(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

Incorporación de componentes de visualización SIG en entornos distribuidos con tecnologías COM/CORBA, aplicación a un sistema de monitorización de flotas

J. A. Bañares, P. Álvarez, R. López y P.R. Muro-Medrano

Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior, Universidad de Zaragoza
María de Luna 3. 50014 Zaragoza

RESUMEN

De todas las capacidades SIG, las funcionalidades de visualización de información geográfica son las más requeridas en el desarrollo de sistemas de información que tratan con datos georeferenciados. Por su parte, las aplicaciones de seguimiento de flotas de vehículos son un ejemplo prototípico en el que, un desarrollo basado en componentes, permite abordar las necesidades de una alta reusabilidad, interoperatividad y escalabilidad del software desarrollado con el objeto de acelerar su proceso de desarrollo, y de evitar soluciones ad hoc. El presente trabajo presenta una aproximación tecnológica para este tipo de sistemas basada en una arquitectura de componentes distribuidos sobre CORBA. Los componentes que requieren la visualización de información geográfica se han desarrollado a partir de MapObjects, una librería de objetos ActiveX. El trabajo ilustra los aspectos tecnológicos de la integración de componentes SIG donde están involucrados distintas plataformas de distribución de objetos, así como adaptaciones requeridas, y posibilitadas por esta tecnología, para poder integrar componentes SIG comerciales.

1 Introducción

Los avances en los sistemas de seguimiento y control han revolucionado las posibilidades de los sistemas de Ayuda a la Explotación (SAE) de flotas de vehículos, posibilitando la recopilación de parámetros reales del funcionamiento de la explotación y ofreciendo información en tiempo real que permite el control de incidencias y una información valiosa a los usuarios de estas aplicaciones. Este avance se debe en gran medida al abaratamiento de los costes de las radiocomunicaciones y a la tecnología GPS [1] junto al avance de las tecnologías de la información.

La infraestructura necesaria para el desarrollo de aplicaciones de seguimiento y control de flotas requiere una serie de servicios básicos comunes (adquisición, análisis, visualización, comunicaciones, etc.) que exigen al equipo de desarrollo conocer e integrar un gran número de tecnologías, como tarjetas con microprocesadores, comunicaciones por radio, comunicaciones en una red de computadores, desarrollo de software de control, etc. Para disminuir el alto coste que suponen las soluciones “ad hoc” y dada la necesidad de integrar distintas tecnologías se ha buscado una alta reutilización, interoperatividad y escalabilidad, para lo cual se ha diseñado un sistema basado en componentes distribuidos que permite un alto grado de intercomunicación entre componentes que ofertan servicios básicos y aplicaciones más finalistas que los utilizan.

Junto con los componentes que resuelven las comunicaciones y la distribución de posiciones GPS, un aspecto fundamental del sistema de localización es la posibilidad de visualizar las localizaciones en tiempo real de los móviles sobre mapas o planos. El módulo de visualización desarrollado se conecta al componente GPS y muestra las posiciones recibidas en un entorno de sistema de información geográfica (SIG). Los componentes de comunicaciones por radio y de adquisición y distribución de posiciones GPS fueron realizadas en CORBA por ser un modelo estable para sistemas orientados a objeto distribuidos que permite abordar la heterogeneidad y el inevitable cambio de las tecnologías[2].

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

Para la implementación del componente de visualización, se optó por el uso de MapObjects, una librería de objetos ActiveX, rápidamente integrable en cualquier lenguaje de programación de propósito general (Visual C++, Visual Basic, ...) y que proporcionan a una aplicación todas las funcionalidades SIG de forma rápida y sencilla.

MapObjects proporciona una potente herramienta para integrar visualización de información geográfica en el desarrollo de aplicaciones. En la concepción de MapObjects, al igual que en todo sistema software, actúan varias fuerzas contrapuestas: por un lado el deseo de dotar al componente de la máxima funcionalidad, que aporte al usuario el mayor valor añadido con el mínimo tiempo de integración posible, y por otro lado la flexibilidad/generalidad, que permita cubrir el amplísimo espectro de necesidades de visualización de datos geográficos en la integración de sistemas de información sin sobrecargar el componente con funcionalidades que sólo una parte del mercado va a emplear. Cuando no consideramos el mercado globalmente sino que nos fijamos en un usuario concreto, el equilibrio entre estas dos fuerzas se desplaza, esto es, según el tipo de aplicaciones que el usuario desarrolle precisará un componente de visualización que presente unas determinadas características. Posiblemente no le importe que su componente de visualización pierda cierta generalidad siempre y cuando gane potencia y velocidad de integración en su campo de trabajo. Esta filosofía nos ha impulsado a elaborar un componente sobre MapObjects, que extiende sus funcionalidades para ajustarse a las necesidades de un amplio abanico de aplicaciones desarrolladas por nuestro grupo de trabajo en los que es preciso un componente de visualización SIG.

