Software Component Development based on the Mediator Pattern Design: the Interactive Graphic Organizer case

F. A. Almarza, H. R. Ponce and M. J. López

***Abstract*— This article presents the redesign of a software application denominated interactive graphic organizer (IGO) using a software component architecture and the mediator pattern design. The original IGO application presented high level of coupling and primitive communication interface which generated problems in the composition of more complex software applications. The main results were a new IGO version that fulfills the characteristics associated with software components, in particular, a robust interface; and mainly, a low coupling level derived from the use of the mediator pattern design. This new version has permitted the reuse of the IGO in the composition of more complex applications.**

***Keywords*— Interactive graphic organizer, software component, mediator pattern design, composition.**

I. INTRODUCCIÓN

EN GENERAL, la utilización de esquemas que combinan elementos lingüísticos, tales como palabras y frases, y elementos no lingüísticos, tales como símbolos, figuras y flechas, para representar relaciones se conocen como *organizadores gráficos* [1], [2]. La utilidad de los organizadores gráficos radica en su capacidad para representar visualmente una operación cognitiva [3]. Por ejemplo, existen formas visuales que ayudan a establecer relaciones causales, componer analogías, identificar similitudes y diferencias, establecer secuencias y presentar un argumento estructurado. Existen organizadores gráficos específicos para representar y desarrollar cada una de estas habilidades cognitivas [4], [5], [6]. Por ejemplo, como se ilustra en la Fig. 1, a través de un diagrama de similitudes y diferencias [7], el alumno cuenta con una técnica visual que le permite efectuar comparaciones entre dos o más objetos o sucesos. Cognitivamente, el procedimiento requiere establecer los elementos a ser comparados, indicar los atributos de comparación y contraste, escribir en los nodos del centro las similitudes y en los nodos laterales las diferencias [8].

Los organizadores gráficos constituyen también una herramienta efectiva y poderosa para la representación y estructuración de contenidos, facilitando su comprensión, ya que permiten al profesor exponer contenidos complejos en un lenguaje accesible e integrar y relacionar nuevos conocimientos con los conocimientos previos del estudiante [9], [10], [11]. Los organizadores gráficos también ayudan al aprendiz a organizar, secuenciar, y estructurar su conocimiento y facilitan la aplicación de nuevas estrategias a los desafíos de aprendizaje que enfrente. También facilitan el descubrimiento de patrones, interrelaciones e interdependencias y el desarrollo del pensamiento creativo [12].

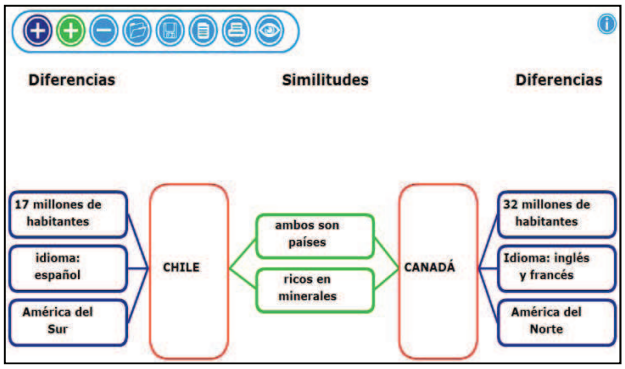

Figura 1. OGI de diferencias y similitudes.

Diversos estudios también indican efectos educativos importantes en la utilización de organizadores gráficos. Por ejemplo, en estudios utilizando técnicas de meta-análisis, el tamaño promedio de los efectos ("effect size") reportados van desde 0.5 a 1.31 [2].

Dadas las importantes ventajas de utilizar organizadores gráficos en contextos educativos, en el año 2005, iniciamos el desarrollo de un conjunto de componentes de software que denominamos Organizadores Gráficos Interactivos (OGI). La aplicación de software resultante es un conjunto de OGI implementados en Adobe Flash utilizando Action Script 2.0. Cada OGI está básicamente dotado de: (a) funcionalidades, que le permiten crear, modificar, eliminar, guardar, recuperar e imprimir lo que el estudiante va desarrollando o ha concluido; y (b) interactividad, mediante la agregación y edición de formas gráficas.

Además, cada organizador es de fácil integración con otras plataformas compatibles con Adobe Flash, tales como insertarlo en una diapositiva de una presentación de Microsoft PowerPoint [13] o bien utilizarlos para construir otras aplicaciones de software de características similares [14]. Durante dos años de desarrollo, se logró implementar un total de 100 organizadores gráficos en idioma español y otros 100 organizadores equivalentes en idioma inglés. Además, fue posible con dicha tecnología desarrollar otras aplicaciones de

F. A. Almarza, Universidad de Santiago de Chile, Santiago, Chile, francisco.almarza@gmail.com
H. R. Ponce, Universidad de Santiago de Chile, Santiago, Chile, hector.ponce@usach.cl
M. J. López, Universidad de Santiago de Chile, Santiago, Chile, mario.lopez@usach.cl

mayor complejidad, como el programa de formación en estrategias lectoras e-PELS [15].

