Utilización de tecnologías de componentes COM/CORBA para la integración de visualizadores interoperables sobre MapObjects®

S. Comella, R. López, P. Fernández, J. Nogueras, J. Zarazaga y P.R. Muro-Medrano*

Grupo IAAA, Departamento de Informática e Ingeniería de Sistemas

Centro Politécnico Superior

Universidad de Zaragoza

María de Luna 3. 50014 Zaragoza

http://diana.cps.unizar.es/iaaa/

1 Introducción

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 del tipo de aplicaciones desarrolladas por nuestro grupo de trabajo.

Las aplicaciones SIG desarrolladas en nuestro grupo integran diversos subsistemas y fuentes de datos, algunos de los cuales preexistentes, siendo necesario que los distintos subsistemas interoperen entre si. Ejemplos de este tipo de aplicaciones son los sistemas de información en que sólo una parte de los datos tienen carácter geográfico o los sistemas de seguimiento de móviles en tiempo real. Estos sistemas necesitan un visualizador geográfico que, además de permitir las acciones típicas (apertura de capas, zoom, pan...), aporte 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:

* Dirección electrónica de contacto: prmuro@posta.unizar.es

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 está contenida un objeto remoto al que se accede mediante el 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, una solución más interesante y ya apuntada, consiste en implementar 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 un 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. El resultado es un componentes que permite, rápidamente, integrar en un sistema un visualizador geográfico y adaptarlo para que se ajuste a las fuentes de datos, entorno de aplicaciones y funcionalidades propias de dicho sistema.

2 Capacidades añadidas al 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 un doble interface uno COM para interaccionar con el componente de visualizador y otro CORBA para interaccionar con el modelo.

2.1 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 GIS, 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 [1]. No es necesario reinventar la rueda, ya existe un modelo para entidades geográficas autodescrito, orientado a objetos, potente y flexible: OpenGIS [2], concretamente en este trabajo se emplea una versión reducida de OpenGIS sobre CORBA [3], lo que entre otras cosas permite emplear un modelo distribuido.

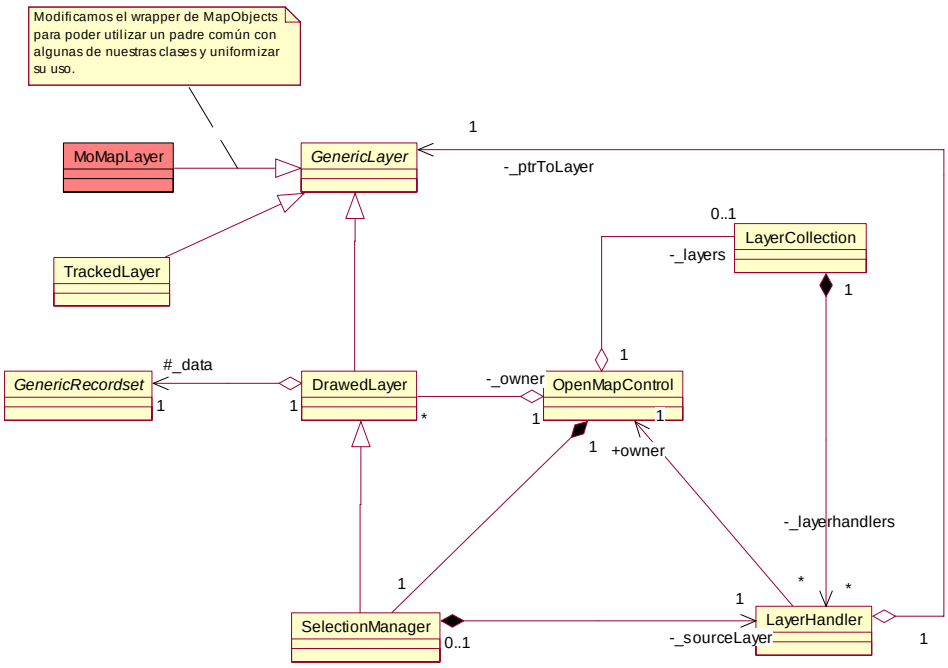

Figura 1. Jerarquía de capas del componente de visualización

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 1**.

Los tipos nuevos de capa (*DrawedLayer* y *TrackedLayer)* hacen esencialmente los 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.

2.2 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.