Esta filosofía de desarrollo basado en componentes se adopta por el hecho de que las aplicaciones SIG desarrolladas en nuestro grupo integran diversos subsistemas y fuentes de datos, algunos de los cuales han sido desarrollados previamente por nosotros o por agentes externos, siendo necesario que los distintos subsistemas interoperen entre sí. Ejemplos de este tipo de aplicaciones, además de las aplicaciones de seguimiento de móviles en tiempo real, son los sistemas de información en que sólo una parte de los datos tienen carácter geográfico, lo que permite su visualización en mapa o el acceso a la información utilizando fundamentalmente la localización través de la funcionalidad de un SIG. Estos sistemas necesitan un visualizador geográfico que, además de permitir las acciones típicas (apertura de capas, zoom, pan...), aportan capacidades extras como el acceso a fuentes de datos heterogéneas, capacidad de interoperar con aplicaciones externas y soporte a los grandes modelos de procesamiento distribuido. Los siguientes ejemplos ilustran estas necesidades:

1. Se quiere visualizar las rutas previstas para unos móviles, estas rutas son editadas por otro sistema, posiblemente “legacy” y almacenadas como secuencias de puntos en tablas de una base de datos relacional.
2. Se precisa visualizar la posición en tiempo real de un móvil, por motivos de diseño esta localización es accesible como un objeto remoto a través del estándar CORBA.
3. Una parte del sistema esta implementada haciendo uso de una herramienta de gestión de bases de datos relacionales (Access, Paradox o cualquier otra). Estas herramientas no tienen la posibilidad de realizar selecciones siguiendo criterios espaciales. Para realizar estas selecciones se desea invocar al visualizador geográfico, realizar en el mismo las selecciones que precisemos, ya sean con el ratón o con una herramienta de análisis espacial y exportar esta selección nuevamente al gestor de la base de datos relacional mediante COM.

Montar estas funcionalidades sobre MapObjects cada vez que se integra una aplicación es costoso, justificándose el desarrollo de un componente que extienda las capacidades de MapObjects para ajustarse a los nuevos requisitos. Este componente complementa MapObjects con nuevos *Automation Objects*. Estos nuevos objetos son expertos en diferentes tareas como por ejemplo la selección de elementos de una capa, la creación de capas a partir de fuentes de datos externas (ODBC, OpenGis sobre CORBA, etc.), la edición gráfica de *symbols*, *renderers* y consultas, la interacción con aplicaciones externas, etc.

Este trabajo presenta los patrones de diseño y las técnicas utilizadas para el desarrollo de un componente de visualización geográfico a partir de MapObject que permite rápidamente su integración y adaptación para que se ajuste a las fuentes de datos, entorno de aplicaciones y funcionalidades propias del sistema.

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

2 Arquitectura de una aplicación de seguimiento de vehículos

Con objeto de situar con más detalle el contexto de trabajo del que se partió, y mostrar una aplicación prototípica del componente de visualización desarrollado, en esta sección se presenta brevemente OODISMAL (véase la figura 1).

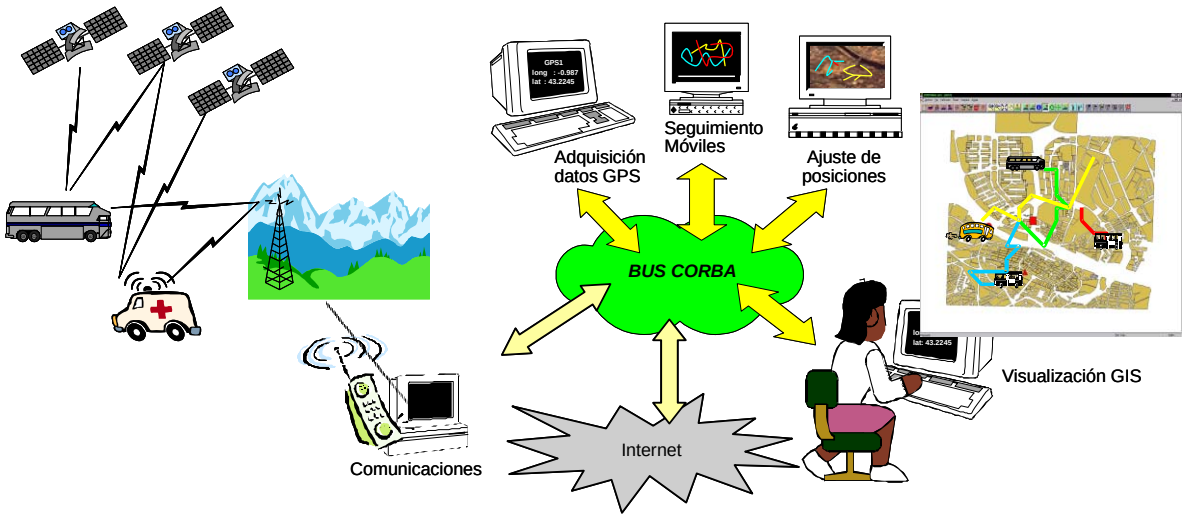

Figura 1. Arquitectura de OODISMAL

OODISMAL (Object Oriented Distributed Information System for Automatic Movil Location) está compuesto por un conjunto de componentes distribuidos en una red de área local, que interoperan entre sí utilizando la infraestructura CORBA[3], y tienen como propósito la adquisición, almacenamiento, procesamiento y visualización en tiempo real en mapas digitales de datos de localización provenientes de móviles que incorporen un dispositivo GPS.