Sin embargo, la aplicación OGI originalmente desarrollada presentaba algunos problemas arquitecturales que hacían necesario un rediseño utilizando un nuevo patrón de diseño, debido a su alto acoplamiento interno; actualizar aspectos relacionados con la interfaz de comunicación, debido a su baja capacidad de composición y aprovechar las nuevas características ofrecidas por ActionScript 3.0 en relación al desarrollo de software basado en componentes. Por lo tanto, el objetivo de este artículo consiste en presentar un rediseño de la versión inicial del OGI utilizando un esquema de desarrollo basado en componentes de software y la utilización del patrón de diseño mediador.

I. MARCO CONCEPTUAL

1. Patrón de diseño Mediador

Orientado al desarrollo de software, un patrón de diseño, consiste en una descripción de clases y objetos que se comunican entre ellos; haciendo posible resolver una problemática de diseño general en un contexto particular [16], [17]. Actualmente, es posible encontrar una variedad de patrones de diseño orientados a solucionar problemas generales. Por ejemplo, el patrón de diseño modelo-vista-controlador (MVC), patrón observador, patrón fachada, entre otros [17].

El patrón de diseño mediador consiste en definir un objeto que tenga como finalidad encapsular la interacción presente en un conjunto de objetos. Con esto se logra estimular la pérdida de acoplamiento, ocultando así las referencias explícitas entre los objetos, modificando así su interacción de forma independiente. La idea básica del patrón mediador es que cada uno de los objetos presentes se comunique con otro a través de un *mediador* central, el que a su vez, conoce a cada uno de los objetos que están a su alcance, pudiendo manipular sus estados de ser necesario.

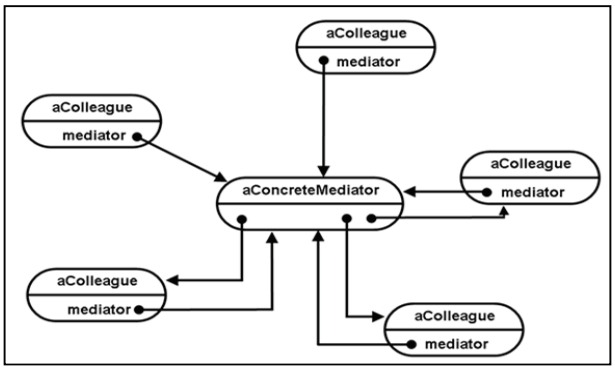

Figura 2. Estructura básica del patrón mediador [17].

Dos son los requisitos que debe cumplir el patrón de diseño mediador. El primero es una clase conocida como "mediador". Esta define la interfaz y las funcionalidades genéricas de la misma. Dependiendo de las necesidades, se pueden implementar mediadores concretos para implementar funcionalidades específicas. El segundo requisito son todos los objetos que se encuentran juntos al mediador y se definen como "colegas". Estos se basan en clases que se puedan comunicar con él. La estructura básica de este patrón de diseño se presenta en la Fig. 2.

2. Desarrollo de software basado en componentes

Se define un componente de software como una unidad de composición con una interfaz contractualmente especificada y explícita sólo con dependencias de un contexto, el que puede ser desplegado de forma independiente o sujeto a la composición de terceros [18]. También es importante destacar que un componente de software posee un conjunto atributos que los caracterizan [19], [20]:

Composición: consiste en la capacidad del componente para agruparse con otros componentes y así obtener un componente de mayor granularidad. **Encapsulamiento:** consiste en la capacidad del componente para ocultar los detalles de su implementación, funcionando como una caja negra. **Interoperabilidad:** consiste en la posibilidad del componente de trabajar tanto de forma independiente como interactuando con otros componentes permitiendo funcionalidades de mayor complejidad. **Multiplataforma:** consiste en la capacidad del componente para funcionar, independiente del sistema operativo y hardware utilizado. Esto permite, aumentar su reutilización. **Auto-contenido:** consiste en la capacidad del componente de trabajar con mínima dependencia de otros componentes o aplicaciones para realizar sus operaciones. Para lograr dicho objetivo, el componente debe presentar un bajo acoplamiento y una alta cohesión.

3. Interfaz de componentes

De acuerdo a lo indicado anteriormente, es crucial que un componente de software cuente con un punto de acceso único y estandarizado para lograr su interoperabilidad con otras aplicaciones. Este punto de acceso es su interfaz. Se define una interfaz como la especificación de su punto de acceso. Es a través de ese punto por el cual el cliente accede a los servicios que provee el componente [21]. Es importante destacar que una interfaz no ofrece la implementación de sus operaciones. Simplemente las nombra y provee sus descripciones y protocolos. La división realizada permite significativas mejoras, entre ellas:

- La posibilidad de realizar cambios en la implementación sin cambiar la interfaz. Esto permite realizar mejoras en el componente sin tener que reconstruirlo.
- Al incluir nuevas interfaces (e implementaciones) no es necesario cambiar la implementación existente, pudiendo así mejorar la adaptabilidad del componente.

