

Fisterra: gestión de empresa con software libre

Javier Fernández García-Boente¹, Alejandro García Castro¹, and Juan José Sánchez Penas

Igalia. Ingeniería en informática y software libre
Gutenberg, 34B 2º, Polígono de A Grela – 15008 A Coruña
acastro,jfernandez@igalia.com, juanjo@dc.fi.udc.es

Resumen Fisterra es un proyecto de Software Libre que tiene como objetivo la creación de una plataforma de desarrollo de aplicaciones de gestión empresarial, y la implementación sobre ésta de soluciones verticales. Fisterra presenta, por lo tanto, una arquitectura orientada a la construcción de este tipo de aplicaciones. En este trabajo describiremos las principales características de la plataforma y las tecnologías, todas ellas Software Libre, que la componen. Presentaremos además nuestras principales experiencias a la hora de aplicar la tecnología seleccionada, basada en GNOME, a este tipo de soluciones.

1. Introducción

El proyecto Fisterra nació en mayo de 2003 con el objetivo de ofrecer una alternativa más completa a las escasas soluciones existentes dentro del software libre para la gestión de negocio.

Las aplicaciones de gestión empresarial suelen ser grandes y complejas, y poseen unas características específicas que las diferencian del resto de tipos de software. Existen diversas iniciativas para intentar desarrollar plataformas y soluciones de gestión de negocio, tanto libres como propietarias, pero desde el proyecto Fisterra se realiza una propuesta diferente, innovadora, y totalmente libre, que utiliza tecnologías GNOME y una arquitectura compleja y flexible.

2. Arquitectura

En esta sección se describen las principales características de la arquitectura Fisterra, entre las que destacan tres aspectos importantes: el uso de una arquitectura de tres capas, con un servidor funcionando como *middleware*; la aplicación del patrón MVC de una forma estricta para el diseño de la interfaz; y la utilización de un contenedor de objetos de negocio, llamados EGBs (la versión que utilizamos en Fisterra de los EJBs).

2.1. Middleware : arquitectura en 3 capas

Fisterra se articula según una arquitectura multicapa:

- Capa de almacenamiento de datos persistente. Consiste en una base de datos relacional (PostgreSQL).
- Servidor de aplicaciones. Consiste en un conjunto de módulos que implementan la lógica de negocio. Se comunican con la base de datos utilizando la biblioteca libGDA. Los objetos de negocio se definen con la tecnología Fistera Barnacle.
- Clientes gráficos. Aplicaciones GTK, siguiendo el patrón modelo-vista-controlador. Se comunican con los componentes de servidor mediante una capa de adaptación CORBA.
- Un servicio de replicación y comunicación entre servidores.

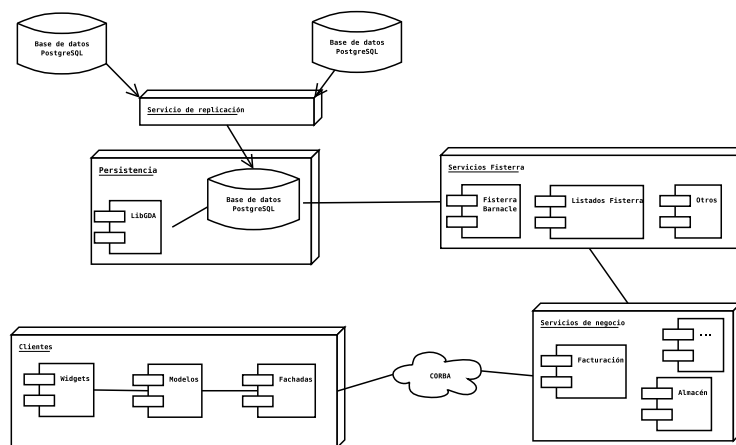


Figura 1. Arquitectura multicapa de Fistera

Servidor de aplicaciones Para conseguir desarrollos eficaces y que puedan ser fácilmente mantenibles, consiguiendo una mayor perdurabilidad es interesante aislar y localizar el comportamiento de gestión de negocio. Por ello en Fistera se ha desarrollado sobre las herramientas GNOME un servidor de aplicaciones. El servidor de aplicaciones da además una serie de servicios de soporte para los objetos que se ejecutan sobre él, que permiten facilitar el desarrollo, servicios como la autenticación y acceso a datos. De esta forma el servidor de aplicaciones proporciona distintos servicios que son utilizados por el cliente para la ejecución de la aplicación.