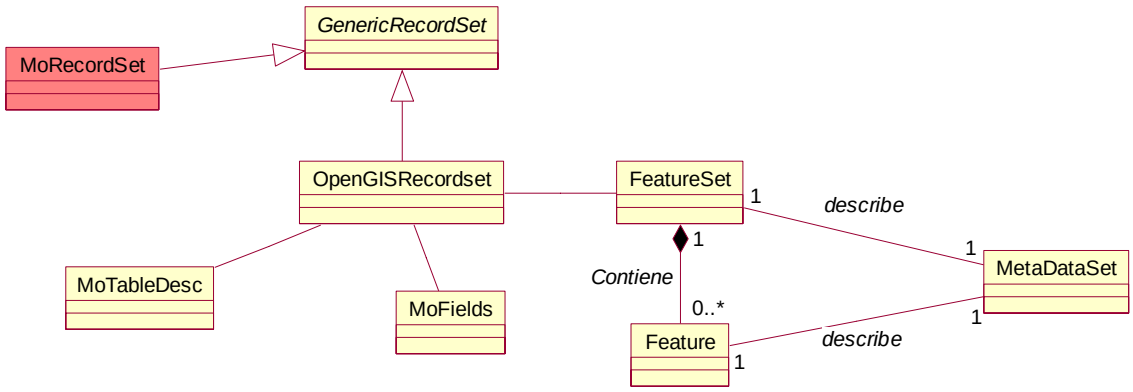

Figura 2. El OpenGISRecordset

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 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 un mismo 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 daots 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.

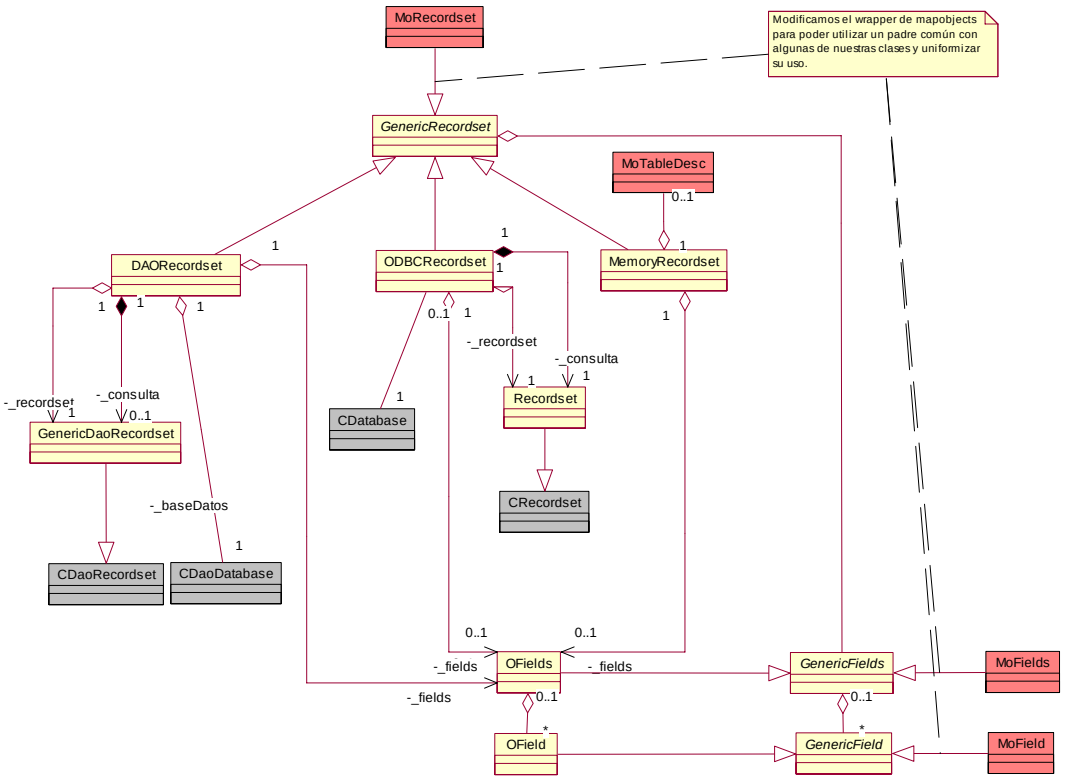

Figura 3. Jerarquía para el acceso a otras fuentes de datos

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 un mismo interface, 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.