II. PROBLEMA

El OGI consiste técnicamente en una vista que permite la inclusión de diversas formas básicas (cuadriláteros, círculos, óvalos, flechas, entre otros) en conjunto a una iconografía sugerente según el organizador utilizado. Esta vista es controlada por una barra de herramientas ubicada en su parte superior y que permite acceder a todas sus funcionalidades. El OGI puede dividirse gráficamente en dos partes claramente marcadas: el espacio de trabajo y la barra de herramientas (Fig. 3).

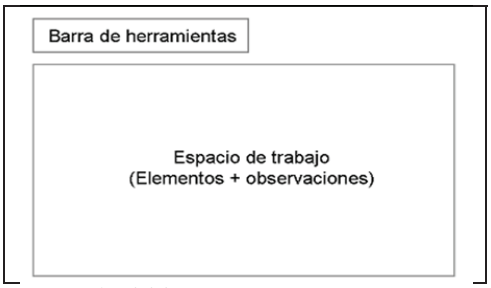

Figura 3. Estructura visual del OGI.

Espacio de trabajo: En esta área se incluye la mayor parte de los elementos presentes en la aplicación, los que se pueden clasificar en: (1) *elementos de diagrama:* correspondientes a las diversas estructuras gráficas que se incluyen en el diagrama y que almacenan la información principal del OGI; y (2) *elementos de observaciones:* corresponden a las estructuras gráficas incluidas para almacenar información complementaria a la incluida en los elementos de diagrama. **Barra de herramientas:** Área correspondiente a la parte superior de la vista del OGI. En esta sección se incluyen los diversos elementos gráficos (botones) que permiten la interacción entre el usuario y la aplicación.

Al iniciar la aplicación OGI, ésta le presenta al usuario los diversos elementos posibles de utilizar, tanto el espacio de trabajo como la barra de herramientas. Además, pone a disposición un conjunto de funcionalidades realizables en el espacio de trabajo, entre ellas: crear, guardar, imprimir y cargar organizador; insertar y eliminar elementos del diagrama, agregar y eliminar observaciones.

Luego de utilizar los OGI en diversos proyectos, tanto evaluando el organizador gráfico [13], como en el desarrollo de nuevas aplicaciones [14], [15], surgieron nuevos requerimientos y demandas sobre el OGI que hicieron necesario el desarrollo de una nueva versión. Los principales problemas generados por la versión original del OGI fueron:

Alto acoplamiento: El OGI cuenta con una gran cantidad de elementos gráficos (bloques, figuras, avisos, entre otros) que interactúan internamente y con el usuario. Todos estos elementos se incluyen en la misma capa visual provocando un alto acoplamiento, lo que dificulta interacción entre los elementos y observándose algunos problemas de usabilidad derivados de dicho acoplamiento (ejemplo: interposición de elementos). La Fig. 4 muestra una vista del entorno de desarrollo del OGI donde se observa dicho acoplamiento.

Interfaz primitiva: El OGI está orientado a su utilización tanto de forma independiente como en conjunto a otras aplicaciones. En este ámbito, el OGI puede ser incluido sin problemas tanto en aplicaciones Web como de escritorio. Pero no cuenta con un punto de acceso para servicios o eventos, presentando falencias como componente, entre las que se destacan: (1) Composición: El OGI puede incluirse en otras aplicaciones y/o componentes, pero no es posible acceder a sus servicios, esto porque no cuenta con un punto de acceso para compartir dichos servicios. Por lo tanto, no puede ser utilizado en la composición de componentes de mayor granularidad. (2) Encapsulamiento: en la versión original del OGI el encapsulamiento es total debido a la carencia de interfaz de comunicación. (3) Interoperabilidad: El OGI puede trabajar de forma independiente y funcionar en plataformas compatibles con Flash. Sin embargo, al no comunicarse con otros componentes, no puede interoperar para componer una aplicación de mayor complejidad.

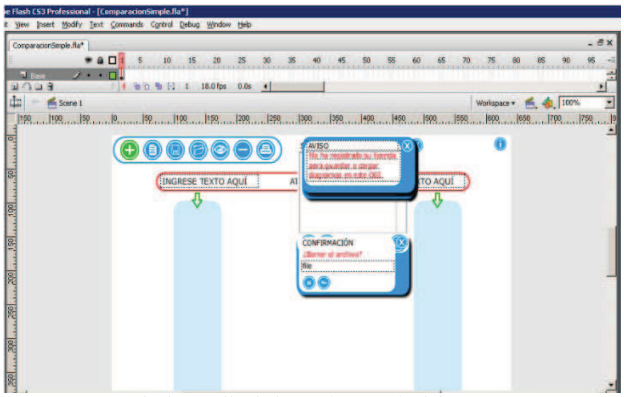
Figura 4. Entorno de desarrollo de la versión previa del OGI.