El trabajo de estos componentes está orquestado por la recepción de llamadas de radio con datos GPS, y sus responsabilidades son las siguientes:

- El componente de radio es un servidor CORBA que ofrece al BUS CORBA los mensajes recibidos por la radio conectada a un PC en una red de área local. Este componente, además de hacer disponible las posiciones GPS, ofrece la funcionalidad necesaria para hacer y recibir diferentes clases de llamadas (voz, o datos) a través de la radio. Debido a problemas de cobertura, o a la visualización de flotas que utilizan distintas tecnologías de comunicaciones puede haber distintos componentes de radio, ofreciendo todas ellas un interfaz único independientemente del tipo de tecnología de comunicaciones (GSM, trunking, etc.).
- El componente de adquisición actúa como un cliente del componente de radio para recibir datos de los dispositivos GPS remotos, actuando a su vez como servidor CORBA de datos GPS. La simulación o análisis de una ruta realizada se lleva a cabo a través del componente de simulación que tiene la misma interfaz que el componente de adquisición, recogiendo los datos de la base de datos a través del componente de persistencia. De esta forma los clientes no distinguen entre datos GPS reales o simulados.
- La funcionalidad ofrecida por los anteriores componentes se puede integrar fácilmente en un SIG, de forma que se permita la visualización de la localización de los vehículos en tiempo real, el acceso a la radio de los vehículos visualizados para hacer llamadas de voz, o la visualización de los resultados del análisis de rutas registrados. En la figura 2 se muestra el aspecto de la interfaz de la aplicación.

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

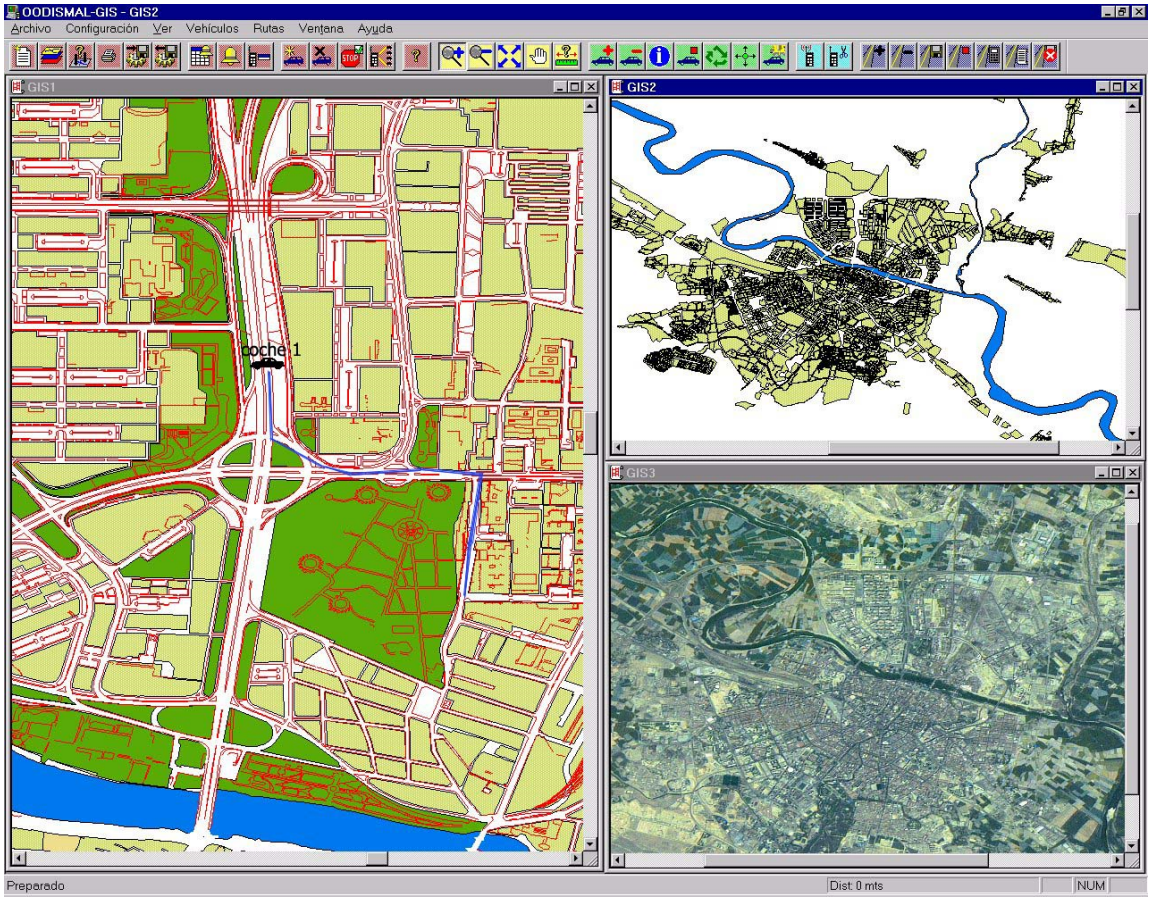

Figura 2. Aspecto de la interfaz de la aplicación de seguimiento de vehículos.