El sistema servidor permite disponer de un único punto de acceso a la base de datos, de forma que todas las operaciones contra ella se ejecutarán desde un único punto. Esto además elimina la dependencia de la base de datos para la gestión de la concurrencia, tal y como ocurría en versiones anteriores, en las que el diseño no fue tan ambicioso.

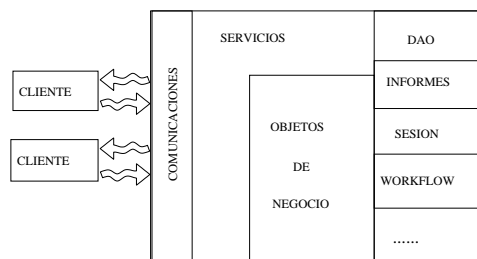


Figura 2. Servidor de aplicaciones

El servidor permite además gestionar la operativa de distintos clientes y disponer de una gestión global de autorizaciones y control de las operaciones.

Podemos describir el servidor explicando los siguientes componentes y capas:

Capa de comunicaciones: Implementa las interfaces disponibles para su utilización desde cliente, se encarga de aceptar las peticiones del cliente y re-dirigirlas al servicio correspondiente. La tecnología de comunicaciones es independiente al sistema, pero para las implementaciones iniciales se utilizó CORBA.

Capa de servicios: Se dividen en 2 tipos, servicios de utilidad y servicios de negocio. Entre los servicios utilidad se encuentran el acceso a datos, gestión de workflow, gestión de sesión y elaboración de informes. Los servicios de negocio (EGBs) implementan comportamientos del negocio soportado por la aplicación y se apoya sobre los servicios utilidad, de esta forma las personalizaciones del modelo de negocio consistirían en modificar o parametrizar estos servicios.

En resumen el servidor de aplicaciones de Fistera es un conjunto de componentes comunicados entre sí, que se encargan de gestionar la lógica de negocio, el almacenamiento persistente, el control de acceso, y ciertas comunicaciones con otros sistemas Legacy.

Algunos de los principales servicios de utilidad de Fistera son:

Fistera Barnacle: Objetos de negocio, con capacidad de almacenamiento persistente.

Sistema de listados e informes: permite la obtención de listados de la base de datos, filtrados u ordenados de una forma determinada.

Servicios de autenticación, autorización y sesión: Se encarga de identificar a los usuarios, y permitirles utilizar y manipular la información adecuada.

Workflow: Módulo que gestiona el ciclo de vida de la información en el sistema, gestionando su manipulación por diferentes personas y procesos.

Planificador de tareas: módulo que permite la ejecución planificada de procesos batch, según una interfaz establecida.

Módulo de Legacy: permite la importación y exportación de datos de la aplicación, para su integración con otros sistemas.

2.2. Diseño MVC

El cliente está construido mediante el patrón MVC (Model View Controller) aplicado sobre todo a los widgets, aunque también a las ventanas que los contienen:

Vista: descrita mediante ficheros XML de glade

Controlador: rellena el widget en base a los datos de su modelo, gestiona los eventos y callbacks.

Modelo: almacena los objetos de datos (*barnacles*) con los que se trabaja en el widget. Es realmente el único objeto que mantiene relación con los objetos de datos que no son accesibles para el resto de objetos del interfaz.