Barreras de actualización: Al tener un alto grado de acoplamiento, cualquier modificación realizada al OGI implica desde modificar pequeñas líneas de código hasta agregar nuevos elementos que implican mayor esfuerzo de programación. En este último caso se hace necesaria la revisión de posibles efectos colaterales con los elementos que componen el OGI debido a que son parte de la misma capa visual. Además, si el cambio aplica a todos los OGI, será necesario hacer dichas modificaciones en el código de cada OGI, demandando tiempo y esfuerzo de programación.

Por las circunstancias anteriormente explicadas, fue necesario realizar un rediseño del OGI, de tal forma que la nueva versión cumpla con un conjunto de nuevas características. Entre ellas:

- Que presente una arquitectura adecuada para un software pequeño, reutilizable y actualizable.
- Reestructurar el OGI de forma que los elementos no se acoplen, mejorando el acceso e interacción con los mismos.
- Una comunicación e interacción rápida y transparente, sean estos OGI u otras aplicaciones que requieran de sus servicios.
- Que la aplicación esté abierta a nuevas modificaciones y actualizaciones, las que puedan ser realizadas de forma rápida y en un tiempo aceptable.
- Que la estructura adoptada, permita facilitar y potenciar las características de la arquitectura a utilizar.

III. REDISEÑO DEL OGI

A continuación, se describen las etapas más significativas desarrolladas en el rediseño del OGI orientado a componentes de software.

Como requisitos de entrada para el rediseño, se decidió revisar el conjunto de experiencias recopiladas acerca del OGI, realizadas con profesores y alumnos [15]. Estas experiencias incluían evaluaciones de usabilidad, pruebas con usuarios, y desarrollo de nuevas aplicaciones que requerían la utilización de los OGI. En dichas experiencias se logró

obtener una gran cantidad de sugerencias de mejoras para la nueva versión del OGI.

Junto a iniciar el rediseño y construcción del nuevo OGI basado en una arquitectura de componentes de software, se decide utilizar el patrón de diseño mediador dado las características de la problemática generada por el OGI inicialmente implementado. Entre los aspectos considerados relevantes para utilizar el patrón mediador estaban: (a) permitir desacoplar los diversos objetos incluidos en el componente; (b) simplificar la comunicación entre los objetos; (c) abstraer el cómo cooperan los objetos; y (d) centralizar el control del componente.

4. Diseño arquitectural del OGI

De acuerdo a lo especificado en la problemática, los diversos elementos del OGI fueron agrupados en capas. Cada una de las capas obtenidas está asociada con una característica específica del componente, permitiendo la utilización del patrón de diseño mediador. Las capas propuestas para el OGI fueron:

Diagrama: esta capa contiene los elementos principales del OGI (flechas, bloques de contenidos, entre otros). **Barra de herramientas**: esta capa incluye una barra de botones para interactuar con el diagrama. **Observaciones**: esta capa incluye y maneja bloques de contenidos adicionales para complementar la información del diagrama. **Avisos**: esta capa contiene todos los elementos que presentan mensajes al usuario para realizar alguna acción.

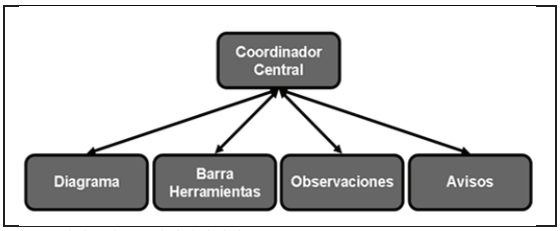

Figura 5. Modelamiento inicial del OGI.

Para la comunicación entre las capas, se debe contar con una entidad coordinadora central, evitando así la comunicación directa entre las demás capas. En la Fig. 5 se puede apreciar la aproximación realizada.

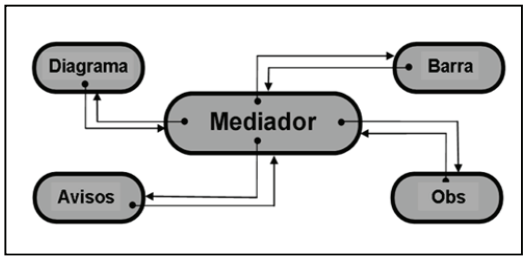

Figura 6. Estructura mediador asociada al OGI.

Tal como se indicó anteriormente, de los diversos patrones de diseño, el patrón mediador permite a un coordinador central encargarse de la comunicación entre un conjunto de objetos, desacoplando los diversos elementos dentro del componente. La nueva estructura del OGI consiste en una unidad central conocida como mediador, la que interactúa con un conjunto de capas independientes entre sí conocidas como colegas. Cada una de las capas tiene sus propias funcionalidades, manteniendo funcionalidades comunes de comunicación exclusiva con el mediador (Fig. 6).

Cada una de las divisiones realizadas tiene sus propias características y funcionalidades, pero también debe heredar funcionalidades comunes con las otras capas, tales como la comunicación con el coordinador central. Esto hace que se requiera la definición de un diagrama de clases (Fig. 7).