Resumiendo las bondades de la arquitectura basada en componentes distribuidas presentada son las siguientes:

- Flexibilidad: Los componentes pueden ser reemplazados con nuevas versiones sin que se produzca ningún impacto sobre el resto del sistema. Aunque en este trabajo se va a presentar el desarrollo del componente de visualización SIG a partir de MapObject, se ha desarrollado un componente sencilla de visualización SIG completamente en Java que permite la visualización de vehículos a través de Internet, pudiéndose sustituir un componente por otra sin afectar al resto del sistema.
- Reusabilidad: El mismo componente puede ser utilizado para diferentes aplicaciones. Este objetivo es el que ha motivado precisamente el desarrollo de un componente de visualización a partir de la librería de objetos ActiveX de MapObject.
- Robustez: Es posible reducir el riesgo de fallo duplicando componentes en diferentes PCs.
- Escalabilidad: Se pueden solventar problemas en los que ser requiera altas prestaciones en las comunicaciones (debido por ejemplo al volumen de vehículos en la flota), o a la funcionalidad requerida (como por ejemplo integración de diferentes tipos de flotas como en el caso de coordinación en casos de emergencia), simplemente aumentando el número de componentes o puestos e trabajo.

La descripción del componente de visualización SIG es el objeto de este trabajo. Una descripción detallada de los componentes de radio y adquisición puede encontrarse en [4], y distintos casos de uso que muestran la bondad de la arquitectura planteada en [5].

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

3 Funcionalidad necesaria del componente de visualización

Para no perder la capacidad de MapObjects de trabajar en diversos entornos de desarrollo, se debe encapsular todos los objetos nuevos como componentes COM, esta labor conlleva un importante esfuerzo de desarrollo (además bastante engorroso). Sin embargo el beneficio que de ello se obtiene a la hora de acelerar la creación de nuevos sistemas de información compensa el esfuerzo suplementario invertido. Se ha comentado con anterioridad que empleamos CORBA para construir el modelo de objetos de las aplicaciones. La integración del componente de visualización en este modelo se hace mediante el objeto OpenGISRecordset, lo que obliga a dotar a este componente de una doble interface uno COM para interaccionar con el componente de visualizador y otro CORBA para interaccionar con el modelo.

3.1 Representación de la información

En programación orientada a objetos se denomina modelo de la aplicación al conjunto de entidades y relaciones que describen la realidad del problema. Una necesidad recurrente en el desarrollo de una aplicación SIG, es visualizar geográficamente parte del modelo, así por ejemplo, en una aplicación de gestión de una red de autobuses habrá objetos de tipo móvil, ruta, parada, etc. Normalmente alguno o todos los objetos del modelo deben ser representados geográficamente. Por si fuera poco, es más que probable que estos objetos no sean estáticos sino que se “muevan” durante la ejecución del sistema. MapObjects aporta varias facilidades para representar estos objetos: eventos antes y después de dibujar cada capa, la *TrackingLayer* y la posibilidad de pintar un *Shape* directamente en el control *Map*. Dados estos mecanismos no resultaría particularmente difícil implementar de modo “ad-hoc” un mecanismo que representara un modelo concreto en el control *Map*, sin embargo, esta solución resulta engorrosa, sería más interesante contar con una utilidad de carácter general que examinara un modelo, interpretara su estructura y se encargara de un modo automático de su representación.

Si queremos que un mecanismo interprete la estructura de un modelo precisamos, claro está, que el modelo esté autodescrito y eso implica necesariamente la existencia de metainformación [6]. No es necesario reinventar la rueda, ya existe un modelo para entidades geográficas autodescrito, orientado a objetos, potente y flexible: OpenGIS [7], concretamente en este trabajo se emplea una versión reducida de OpenGIS sobre CORBA [8], lo que entre otras cosas permite emplear un modelo distribuido.

