GisToolkit: Construcción de Aplicaciones GIS Empleando Componentes Reutilizables y Ligadura Dinámica

Ricardo Coppo Claudio Delrieux

Departamento de Ing. Eléctrica y Computadoras
Universidad Nacional del Sur, Bahía Blanca, ARGENTINA
e-mail: rcoppo@bblanca.com.ar, claudio@acm.org

1 Identificación del problema

En los últimos años ha habido un incremento notorio en el interés por los sistemas que procesan información geográfica (GIS). En respuesta a una creciente demanda de aplicaciones, la industria del software ha respondido creando una gran cantidad de programas, por lo general incompatibles entre sí, que compiten por la supremacía de uso y preferencia de los usuarios.

Como ninguno de los productos disponibles puede satisfacer la totalidad de las expectativas y requerimientos de todos los posibles usuarios, la mayoría de las actividades de procesamiento de datos GIS se realiza empleando no un único programa sino más bien combinaciones de los mismos. Se aprovecha de cada programa las características mas sobresalientes o más fáciles de usar. De esta forma se construyen verdaderas 'cadenas' de procesamiento en las que los resultados de un programa son utilizados como entrada en el siguiente.

Si bien esta metodología es funcional, presenta dos serios inconvenientes, la primera y fundamental es la dificultad de repetir cadenas de procesamiento en forma exacta, y la segunda es la poca flexibilidad frente a cambios de los datos de entrada o a la investigación de alternativas de procesamiento (ensayos de diferentes procesos parametrizados para identificar el mejor). Además, conforme aumenta la complejidad de la aplicación GIS y los resultados intermedios se utilizan para la obtención de diferentes vistas de visualización finales, la metodología no identifica o almacena la metainformación requerida de la red de programas involucrados complicando la documentación del mismo.

Por dicha razón, en este trabajo se propone como solución definir un mecanismo de desarrollo de aplicaciones GIS en base al paradigma de construcción de aplicaciones mediante componentes de ligadura dinámica. En este paradigma el usuario se encuentra con dos subsistemas: un subsistema modelador visual de aplicaciones y un subsistema compuesto de herramientas GIS basado en un *toolkit* de componentes que se ligan dinámicamente con el otro subsistema. Las herramientas del *toolkit* son ensamblados en forma gráfica dentro del modelador visual a fin de construir *pipes* individuales de procesamiento, hasta conformar una red de componentes que definen la aplicación. (Figura 1). A continuación se detalla los dos subsistemas y sus principios de diseño.

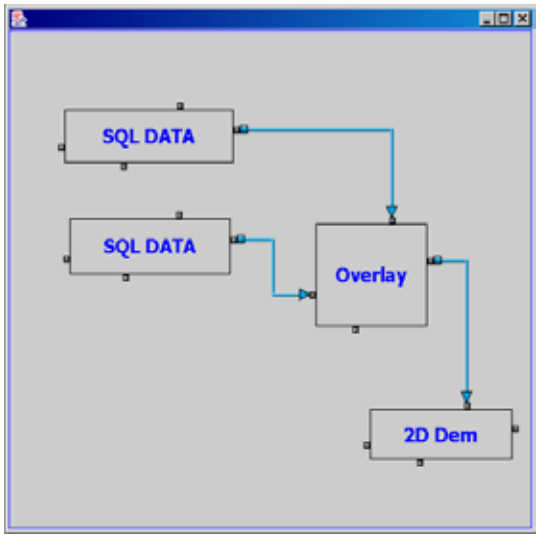

Figura 1: Componentes dinámicos interconectados

2 El GisToolkit

Cada herramienta del GisToolkit es un componente de software reutilizable que puede ser manipulado gráficamente por medio del modelador visual. En general podrán ser divididos en productores, transformadores y, en consumidores de información. El modelador visual no conoce de antemano la funcionalidad expuesta por el componente y aprovecha la ligadura dinámica de los componentes para descubrir sus características. Cada componente debe proveer soporte para resolver las propiedades de introspección, configuración, persistencia e interconexión de una manera consistente para permitir la construcción de la aplicación GIS.

2.1 Introspección

La introspección es la propiedad mediante la cual el componente expone su funcionalidad al modelador visual. Existen varias metodologías propuestas en la industria para realizarlo: el uso de estructuras de clases y herencia de objetos, la definición de interfases a través de lenguajes especiales y el uso de patrones de diseño de software.