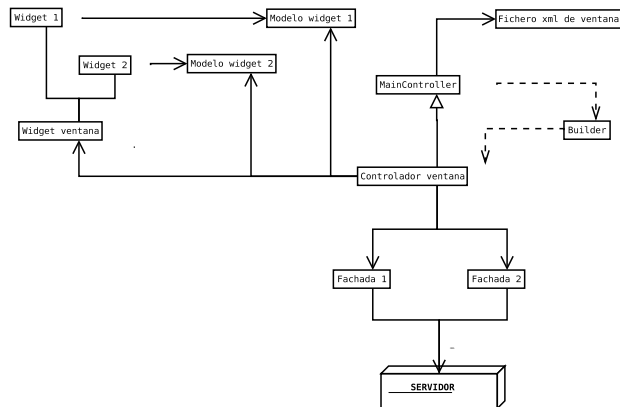


Figura 3. Arquitectura cliente de Fisterra

La vista está compuesta por ventanas en los que se incrustan los widgets, cada uno de los cuáles tiene asociado su modelo.

Una ventana se compone de varios widgets, está gestionada por un controlador que tiene que heredar de un controlador padre (*FControl*). Otro objeto, el *main_controller*, utiliza un fichero XML del que cargará la definición de la ventana que se solicite, usando un objeto *builder* que crea la ventana. Esta factoría es la responsable de la creación de todas las ventanas del sistema, definidas todas en este fichero XML.

El controlador tiene que tener una referencia de cada widget que maneja, para poder ordenarles acciones en función de las necesidades de la ventana. Y será el controlador el que a través de una o varias fachadas (cada una proporciona

un servicio) acceda al servidor mediante CORBA, pero de forma transparente. Es decir, la responsabilidad de conseguir los modelos para los widgets es del controlador de la ventana, y además es el que al final se encarga de pedir los modelos y llamar a las fachadas adecuadas para realizar las operaciones.

Un aspecto importante en las fachadas es que a ellas les llegan modelos de widget que contienen los datos validados y ellas son las encargadas de transformarlos a objetos del dominio, con los que solo se trabaja en el servidor.

2.3. Contenedor de EGBs

Los servicios implementados en las aplicaciones desarrolladas bajo *Fisterra* están encapsulados en componentes denominados *EGBs* (Enterprise Gnome Beans). El servidor de aplicaciones descrito anteriormente es un contenedor de estos componentes y se encarga de su inicialización y liberación cuando estos ya no sean requeridos.

Ya se ha comentado anteriormente que la arquitectura *Fisterra* ha sido diseñada en 3 capas. Gracias al sistema de comunicación distribuida basada en *IIOP*, el *middleware* puede implantarse de forma independiente a las capas de interfaz y datos. Dichas capas pueden residir en máquinas distintas lo que confiere una gran flexibilidad y tolerancia a fallos. La replicación del *middleware* y de la capa de datos permite aplicar técnicas de balance de carga que optimizan el rendimiento de la aplicación.

Los componentes denominados *EGBs* forma el corazón de este *middleware*, manteniendo independiente la lógica de negocio con el acceso a datos y la interfaz. Disponen además de facilidades para el programador, como son la gestión de transacciones y errores. Estos componentes puede clasificarse en 2 grupos:

EGBs de Servicio: Implementan los servicios requeridos por la capa de comunicaciones, no tienen lógica de negocio. Permiten crear servicios que acceden directamente a las bibliotecas como las de autenticación, datos, etc.

EGBs de Negocio: Implementan procesos de negocio propiamente dichos. Garantizan que las operaciones se realizan de forma transaccional, asegurando la recuperación de los datos en caso de error en la operación. En teoría no deberían accederse desde la capa de comunicaciones, debería pasarse siempre por los EGBs de servicio.

3. Tecnología

Fisterra utiliza como base el conjunto de tecnologías que conforman el Gnome SDK, para las interfaces de usuario y el servidor de aplicaciones. Sobre éstas, integra e introduce otras tecnologías, para dar soporte a persistencia, e implementar la comunicación, tal y como hemos podido ver en apartados anteriores. Sobre ellas, *Fisterra* constituye por si mismo un completo **framework de desarrollo de aplicaciones**. A continuación detallamos las principales tecnologías usadas en el proyecto:

GTK Librería que proporciona los widgets y controles.