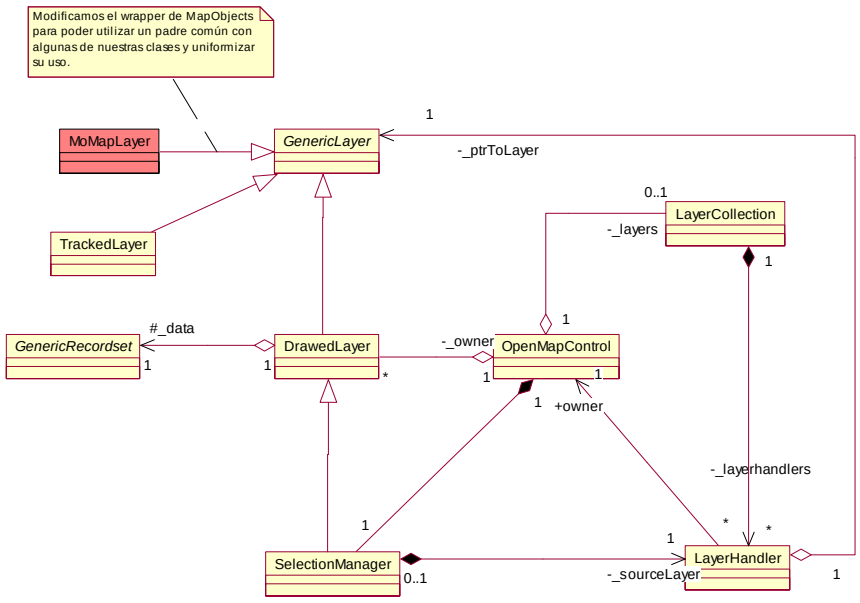

Figura 3. Jerarquía de capas del componente de visualización

Hoy por hoy MapObjects no soporta OpenGIS, parece ser que esta situación se modificará en corto plazo pero empleará la especificación de OpenGIS sobre COM, así que se ha tenido que encarar la construcción del soporte a este estándar de interoperabilidad en SIG. Para ello se ha creado una jerarquía de clases que permita la integración de manera homogénea de los nuevos tipos de datos dentro de MapObjects. Para ello se han añadido dos nuevos tipos de capas a MapObjects: *DrawedLayer* y *TrackedLayer*, (ver figura 3).

Los tipos nuevos de capa (*DrawedLayer* y *TrackedLayer)* hacen esencialmente lo mismo: reciben un objeto con un interface compatible al del *Recordset* de MapObjects (por ejemplo un *OpenGISRecordset* del que hablamos a continuación*)* y representan los registros que contiene. La diferencia entre ambas es la forma de representación, el *DrawedLayer* recorre todos los registros del "RecordSet" y los "pinta" mediante el servicio *DrawShape de MapObjects*. Por otra parte el *TrackedLayer* esta orientado a la representación de eventos dinámicos representándolos mediante su inserción en la *TrackingLayer* de MapObjects, solución más eficiente, pero limitada a elementos de información dinámicos representables mediante puntos.

Las modificaciones realizadas presentan varias dificultades técnicas, por ejemplo: la representación de más de una capa lógica en una física (varias *TrackedLayer* en una *TrakingLayer* para disponer de más de una capa de seguimiento de eventos), empleo de "símbolos" y "*renderers*" en las nuevas capas añadidas, tratamiento homogéneo de los nuevos objetos y los de MapObjects. La mayor parte de las dificultades viene derivadas del hecho que de los objetos COM proporcionados por MapObjects no se puede heredar y en ocasiones ni siquiera pueden ser instanciados directamente, sin embargo, la potencia que nos aporta la nueva jerarquía compensa el esfuerzo que conlleva.

3.2 Acceso a nuevas fuentes de datos

Para facilitar el acceso a fuentes de datos heterogéneas se ha creado una nueva estructura jerárquica de clases mediante la cual se consigue que el acceso a los datos se realice siempre a través de la misma interfaz. Esta interfaz presenta las mismas operaciones que el RecordSet de MapObjects para facilitar la integración de las nuevas fuentes de datos. La especialización en esta jerarquía se realiza en función de las fuentes de datos de forma que cada clase es un especialista en el acceso a un determinado formato de información. Así por ejemplo se pueden crear clases especializadas en el acceso a la información espacial contenida en una base de datos relacional, como es el caso del ODBCRecordset o que faciliten la interoperabilidad con otras aplicaciones, caso del OpenGISRecordset (ver figura 4).

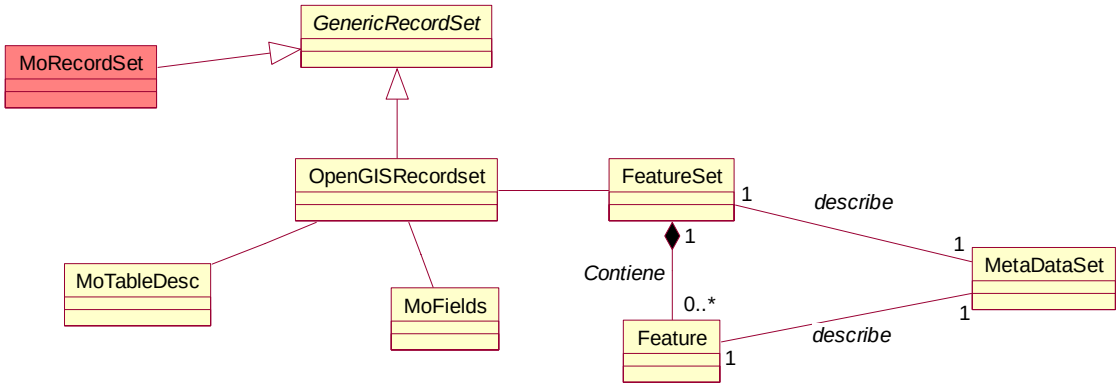

Figura 4. El OpenGISRecordset

La clase OpenGISRecordset se encarga de proporcionar acceso al contenedor de entidades geográficas de OpenGIS denominado *FeatureSet.* El *OpenGISRecordset* se encarga de leer la metainformación que describe las entidades contenidas en el *FeatureSet* y con ellas construye los objetos de tipo *TableDesc*, *Fields* necesarios para trabajar con MapObjects. De esta forma, utilizando la clase OpenGISRecordset, se puede interoperar, por ejemplo, con una aplicación que utilizase un interfaz compatible con el estándar OpenGIS para servir información geográfica.