Una de las primeras formas de obtener la introspección fue la de producir componentes construidos sobre una estructura jerárquica de clases en la que las clases superiores definen métodos y propiedades abstractas que deben ser implementadas por el componente final. Un prolijo diseño de la estructura también permite resolver los problemas de interconexión mencionadas más adelante. En esencia, la introspección en este caso se obtiene en base a especificaciones de diseño preestablecidas antes de iniciar el proyecto. Su mayor inconveniente es la falta de estandarización y normalización entre empresas y desarrolladores, lo cual determina que los componentes finales fuesen raramente compatibles entre si.

Una mejora sustancial se obtuvo con el empleo de lenguajes de especificación de interfases, por ejemplo el IDL (Interfase Definition Language) que es la base de la interconexion de componentes empleando CORBA. Casi todas las herramientas actuales de desarrollo de programas pueden procesar un archivo de IDL y producir objetos que luego son compilados y vinculados con la aplicación desarrollada. En este caso el inconveniente más agudo es la falta de flexibilidad en la extensión del modelador visual con nuevas herramientas sin

reconstruir toda la aplicación.

El diseño de software mediante patrones y la ligadura dinámica de los componentes en el momento de ejecución permite realizar la introspección en el momento más oportuno. Como ejemplo de esta tecnología puede citarse la declaración de propiedades y métodos como '*published*' empleado por los lenguajes visuales de Borland y los patrones establecidos por los JavaBeans del lenguaje Java.

2.2 Configuración dinámica

Cada componente posee propiedades que definen un estado. Es importante discernir entre aquellas propiedades que configuran el funcionamiento de la herramienta (por ejemplo la base de datos a la cual se accede, los parámetros de una proyección geográfica, etc.) y aquellos que son empleados durante la ejecución del mismo.

Entre los primeros también deben diferenciarse aquellas propiedades que son empleadas por el modelador visual y los que son del componente. La posición, el tamaño y el aspecto visual que presenta el componente sobre la superficie de diseño suelen ser heredadas del soporte de ejecución del lenguaje empleado mientras que las propias del componente sirven como parámetros de su función.

Cuando la propiedad requiere un mecanismo de definición complejo, (por ejemplo la ejecución de un cuadro de dialogo para obtener el nombre de un archivo, o de un 'asistente' para definir varias propiedades interrelacionadas) el subsistema debe poseer un mecanismo para invocar una función de configuración dinámica de su estado.

Nuevamente la elección puede ser estática, definida en el momento del ensamblado de la aplicación, o dinámica a través de una estructura de clases e interfases que permite registrar en el momento de la ligadura la existencia de métodos de configuración para cada propiedad. Este último es el mecanismo propuesto a través de la interfase '*PropertyEditor*' de Java que permite implementar clases especiales de configuración para las propiedades que lo requieran. El modelador visual puede comprobar la existencia de una clase de configuración y producir su invocación cuando lo desea.

2.3 Persistencia

Una vez instanciado el componente en el modelador visual el usuario configura parámetros específicos de la instancia a través de mecanismos estandarizados para los tipos de variable primitivas (entero, string, etc.) y mediante la invocación de los métodos de configuración dinámica mencionados anteriormente.

El estado de cada componente debe poder ser almacenado para ser transferido a la instancia de la herramienta en el momento de su ejecución y para que el modelador visual lo pueda incluir en su archivo de proyecto a fin de reconstruir la aplicación entre sucesivas invocaciones. Cada herramienta puede ser una combinación de otras herramientas y por ende el mecanismo de persistencia debe proveer una estructura adecuada para permitir empotrar una herramienta en otra.

Una forma sencilla de obtener la persistencia es a través de la simple serialización de objetos provista por la mayoría de los lenguajes de programación modernos. Sin embargo es preferible la selección de un mecanismo que permita compartir con mayor facilidad los diseños entre aplicaciones.

El lenguaje XML provee por excelencia una estructura sencilla y extensible para implementar la persistencia. Mediante el uso de tags adecuados se puede almacenar el nombre

de cada propiedad y su valor de cada componente. Mediante una DTD la estructura de toda la aplicación queda expuesta para que otras aplicaciones futuras pueden analizarlo y se independiza así el desarrollo del modelador visual del lenguaje de programación.

2.4 Interconexion

La interconexión se refiere a la capacidad de interacción entre los herramientas del GisToolkit. Se realiza obtiene bajo el paradigma de la transferencia de eventos a través de la red de herramientas interconectados. Una herramienta del GisToolkit puede declararse productor o consumidor de eventos de un determinado tipo. Siguiendo la convención usada en las paquetes de Java, un consumidor debe registrarse con el productor de la que quiere recibir notificación de un evento. Los productores mantienen una lista dinámica de los consumidores interesados.