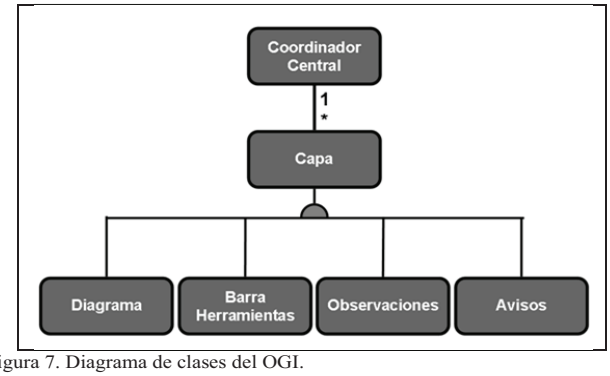

Figura 7. Diagrama de clases del OGI.

A continuación se procedió al diseño de la interfaz de comunicación requerida para un componente de software.

5. Diseño de interfaz

En la fase de desarrollo de la interfaz, se establecieron las especificaciones para que el OGI pudiese interactuar con otros componentes o aplicaciones externas, solicitando y prestando servicios. Para el OGI se definen dos tipos de prestaciones: servicios y eventos.

Servicios: Corresponden a acciones que una aplicación o componente externo solicita al OGI para que entregue información de su estado o contenidos. O bien, para incorporar nuevos datos o la realización de tareas específicas.

Eventos: Corresponde a la técnica para especificar determinadas tareas que deben realizarse como respuesta a acciones concretas. Estas acciones son originadas por la interacción entre el usuario y el entorno del componente (ejemplo: presionar un botón y realizar un cambio en la interfaz gráfica). En los eventos pueden identificarse tres elementos importantes: (1) **Origen del evento:** Corresponde al objeto o componente que gatilla el inicio del evento. También se denomina objetivo del evento, debido a que el entorno le asigna un evento a ejecutar. (2) **Evento:** Corresponde a la identificación del evento propiamente tal. Esto permite individualizarlo de un conjunto de eventos distintos cuyo origen es el mismo objeto. (3) **Respuesta:** Corresponde al procedimiento a seguir luego que se activó el evento desde el objeto origen.

Los diversos servicios ofrecidos por el OGI son definidos por una clase *Interface*, en las que se incluye el nombre, los parámetros requeridos y el retorno de dicho servicio, si fuese necesario. Esta clase estandariza la comunicación con otras aplicaciones. Dicha clase la hemos denominado como *IgoApi*. Los eventos, por su parte, serán generados por el componente

al momento que se realice alguna acción (guardar, cargar y nuevo diagrama, entre otras). El listado de los servicios y eventos se presentan a continuación (Fig. 8).

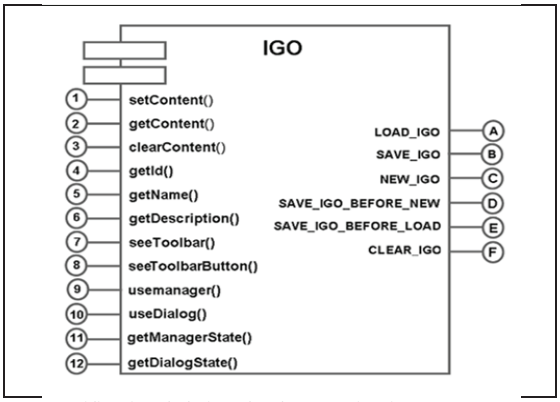

Figura 8. Especificación de la interfaz de comunicación.

6. Diseño detallado

A continuación, se presenta el modelamiento interno de la nueva versión del OGI en base a la nomenclatura usada en el modelo de componentes JavaBeans, mostrando el patrón de diseño mediador utilizado (Fig. 9).

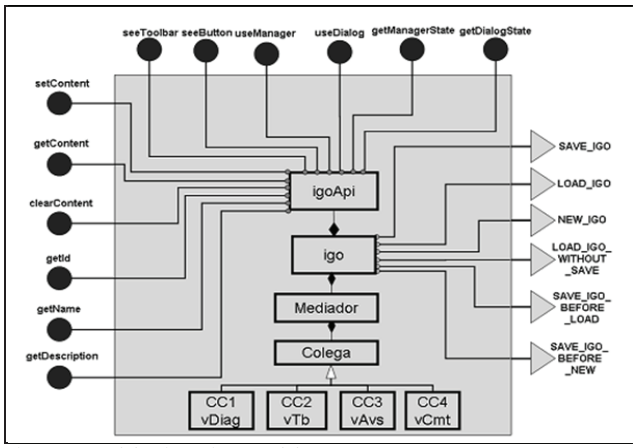

Figura 9. Modelamiento interno del componente OGI.

La incorporación de la interfaz *IgoApi* es importada como librería a la clase que implementa al mediador (Fig. 10).

```
// Se importa la interfaz
  import clases.Interfaz.IgoApi;
  // La clase principal hereda de MovieClip e implementará los métodos
  // definidos en la interfaz IgoApi
  public class concreteMediator extends MovieClip implements IgoApi {
    // Código de la clase
  }
```
Figura 10. Inclusión de la Interfaz IgoApi.