A la hora de abordar la construcción de un sistema de información que utilice elementos ya preexistentes se podría dar el caso de que la información geográfica fuera almacenada en una base de datos relacional, por ejemplo se podría que tener que representar en una nueva aplicación la

información generada por una aplicación de seguimiento de móviles que almacena las posiciones en unos determinados campos de una tabla de una base de datos como podría ser Oracle. ODBC es una librería de Microsoft que permite el acceso a diferentes Sistemas Gestores de Bases de Datos Relacionales a través de una misma interfaz. Aprovechando esta tecnología se ha desarrollado la clase ODBCRecordset que permite acceder a la información de una base de datos como si fuera un Recordset de MapObjects. Además de contar con los objetos ODBCRecordset para acceder a las bases de datos también se cuenta con la clase DAORecordset que permite acceder a bases de datos de Microsoft Access. Esta última clase tuvo que ser creada debido a varios bugs en los *drivers* de ODBC de algunos gestores de bases de datos (ver figura 5).

También se puede pensar en un "Recordset" que no encapsule ninguna fuente externa sino que el mismo mantenga la información en memoria. Esta entidad, denominada *MemoryRecordset* se muestra tremendamente útil para todos aquellos casos en que el componente de visualización precise mantener internamente un conjunto de elementos geográficos no persistentes. Además, como toda la jerarquía de "RecordSets" comparte una misma interfaz, el objeto *DrawedLayer*, igual puede representar los registros de un *OpenGISRecordset,* de un *ODBCRecordSet* o de un *MemoryRecorset*.

Por supuesto que la "traducción" de información que se produce al emplear esta arquitectura acarrea los problemas característicos de la integración de fuentes de datos heterogéneas: Perdida de información, heterogeneidad sintáctica y semántica, etc., cuya solución queda muy lejos del alcance de este trabajo.

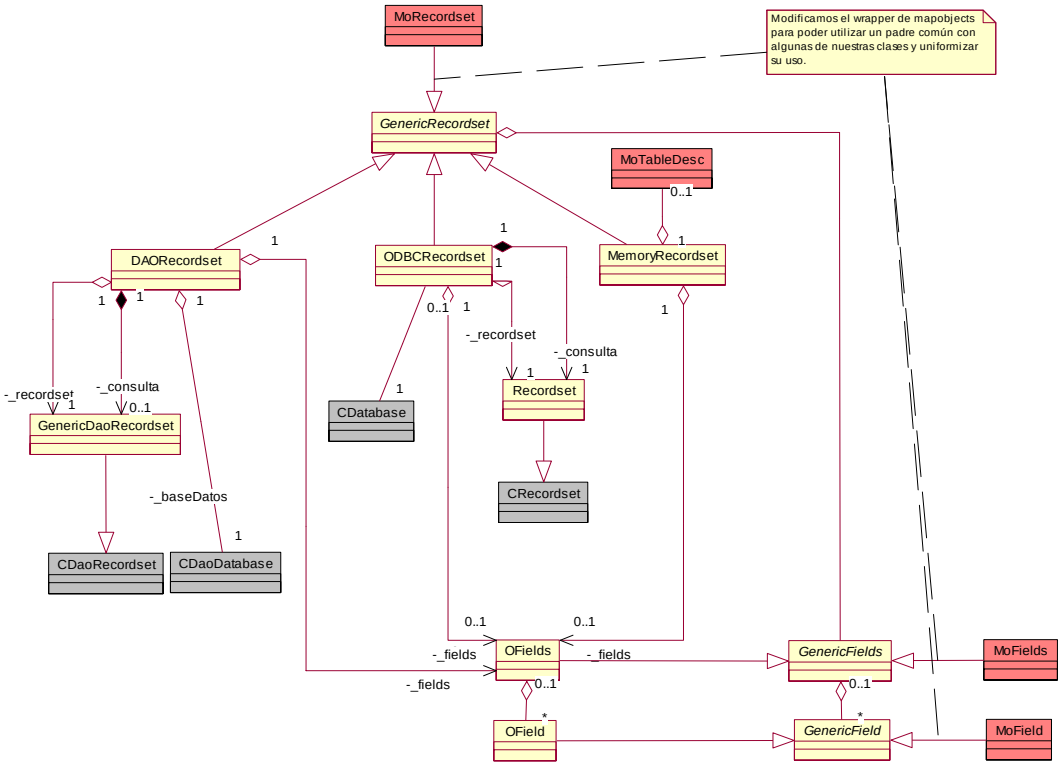

Figura 5. Jerarquía para el acceso a otras fuentes de datos

3.3 "Framework" para la creación de aplicaciones

El objetivo principal de los diseños presentados es acelerar el proceso de desarrollo de una aplicación con un componente SIG. Hasta ahora nos hemos centrado en aumentar la potencia del componente de visualización, en este momento nuestra estrategia cambia, pretendiendo crear una infraestructura que nos permita:

1. crear una serie de herramientas estándar en forma de módulos independientes y poder reutilizarlas de un sistema a otro

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

2. permitir la construcción de nuevas utilidades especificas a un desarrollo concreto y
3. que la inclusión o exclusión de dichas utilidades resulte trivial y no afecte al resto del sistema.