A los fines de no crear una interfase de comunicación estándar para todos las herramientas del GisToolkit (que le quitaría gran flexibilidad al desarrollo) el modelador visual utiliza el mecanismo de introspección para identificar los eventos que pueden ser generados o consumidos por cada herramienta; permitiéndose la interconexión solamente entre componentes.

3 El modelador visual

El segundo subsistema está compuesto por un '*canvas*' o 'formulario' de diseño visual sobre la cual se pueden colocar las herramientas del toolkit. Durante el diseño de la aplicación, el usuario elige los componentes que desea emplear de una barra de botones y los instancia en alguna posición del '*canvas*' en forma similar a lo realizado con un lenguaje de programación visual.

El modelador presenta en forma de pequeños iconos los 'puntos de conexión' que admite cada componente. El usuario interconecta los mismos empleando el ratón para conformar la red dirigida. Ante cada nueva conexión el modelador visual realiza la introspección de los componentes involucrados y trata de hallar eventos compatibles entre los mismos. De existir más de una posibilidad el usuario debe discernir la correcta combinación de consumidor y productor.

Una vez finalizado el diseño de la aplicación el modelador visual permite su ejecución a través del disparo de un evento inicial, este luego se propaga a través de la red. Uno de los aspectos más interesantes del diseño de este subsistema es su capacidad de incorporar nuevas herramientas sin recompilar el programa, e inclusive podría configurarse para localizarlos en una red o la internet.

Todo el diseño de la aplicación puede ser almacenado empleando la misma estructura de XML propuesto para la persistencia de las herramientas, extendiéndola con nuevos tags para almacenar la estructura misma de la red. Se logra así una estructura donde una aplicación completa puede ser considerado como un nuevo componente e integrado como 'super' herramienta en el *GisToolkit*.

4 Conclusiones

El uso de componentes dinámicamente enlazados provee al desarrollo de aplicaciones GIS una nueva metáfora de construcción. El *GisToolkit* y el *Modelador Visual* permiten enla-

zar componentes, posiblemente desarrollados por diferentes organizaciones, en aplicaciones completas o a su vez en nuevos componentes más complejos.

Los componentes del *Gistoolkit* no deben ser especificados de antemano, el *Modelador Visual* descubre los métodos y propiedades expuestas por la herramienta del toolkit empleando un mecanismo de introspección. El usuario enlaza los componentes en el entorno visual, quien verifica la correcta conexión de los mismos. Una vez finalizado la aplicación la misma puede ser almacenado empleando la el mecanismo de persistencia de los objetos instanciados.

El sistema propuesto requiere de diversas tecnologías que actualmente están en permanente evolución. Diversos organismos están trabajando en estándares cuya adopción aportarán significativamente a la construcción de aplicaciones distribuidas. En particular el enlace dinámico presentado por el lenguaje Java, junto con su arquitectura de JavaBeans, resulta de especial interés. Otras arquitecturas como las propuestas por CORBA (Object Management Group) y DCOM (Microsoft) también son aplicables.

A medida que estos estándar maduren y sean adoptadas por la comunidad de desarrolladores será posible integrar una red de aplicaciones GIS que facilitará el trabajo de los investigadores quienes no requerirán de conocimientos especializados de informática para construir sus modelos.

Bibliografía

1. Alan Burns and Andy Wellings, *Real-Time Systems and Programming Languages*, Addison-Wesley, 1997.
2. Chauvet, Jean-Marie *Corba, ActiveX y Java Beans*, Ediciones Gestión 2000 S.A., Barcelona, 1997.
3. Dibble, P.: *Real-Time Java Platform Programming.* Prentice Hall PTR, 2002.
4. Graham, Hamilton *JavaBeans API Specification*, Sun Microsystems, 1997.
5. Borland Software Corporation, *Delphi Developer's Guide*, Inprise Corp., Scotts Valley, CA, USA, 2001.
6. Marchal, B. *XML con ejemplos* Pearson Educación de Méjico, 2001.
7. OMG (Object Management Group), *The Common Object Request Broker Architecture and Specification*, OMG, Framingham, MA, USA, 1995.
8. Page-Jones, M. *Fundamentals of Object-Oriented Design in UML*, The Addison-Wesley Object Technology Series, 1999.
9. Pressman, Roger S., *Ingeniería del Software, Un enfoque práctico*, 5ta edición, McGraw Hill, 2002.
10. Siegel, John *CORBA Fundamentals and Programming*, John Wiley & Sons, New York, 1996.
11. Taylor, David A. *Business Engineering with Object Technology*, John Wiley & Sons, NY, USA, 1995.