en el servidor de nombres, si la excepción devuelta es debida a que el nombre ya está registrado, en cuyo caso se mantiene en un estado de espera para entrar en funcionamiento en el momento en el que el servidor actual deje de funcionar.

El parche realizado fue propuesto a los desarrolladores para que se incluyese en el *CVS* de ORBit2, pero no fue tenido en cuenta hasta el momento; para el proyecto *Fisterra* sería importante introducirlo en el repositorio para evitar tener que mantener el parche por separado. Uno de los problemas en este sentido es que los desarrolladores de GNOME no muestran mucho interés por este tipo de funcionamiento porque no tiene que ver directamente con las necesidades del escritorio, no entrando por lo tanto dentro de los objetivos básicos de ORBit2.

Como trabajo futuro con ORBit2 planteamos la revisión de las propuestas de seguridad para permitir el uso en entornos eminentemente inseguros. Además nos gustaría revisar el código del servidor de nombres para comprobar el código y mejorar sus servicios, así como sus utilidades. Como punto interesante para el trabajo futuro también pensamos en la implementación de nuevos servicios

de CORBA, que permitan realizar búsquedas más complejas de objetos, como el servicio *trader* o la integración con el sistema de activación de GNOME.

Otra posibilidad interesante para el futuro es la comprobación del rendimiento en condiciones de carga elevada. A falta de realizar pruebas detalladas y comparaciones, el rendimiento parece adecuado, al menos en los entornos y condiciones en los que se ha desplegado.

8. Agradecimientos

Los autores del artículo quieren agradecer a Frank Rehberger por su apoyo en el diseño de la modificación de ORBit2. Además queremos agradecer a Javier Vázquez Lamas, Xavier Castaño García, Alberto García González, José Juan González Alonso, Jose Dapena Paz, Jose María Casanova Crespo, Javier Fernández García-Boente, Alejandro Piñeiro Iglesias, Xavier Rodríguez Calvar, Eloy Froufe Pérez y Sergio Villar Senín su colaboración en el desarrollo del proyecto *Fisterra*.

Referencias

1. García-Castro, A., Dapena-Paz, J.: Gnome for business appliances: A case study. Technical report, IV GUADEC, Dublin (2003)
2. Casanova-Crespo, J.M., García-Castro, A.: Hacia *fisterra* 2.0: aplicaciones de empresa para pymes. Technical report, Congreso Hispalinux (2003)
3. Casanova-Crespo, J.M., García-Castro, A., Sánchez-Penas, J.J., Dapena-Paz, J.: *Fisterra* 2: free software for enterprise management. Technical report, Open Source International Conference (OSIC) (2004)
4. Fowler, M.: *Analysis Patterns: Reusable Object Models*. Object Technology Series. Addison-Wesley, Reading, Massachusetts (1997)
5. Henning, M., Vinoski, S.: *Advanced CORBA Programming with C++*. Addison-Wesley (1999)
6. Casanova-Crespo, J.M., García-Castro, A.: *Fisterra* barnacle: Persistencia de objetos en gobject. Technical report, I GUADEC-ES, Almendralejo (2004)
7. Rehberger, F.: Gnome - secure ipc. Technical report, V GUADEC, Kristiansand (2004)