Así por ejemplo, casi todas las aplicaciones precisan de ciertas utilidades como un *LayerManager* (Figura 6), herramienta gráfica especializada en gestionar el conjunto de coberturas a representar en un control *Map* (tanto las normales de MapObjects como las nuestras), o las herramientas *Toolbox,* tanto de manejo de mapas y análisis, o los visualizadores de propiedades tabulares, o la pantalla resumen (*OverView*), que visualiza todo el área de interés del mapa indicando donde se encuentra el "zoom".

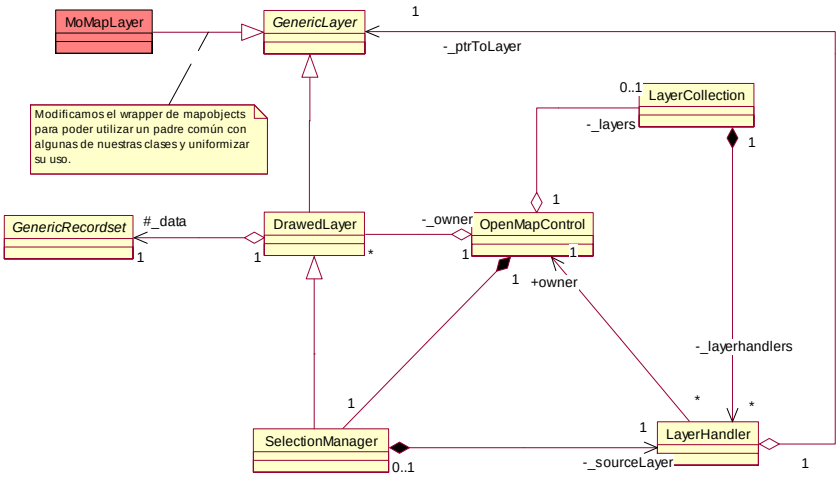

Figura 6. Ejemplo de utilidad creada: el *LayerManager*

Estas herramientas o utilidades responden al concepto de componente [9], es más, podría ser interesante implementarlas como ActiveX [10], lo que conlleva un esfuerzo extra de desarrollo pero nos permite mantener la capacidad mutiplataforma de desarrollo de MapObjects. Por lo tanto el problema no es sino el clásico de integración de componentes. Frecuentemente se ha afirmado que la integración de componentes es un subproblema del más general de integración de datos [9], puesto que los componentes se comunican siempre transmitiéndose datos de algún tipo, sin embargo, lo cierto es que una aplicación que se integre en base a componentes precisa de cierta estructura o "*framework*" que coordine la comunicación entre los mismos. ActiveX proporciona un mecanismo de eventos [10] para facilitar esta sincronización. Mediante este mecanismo, el control ActiveX avisa a su contenedor de que hechos o eventos ocurren, así el control *Map* de MapObjects avisa, de los eventos mas relevantes: clicks del ratón, perdidas de foco, etc., sin embargo, este mecanismo se queda corto para el desarrollo de aplicaciones por los siguientes motivos:

1. Puede haber más de una entidad a la que le interesen los eventos, por ejemplo un click de ratón en el control *Map* puede provocar que la *Toolbox* de manejo de mapas realice un "zoom", pero también que la utilidad que muestra las coordenadas se actualice.
2. Se necesita la creación de nuevos eventos que informen de un modo más detallado de cualquier cambio en el visualizador. Por ejemplo: un nuevo evento que indique que se ha producido un "zoom" interesa al *OverView* (utilidad que muestra un mapa de situación de la información geográfica que se está mostrando en detalle) y a la utilidad que muestra la escala o un nuevo evento que se "dispare" cuando se ha producido un cambio en la selección, cambio que afectará al dialogo que muestra los atributos no geográficos de los elementos seleccionados de forma tabular.

El mecanismo que hemos empleado para resolver este problema consiste en la creación de un canal de eventos (*EventChannel*, ver figura 7), este patrón de diseño es una variación del patrón sujeto-observador [11], en el que uno o más observadores "se subscriben" para ser notificados de cambios en el sujeto (elemento observado).

Con este modelo de trabajo toda entidad cuyos cambios de estado puedan interesar a alguien (por ejemplo si se realiza una selección en el control *Map*) "dispara" un evento insertándolo en el canal. El

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

canal a su vez se encarga de notificar a todos los objetos que han registrado su interés en ese determinado evento. Para implementar el patrón “canal de eventos” se inserta un objeto (el *EventChannel*) entre los observadores y el sujeto con el fin de desacoplarlos permitiendo que haya más de un sujeto.