La implementación del patrón de diseño mediador también requirió el desarrollo de las funcionalidades de comunicación entre el mediador (coordinador) y los colegas (capas). A continuación, se definen dichas funcionalidades:

ListernMediator: funcionalidad que es implementada por los colegas. Dicha funcionalidad pide una referencia al mediador para que escuche las solicitudes de servicios realizadas por los colegas (Fig. 11).

```
Public override function listenMediator(m:Mediator):void
{
        this.objetoMediador=m;
}
```
Figura 11: Implementación del método listenMediator en una clase colega.

Send: funcionalidad que permite el envío de información desde los colegas (Fig. 12) hacia el mediador (Fig. 13). Esto permite la primera parte de la mediación, consistente en el envío de una solicitud por parte de los colegas hacia el mediador solicitando los servicios requeridos.

```
public override function send(mensaje:String):void
{
        this.objetoMediador .send(mensaje,colega.id);
}
```
Figura 12. Implementación del método send en una clase colega.

```
public override function send(mensaje:String,id:String):void
{
        switch(mensaje){
          // Lista de mensajes posibles
          case "ACCION1":
                // Notifica a diagrama  que ingrese elemento
                diagrama.notify("ACCION1");
                 break;
          default:
                 // No es acción normal
                 Trace("Mensaje no corresponde");
        }
}
```
Figura 13. Implementación del método send en una clase mediador.

Notify: funcionalidad que permite el envío de información desde el mediador hacia los colegas. Esta información puede ser tanto una respuesta del mediador hacia un colega, una orden para ejecutar un método, o bien, la solicitud para una configuración a realizar en dicho colega (Fig. 14).

```
public override function notify(mensaje:String):void
{
                        switch(mensaje){
          // Lista de mensajes posibles
          case "ACCION1":
                // El colega debe ejecutar la acción solicitada
                accion1();
                 break;
          default:
                 // No es acción normal
                 Trace("Mensaje no corresponde");
        }
}
```
Figura 14. Implementación del método notify en una clase colega.

IV. RESULTADOS OBTENIDOS

Luego de desarrollar el OGI, fue posible desprender un conjunto de ventajas obtenidas por la utilización de los nuevos recursos. A saber: desacoplamiento de las diversas partes del OGI, desarrollo del OGI como un componente de software, y actualización y mejora de las diversas partes del componente.

7. Desacoplamiento de los elementos del OGI

En el rediseño del OGI, se consideró el problema del alto acoplamiento provocado porque todos los elementos estaban en la misma vista, dificultando la interacción entre ellos. Este hecho se observó principalmente en funcionalidades que involucraban arrastre e intersección entre elementos. La idea a seguir consistió en agrupar los elementos en capas de funcionamiento común pasando a ser, según el patrón mediador, clases colegas que permiten desacoplar los elementos presentes en el componente. La Fig. 15 visualiza el desacoplamiento realizado en el OGI.

Este desacoplamiento significó una gran ventaja debido a la posibilidad de interactuar con un elemento de una capa en particular sin afectar el estado de las otras capas presentes. Esta característica es muy importante al momento de controlar los diversos elementos presentes en el OGI, ya que es una aplicación orientada a estructurar información gráficamente.

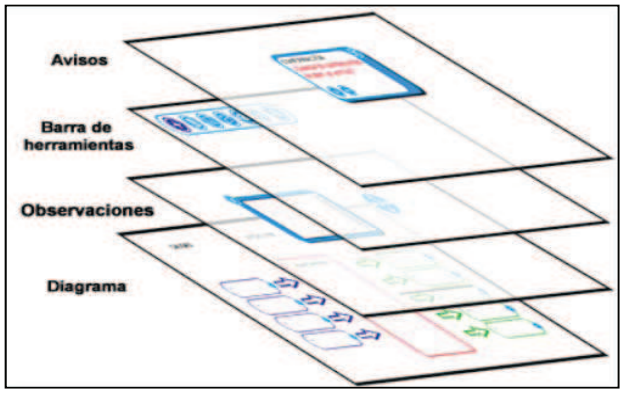

Figura 15. Desacoplamiento de los elementos gráficos del OGI.

8. Desarrollo como componente de software

Como componente de software, el OGI cumplía de forma parcial con las características básicas de un componente de software. En su rediseño, se optó por seguir el desarrollo basado en componentes, incluyendo y/o corrigiendo dichas características que lo limitaban como componente propiamente tal. Esta decisión permitió contar con varias ventajas tales como la reutilización y la adaptabilidad. A continuación, se indican las características de un componente de software presentadas previamente y de qué forma el OGI cumple con ellas.