2.3 "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 GIS, hasta ahora nos hemos centrado en aumentar la potencia del componente de visualización, ahora nuestra estrategia cambia, pretendemos 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
2. permitir la construcción de nuevas utilidades especificas a un desarrollo concreto y
3. que la inclusión o exclusión de dichas utilidades resulta trivial y no afecte al resto del sistema.

Así por ejemplo, casi todas las aplicaciones precisan de ciertas utilidades como un *LayerManager* (Figura 4), 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”.

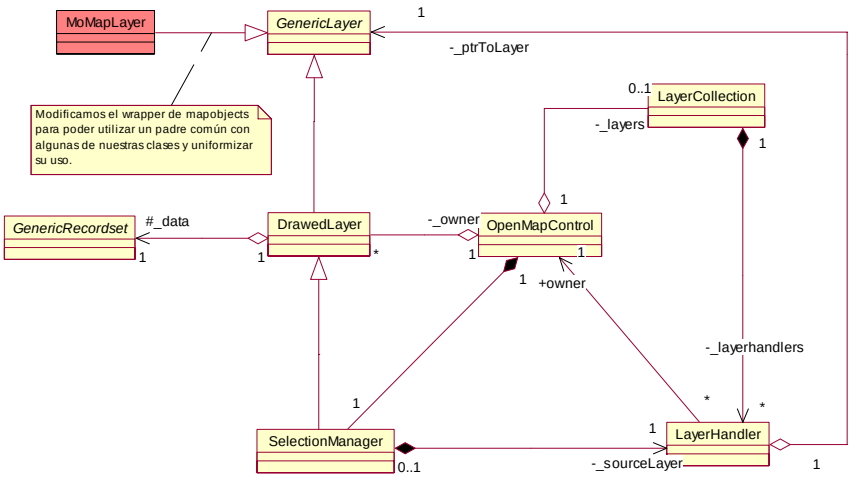

Figura 4. Ejemplo de utilidad creada: el *LayerManager*

Estas herramientas o utilidades responden al concepto de componente [4], es más, podría ser interesante implementarlas como a ActiveX [5], 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 [4],

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 [5] 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*), este patrón de diseño es una variación del patrón sujeto-observador [6], en el que uno o más observadores "se subscriben" para ser notificados de cambios en el sujeto (elemento observado).

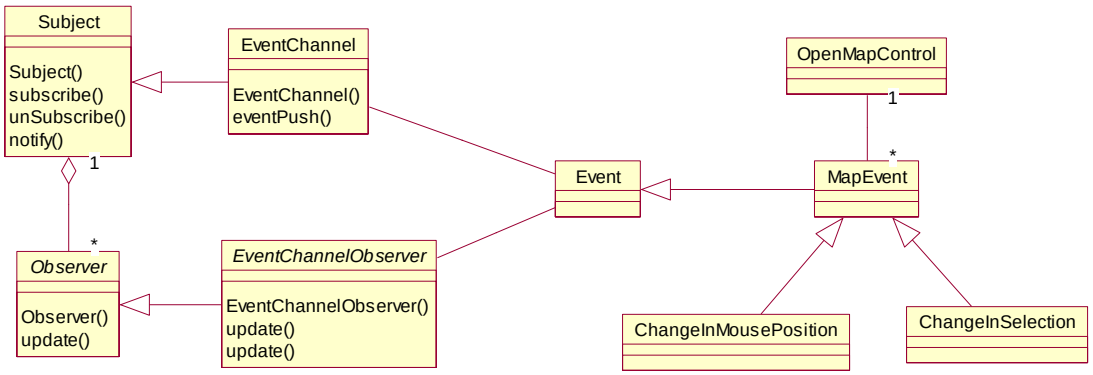

Figura 5 Simplificación de la jerarquía de eventos del "*framework*" de la aplicación

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 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 es sujeto con el fin de desacoplarlos permitiendo que haya más de un sujeto.

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).

3 Conclusiones

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 reescribir 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.

Agradecimientos

Este trabajo ha sido parcialmente financiado por el proyecto TIC98-0587 de la Comisión Interministerial de Ciencia y Tecnología (CICYT) y por el programa CONSI+D de la Diputación General de Aragón a través del proyecto P-18/96

Bibliografía

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

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

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

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

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

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