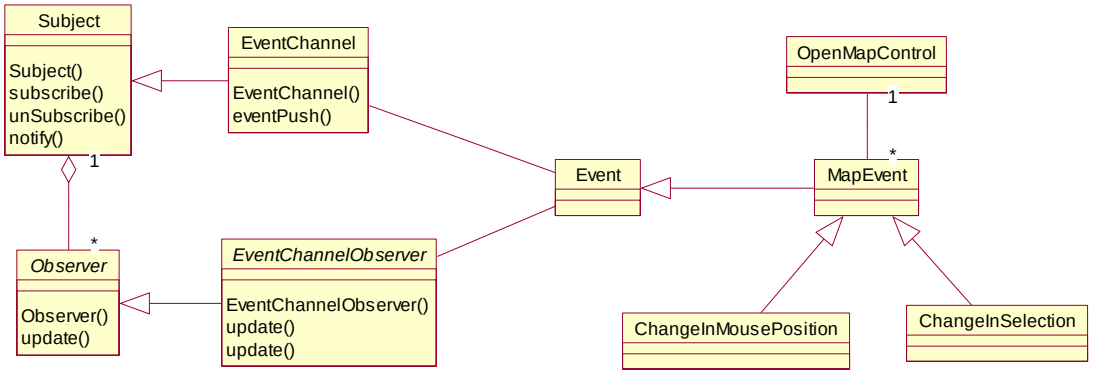

Figura 7. Simplificación de la jerarquía de eventos del "*framework*" de la aplicación

La creación de un canal de eventos proporciona importantes ventajas. Por ejemplo se puede especificar diversos tipos de eventos, de modo que cada uno contenga la información precisa para detallar el cambio, por ejemplo: el evento “cambio de capa activa” contendrá que capa ha dejado de ser la activa y cual pasa a serlo. Otra importante ventaja es el desacoplamiento, los sujetos desconocen a los observadores, cuando un sujeto “dispara” un evento no sabe a quién puede interesarle, es más, dependiendo de la configuración de la aplicación, puede que no haya nadie a quien le interese dicho evento. Los observadores tampoco deben conocer a todos los sujetos, sólo a aquellos cuyos cambios les interesen, por ejemplo, a la *OverView* sólo le interesan los cambios en el control *Map*, mientras que no tiene por que conocer la existencia de un *SelectiónManager* (utilidad que se encarga de gestionar las selecciones en las capas de información).

4 Conclusiones

Este trabajo ha presentado las técnicas utilizadas para el desarrollo de un componente de visualización geográfico con el objeto de acelerar el proceso de desarrollo de una aplicación con un componente SIG. Para el desarrollo de dicho componente se ha partido de MapObjects, aumentando la funcionalidad típica (apertura de capas, zoom, pan...), con capacidades extras como el acceso a fuentes de datos heterogéneas, capacidad de interoperar con aplicaciones externas y soporte a los grandes modelos de procesamiento distribuido.

MapObjects, como todos los componentes ActiveX, presenta una importante carencia: la imposibilidad de extender las funcionalidades de los componentes empleando mecanismos como puede ser la herencia directa. Esta característica obliga a realizar algunos "artificios" (como por ejemplo añadir clases genéricas que hagan de ancestro común entre las nuevas clases y las existentes en MapObjects) que conllevan cierto esfuerzo técnico. Sin embargo este esfuerzo se ve recompensado porque se crea la mayor parte de la infraestructura necesaria para la creación de una aplicación lo que permite acelerar la creación de nuevos sistemas de información. Por ejemplo si necesitásemos añadir un nuevo formato de fichero en el que se encuentre almacenado los datos geográficos, como la infraestructura de clases necesaria ya esta creada, "bastaría" con extender mediante herencia la clase genérica *GenericRecordset* y rescribir los métodos necesarios con el código necesario para acceder al nuevo formato. De esta forma tendríamos integrada en la aplicación, y por tanto en MapObjects, un nuevo formato gráfico.

(Draft) Sistemas de información geográfica: una aproximación desde la ingeniería del software y las bases de datos.
Monografías y publicaciones. Colección ingeniería informática.
Madrid: Fundación Dintel, 2001, p.47-62. ISBN 84-931933-1-3.

Bibliografía

[1] E.D. Kaplan (ed.). *Understanding GPS Principles and Applications*. Artech House Publishers. 1996.

[2] S. Vinoski. *CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments*. *IEEE Communications Magazine*, pp 46-55. Feb. 1997.

[3] Object Management Group. *Object Management Architecture Guide, Version 3.0*. Framingham, MA: Object Management Group, 1995.

[4] P. Muro-Medrano, D. Infante, J. Guillo, J. Zarazaga, J. Bañares. "A CORBA Infraestructure to Provide Distributed GPS Data in Real Time to GIS Applications". *Computers, Environment and Urban Systems*. Vol. 23, pp. 271-285. 1999.

[5] F.J.Zarazaga, P. Álvarez, R. López, J. Nogueras, J. Valiño, P.R. Muro-Medrano. "Use Cases of vehicle location systems using CORBA based distributed real-time GPS data and services". Computers, Environment and Urban Systems. Pendiente de publicación.

[6] The OpenGIS Abstract Specification. Topic 11: Metadata

[7] The OpenGIS Guide. Introduction to Interoperable Geoprocessing and the OpenGIS Specification. Third Edition

[8] OpenGIS Simple Features Specification For CORBA. Revision 1.0.

[9] C. Szyperski. "*Componente Software. Beyond Object-Oriented Programming"*. Addison-Wesley. 1997.

[10] D.S. Platt. "The essence of COM with ActiveX". Prentince Hall 1997.

[11] E. Gamma, R. Helm, R. Johnson, J. Vlissides. "*Desing Patterns. Elements of Reusable Object-Oriented Software".* Addison-Wesley Publishing Company. 1994.