Composición: Al incluir la interfaz de comunicación *IgoApi,* se genera un punto de acceso para que el OGI cuente con un conjunto de prestaciones disponibles para cualquier componente con el que se deba componer para lograr un componente de mayor complejidad. **Encapsulamiento:** el OGI cuenta con un encapsulamiento adecuado debido a que funciona como una caja negra, esto es, el componente que quiera trabajar con las prestaciones del IGO sólo debe acceder a su interfaz de comunicación, sin tener que saber lo que realiza el OGI internamente. **Interoperabilidad:** Luego del rediseño, y gracias a su interfaz de comunicación, el OGI permite la composición y encapsulamiento. Estas características le permiten al IGO tanto trabajar de forma independiente como en conjunto a uno o más componentes que requieran de sus servicios para lograr una aplicación de mayor complejidad. **Multiplataforma:** Al estar basado en Flash, el OGI puede ser incluido en cualquier navegador que tenga el plug-in de Flash Player, sea este Windows, Linux o Mac. **Auto-contenido:** El OGI presenta un muy bajo acoplamiento interno, ya que las capas son controladas por el mediador, generándose de esta forma una alta cohesión funcional [22].

9. Actualización y mejora de las capas

El proceso de actualización/mejora en la versión anterior del OGI era complejo, esto porque era necesario revisar el impacto de agregar un nuevo elemento sobre los demás ya incluidos, originado por el alto grado de acoplamiento presente en la vista. El desarrollo basado en componentes y principalmente el patrón de diseño mediador, permitieron el desacoplamiento de los elementos presentes en la vista a capas de funcionalidades comunes. Esto permite que, al realizar cambios en una capa en particular, sólo debe manejarse dicha capa sin tener que revisar el efecto en las demás. El peor caso consiste en la inclusión de nuevas características para la capa, haciendo que el mediador deba ser modificado, sin tener que alterar las demás capas presentes (Fig. 16).

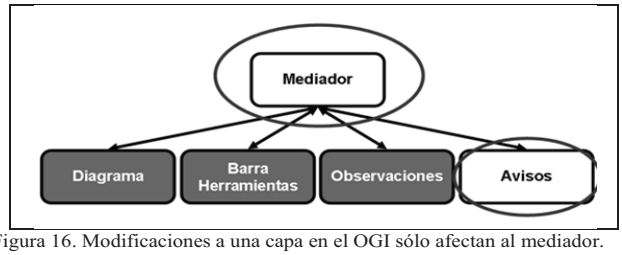

Figura 16. Modificaciones a una capa en el OGI sólo afectan al mediador.

Para el caso de incluir una nueva capa en el componente, el mediador debe ser modificado para que reconozca los servicios que ofrece dicha capa. Se deben incluir los llamados a las funcionalidades que gestionará el mediador hacia la nueva capa (Fig. 17).

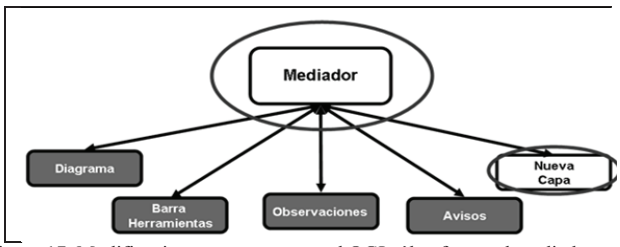

Figura 17. Modificaciones a una capa en el OGI sólo afectan al mediador.

V. CONCLUSIONES

De los resultados obtenidos es posible concluir que el desarrollo de software orientado a componentes presenta ventajas importantes para el desarrollo de software educativo. Los componentes presentan características que los hacen reutilizables en diversos tipos de aplicaciones, junto con la posibilidad de realizar cambios de forma rápida y efectiva. Otro factor importante fue la incorporación de un patrón de

diseño que aporta una estructura ordenada para el nuevo componente desarrollado. Dicha estructura, además de orden, entrega posibilidades de abordar nuevos requerimientos de forma rápida y efectiva. Por ejemplo, basta con cambiar la capa del diagrama del OGI para cambiarlo por otro diagrama completamente distinto y, por ende, un nuevo OGI, pero con las mismas funcionalidades aportadas por las demás capas. Las nuevas características en rediseño permitieron generar una comunicación clara y ordenada entre el OGI y otras aplicaciones que lo contendrán.

AGRADECIMIENTOS

A la Comisión Nacional de Ciencia y Tecnología (CONICYT) de Chile, en particular, al Fondo de Fomento al Desarrollo Científico y Tecnológico (FONDEF) quienes financiaron en parte el desarrollo de esta investigación a través de los proyectos TE04i1005 y D08i1010.