LibGlade Librería para la descripción visual de interfaces de usuario.

GObject Sistema que permite realizar orientación a objetos en *C*, usado por el proyecto GNOME.

LibXML, libXSLT Manipulación de ficheros XML, y transformaciones XSL.

Orbit Implementación ligera de CORBA, protocolo de invocación remota de métodos, y comunicación de objetos interoperable.

libGDA Librería de acceso a bases de datos del proyecto Gnome.

PostgreSQL Gestor de bases de datos relacional.

3.1. Interfaz de usuario

Fisterra utiliza las bibliotecas *GTK+* y *libglade* para las interfaces de los clientes, *GObject* para la implementación de objetos en *C* y *libXML* para el parseo de datos XML. Todas ellas basadas en GNOME ([?]).

Cada ventana se crea a partir de un fichero *glade* para definir las propiedades de su controlador. Como ya se ha comentado anteriormente, una ventana puede estar formada por uno o varios widgets. La plataforma *Fisterra* proporciona varios widgets predefinidos, entre los que destacan los widgets del sistema de listado ya que proporcionan una gran velocidad de desarrollo para las ventanas de tipo listado. *Fisterra* proporciona otros widget predefinidos, como los de autorización del sistema y el planificador de tareas.

En el fichero *f_client_fisterra.xml* se definen todos los controladores que manejan cada uno de los widgets y ventanas *glade* que se han creado.

Un ejemplo de una entrada de este fichero podría ser el siguiente :

```
controller name="f_controller_window1"
builder="FControlWidgetBuilder"
file="f_window1.glade
widget="window1-widget"
menu="on_window1_activate"
embedded="true"
```

Cada ventana debe tener su propio *builder*, en el que define el método de construcción para la creación de todos los widgets que se han de mostrar y conectar las señales a cada uno de los elementos definidos en el *glade*. El *controller* es el encargado de definir las funciones que se llaman cuando se ejecuta y una señal.

El desarrollo de interfaces mediante *Fisterra* pretende evolucionar hacia la integración con *LibGlade*, definiendo widgets compatibles con las herramientas gráficas proporcionadas por GNOME como *glade* y *glade-2*. Esto permitirá utilizar los widgets de negocios definidos en *Fisterra* para la definición de las ventanas.

3.2. Sistema de comunicaciones

La capa de comunicaciones es la implementación en una tecnología específica, en este caso CORBA, de los interfaces cliente/servidor y del canal de transferencia de información. Se encarga de aceptar las peticiones en el cliente y redirigirlas

al servicio correspondiente. La tecnología de comunicaciones se mantendrá abstracta al sistema, pero para las implementaciones iniciales se utilizará CORBA. Para más detalle a cerca de implementaciones CORBA, puede consultarse el libro citado en la bibliografía a tal efecto ([?]);

El *ORB* que utiliza el proyecto GNOME es ORBit. Está presente desde el principio del proyecto, debido a la necesidad de tener una implementación de los estándares CORBA especialmente eficiente.

GNOME utiliza ORBit como la base para su tecnología de componentes *Bonobo*. Esta se usa para conectar los applets en las barras y paneles del escritorio, conectar las diferentes vistas en el gestor de ficheros *Nautilus*, o el modelo de documentos de *Gnome Office*.

ORBit proporciona interfaces para *C*, *C++*, *Python* y *Perl* directamente y puede comunicarse con otros *ORB*, según los estándares establecidos. En *Fisterra* se utiliza principalmente para la comunicación entre componentes de cliente y servidor, entre los componentes disponibles del lado de servidor.

En la actualidad, el proyecto *fisterra* está intentando integrar *ORBs* implementados en distintos lenguajes, como *java* y *C#* y así diversificar las plataformas en las que corren los clientes. Otras tecnologías de comunicación han cobrado importancia para el desarrollo del proyecto, como *SOAP* u otras implementaciones de CORBA, como *ORBacus* o *Jacorb*.