REFERENCIAS

[1] D. Hyerle, Visual tools for constructing knowledge, Association for Supervision and Curriculum Development, Virginia, 1996.
[2] R. Marzano, D. Pickering y J. Pollock, Classroom instruction that works: research based strategies for increasing student achievement, Association for Supervision and Curriculum Development, Virginia, 2001.
[3] B. Beyer, Improving student thinking: a comprehensive approach, Allyn and Bacon, Boston, 1997.
[4] C. Griffin y B. Tulbert, The effect of graphic organizers on students' comprehension and recall of expository text: A review of the research and implications for practice. Reading and Writing Quarterly: Overcoming Learning Difficulties, vol. 11, no. 1, pp. 73-89, 1995.
[5] N. Gallavan y E. Kottler, Eight types of graphic organizers for empowering social studies students and teachers, The Social Studies, May/June, 117-128, 2007.
[6] D. Mitchell y C. Hutchinson, Using graphic organizers to develop the cognitive domain in physical education. Journal of Physical Education, Recreation & Dance, vol. 74 no. 9, pp. 42-47, 2003.
[7] S. Parks y B. Howard, Organizing thinking: graphic organizers, Critical Thinking Press & Software, Pacific Grove, CA, 1990.
[8] H. Ponce, M. López y J. Labra, Organizadores Gráficos Interactivos. En Farias, M y Oblinovic, K. (eds.), Aprendizaje Multimodal-Multimodal Learning, PUBLIFAHU-USACH, Santiago de Chile, pp. 183-190, 2008.
[9] A. Stull y R. Mayer, Learning by doing versus learning by viewing: three experimental comparisons of learner-generated versus author-provided graphic organizers. Journal of Educational Psychology, vol. 99, no. 4, pp. 808–820, 2007.
[10] N. Strangman, T. Hall y A. Mayer, Graphic organizers and implications for universal design for learning: Curriculum Enhancement Report, US National Center on Accessing the General Curriculum, 2004, Recuperado el 15 de marzo de 2009, de: http://www.k8accesscenter.org.
[11] N. Witherell y M. McMackin, Teaching writing through differentiated instruction with leveled graphic organizers, Scholastic, New York, 2005.
[12] L. Campbell, B. Campbell y D. Dickinson, Inteligencias múltiples: usos prácticos para la enseñanza-aprendizaje, Editorial Troquel, Buenos Aires, 2000.
[13] M. López, H. Ponce, J. Labra y H. Jara, Organizadores gráficos interactivos: add-in para MS PowerPoint, XIII Taller Internacional de Software Educativo, TISE. Diciembre 2, 3 y 4. Santiago, Chile, 2008.
[14] H. Ponce, M. López y J., Labra, Programa de Formación en Estrategias de Aprendizaje Lector, En J. Sánchez (Ed.), Nuevas ideas en informática educativa, LOM Ediciones, Santiago de Chile, vol. 3, pp. 193-216, 2007.
[15] H. Ponce, M. López, J. Labra, J. Brugerolles, y C. Tirado, Evaluación experimental de un programa virtual de entrenamiento en lectura significativa (e-PELS). Revista Electrónica de Investigación Psicoeducativa, vol. 5, no. 2, pp. 399-432, 2007
[16] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King y S. Angel, A pattern language: towns, buildings, construction, Oxford University Press, New York, 1977.
[17] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Design patterns: elements of reusable object-oriented software, Addison-Wesley Professional, 1995.
[18] C. Szyperski, Component software beyond Object–Oriented Programming, Addison–Wesley, Edinburgh Gate, 1998.
[19] J. Zeyu, H. Tsao y Y. Wu, Testing and quality for component-based software, Artech House Library, Boston, 2003.
[20] J. Montilva, N. Arapé y J. Colmenares, Desarrollo de software basado en componentes. Actas del IV Congreso de Automatización y Control, Mérida, Venezuela, 2003.
[21] I. Crnkovic, M. Larsonn, Building reliable component-based software systems, Artech House, Boston, 2002.
[22] J. Bieman y L.M. Ott, Measuring functional cohesion, IEEE Transactions on Software Engineering, vol. 20, no. 8, pp. 644-657, 1994.

Francisco A. Almarza es graduado en Ingeniería Civil en Informática de la Universidad de Santiago de Chile (USACH), y se desempeña como ingeniero de software en VirtuaLab-USACH, laboratorio multidisciplinario de investigación, desarrollo y transferencia de tecnologías visuales. Su área de investigación se centra en el desarrollo de componentes de software y la utilización de patrones de diseño para optimizar el desarrollo de software.

Héctor R. Ponce es graduado en Ingeniería Civil en Informática de la Universidad de Santiago de Chile (USACH), además posee un PhD por la Universidad de Lincoln, Inglaterra. Es profesor asociado del área de sistemas de información en el Departamento de Contabilidad y Auditoría, Facultad de Administración y Economía de la USACH. Es director científico de VirtuaLab-USACH, laboratorio multidisciplinario de investigación, desarrollo y transferencia de tecnologías visuales. Su área de investigación se centra en el desarrollo de componentes de software que implementan estrategias visuales y la evaluación de su impacto en el aprendizaje.

Mario J. López, Ph.D. es profesor asociado en el área de tecnologías de información y comunicaciones en el Departamento de Ingeniería Industrial, Universidad de Santiago de Chile (USACH). Es director general de VirtuaLab-USACH, laboratorio multidisciplinario de investigación, desarrollo y transferencia de tecnologías visuales. Ha sido director de diversos proyectos de investigación y desarrollo con financiamiento público y privado; posee numerosas publicaciones en revistas, capítulos de libros y congresos internacionales.