3.3. Acceso a base de datos

Para el almacenamiento persistente, *Fisterra* utiliza la biblioteca *libGDA* (que aísla el gestor concreto utilizado del resto del sistema), y el gestor de bases de datos relacionales PostgreSQL. Todo el almacenamiento persistente se realiza para almacenar la información contenida en los *Fisterra Barnacle* [?], pensado como un sistema de persistencia de objetos para *GObject*.

Los *Fisterra Barnacle* son contenedores de información de negocio, sin un comportamiento asociado. La información se almacena como un conjunto de atributos (par de nombre y valor), siendo los valores de tipos básicos, almacenables en la base de datos.

Se utiliza *libGDA* para permitir la independencia del gestor de bases de datos, y así poder sustituir con menos esfuerzo *PostgreSQL* por otro gestor en un futuro.

Se utilizan transacciones y restricciones de integridad para asegurar la calidad y fiabilidad del modelo de datos, y diferentes técnicas de optimización de consultas y réplicas de datos, para mejorar el rendimiento y escalabilidad del sistema.

Fisterra Barnacle El estado actual del desarrollo dota a la biblioteca *Fisterra* de unas funcionalidades básicas para la gestión de la persistencia de los objetos. El prototipo actualmente en funcionamiento tiene las siguientes características: primera aproximación a la descripción del modelado de clases; sistema de generación de código fuente; *dataobject*, que son objetos básicos con operaciones *get* y *set* sobre atributos con tipos básicos; métodos generados automáticamente

para el almacenamiento y recuperación de *barnacles* utilizando `libgda`; *mapping* automático de transporte de *barnacles* a través de CORBA basado en Orbit; generación del esquema relacional en base a la descripción de los *barnacle* para PostgreSQL; sistema de identificación única de los objetos; y gestión de versiones de objetos para control de concurrencia.

El modelo de persistencia actual está basado en código generado a partir de la definición del modelo de objetos en XML (describiendo atributos, clases y relaciones). Al procesar este fichero XML con el generador de código, se obtiene una estructura de compilación con los *dataobject*, los *barnacle* DataAccess Objects, un esquema de base de datos descrito en SQL y un *mapping* CORBA para los *dataobject*.

Los objetos poseen un identificador único: el código `barnacle`. Esta clave está formada por tres campos: identificador del *host* donde se instanció por primera vez el objeto, identificador de la clase y código de secuencia del objeto. Estas tres partes de la clave implementadas mediante un único entero de 64 bits, nos aporta las siguientes características: dos objetos creados en máquinas distintas no tendrán nunca el mismo identificador; y el código incluye información sobre la clase de un objeto, lo que facilita su recuperación.

Todo el sistema de *Barnacles* posee un estricto control de versiones. Este control permite que se la ejecución concurrente mantenga la coherencia de los datos. Esta característica es especialmente útil en entornos donde varios procesos trabajan con los mismos datos y estos datos son modificados. En caso de que varios procesos modifiquen el mismo objeto al mismo tiempo, el sistema de control de versiones se dará cuenta de la posible inconsistencia y exigirá que todos los procesos que pretendan modificar el objeto, después de una primera modificación, vuelvan a solicitarlo.

4. Conclusiones

- Fistera es una **plataforma libre** para el desarrollo de aplicaciones de gestión empresarial (tipo *ERP*).
- Muy alta productividad con *C*, gracias a un claro protocolo de uso del lenguaje, la meticulosidad en la comprobación de los problemas de memoria y el uso de **orientación a objetos** mediante *GObject*.
- El modelo de persistencia basado en *Fistera Barnacles* permite abstraer la estructura relacional de la base de datos facilitando la **aproximación objetual**, más fácil de manejar.
- El **control de versiones** garantiza la **concurrencia** de la operaciones de modificación de datos.
- El sistema de **componentes de negocio**, denominados *EGBs*, permite la separación de la lógica de negocio del acceso a datos y la interfaz.
- El patrón *MVC* completa el objetivo de **independencia de implementación** (separación de roles de desarrollo)