Desarrollo de software educativo basado en componentes: el caso de e-pels

Mauricio G. Gallardo Ch.
VirtuaLab-USACH
Universidad de Santiago de Chile
Av. L.B. O'Higgins 3363
Santiago, Chile
Tel: + 56 2 718 03 22
mauricio.gallardo@usach.cl

Héctor R. Ponce A.
VirtuaLab-USACH
Universidad de Santiago de Chile
Av. L.B. O'Higgins 3363
Santiago, Chile
Tel: + 56 2 718 03 12
hector.ponce@usach.cl

Mario J. López V.
VirtuaLab-USACH
Universidad de Santiago de Chile
Av. L.B. O'Higgins 3363
Santiago, Chile
Tel: + 56 2 718 03 11
mario.lopez@usach.cl

ABSTRACT

This article presents the development of an educational application using a software component-based architecture, highlighting its advantages. The case analyzed is the transformation of e-PELS, a formation Program in Reading Strategies derived from a pedagogic methodology centered in information processing and cognitive skills development. The software that supports this program is a fundamental part in its pedagogic implementation. A relevant problem that this application presents is its monolithic conception, which does not permit an accelerated modification or the incorporation of new functionalities. The transformation of e-PELS consisted of going from a monolithic application to software component-based application. For this, the development process is described, the re-utilization and adaptation of components which were integrated through a successive series of versions using a composition mechanism. Finally, the advantages of software component-based development are discussed, and the disadvantages detected of this type of architecture are indicated.

RESUMEN

Este artículo presenta la transformación de un software educativo utilizando una arquitectura de componentes de *software,* resaltando las ventajas de utilizar esta arquitectura. El caso analizado es la transformación de e-PELS, un *P*rograma de formación en *E*strategias *L*ectora*S* (PELS) derivado de una metodología pedagógica centrada en el procesamiento de información y desarrollo de habilidades cognitivas. El software que apoya el programa PELS, es una pieza fundamental en la implementación pedagógica del programa de formación en estrategias lectoras. Un problema fundamental que presenta dicha aplicación es su concepción monolítica, que no permite una modificación acelerada ni la incorporación de nueva funcionalidades. La transformación de dicha aplicación consistió en pasar de una aplicación monolítica a una aplicación basada en componentes de software. Para ello, se describe el proceso de desarrollo, la reutilización y adaptación de componentes que fueron integradas a través de una serie sucesiva de versiones mediante los mecanismos de composición. Finalmente, se discuten las ventajas del desarrollo por componentes, y las desventajas detectadas de este tipo de arquitectura.

KEYWORDS

Componentes de *software,* mecanismos de composición de componentes, interfaces de componentes, e-PELS.

INTRODUCCIÓN

La variedad y complejidad de los requerimientos educativos impone sobre el desarrollo de software una serie de demandas que permitan atenderlos de manera eficiente, con calidad y con una relación costo-beneficio razonable para el desarrollo de la industria y un precio justo para el sector educativo. Sin embargo, la realidad se caracteriza por una situación catalogada como *crisis* para el desarrollo de *software* educacional, caracterizada principalmente por [1]:

- Excesivo coste de desarrollo, tanto en recursos como en tiempo que es difícil de estimar previamente.
- Deficiente calidad en el proceso de desarrollo así como del producto final.
- Falta de capacidad para adaptarse a requerimientos cambiantes, de forma rápida y eficiente.

Para resolver en parte esta problemática se ha planteado la necesidad de modificar el proceso de desarrollo de *software* encaminándolo hacia una mayor reutilización composicional; adoptando para ello un modelo de desarrollo de software basado en componentes [2].

En este artículo se presenta el caso de un software educativo denominado e-PELS que presentaba dificultades para adaptarse a nuevo requerimiento educativos. Dado el problema, se señala el proceso de transformación utilizando una arquitectura basada en componentes de software; indicando sus principales ventajas y desventajas detectadas en el proceso.

PROBLEMA

El caso analizado es la transformación de e-PELS, un *P*rograma de formación en *E*strategias *L*ectora*S* (PELS)

Gallardo, M., Ponce, H., López, M. (2009). Desarrollo de Software Educativo basado en componentes: el caso de e-PELS. . En J. Sánchez (Ed.): Nuevas Ideas en Informática Educativa, Volumen 5, pp. 108 – 117, Santiago de Chile.

derivado de una metodología pedagógica centrada en el procesamiento de información y desarrollo de habilidades cognitivas. El software que apoya el programa PELS, es una pieza fundamental en la implementación pedagógica del programa de formación en estrategias lectoras [3].

e-PELS implementa una serie de micro estrategias de aprendizaje independientes entre sí, tales como: subrayado de párrafos, coloreado, elaboración de parafraseo, identificación de estructura subyacente, formulación de auto preguntas, organizadores gráficos y elaboración de resúmenes (Figura 1).

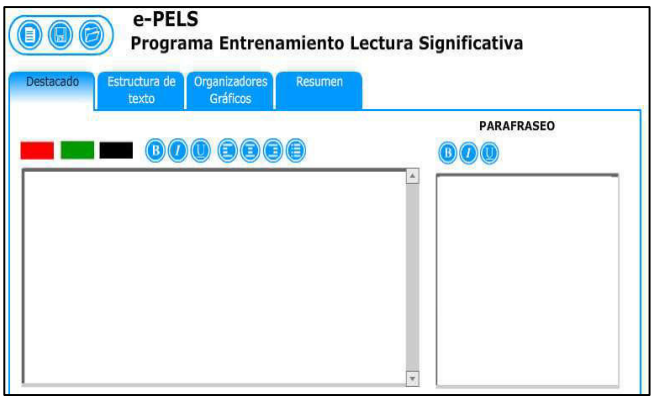

FIGURA 1: Versión original de e-PELS

Las funcionalidades de e-PELS en término de estrategias de aprendizaje corresponden a [4]:

Destacado: Subrayado y Coloreado:

e-PELS implementa la función a través de un editor de textos para el subrayado de párrafos y el coloreado de párrafos, utilizando colores con significado específico (ejemplo: rojo para conceptos claves y verde para ilustraciones de dichos conceptos) que permiten destacar ideas que el estudiante considere relevantes.

Parafraseo

e-PELS incluye un editor para parafrasear las ideas principales y secundarias destacadas.

Estructura Subyacente y autopreguntas

e-PELS implementa una función (integrada al de autopreguntas) para identificar explícitamente la estructura del texto; y además, incorpora una conjunto de diagramas que facilitan al estudiante la identificación de la estructura subyacente el texto bajo análisis.

Autopreguntas

e-PELS integra las estrategias estructura subyacente y autopreguntas para facilitar la identificación de la estructura del texto y de sus elementos constituyentes. Las preguntas que se incluyen deberían permitir al alumno reflexionar acerca de lo leído de una forma más estructurada.

Organizadores Gráficos Interactivos

e-PELS incorpora un conjunto de organizadores gráficos interactivos que facilitan organizar ideas o argumentos tratados en los textos bajo análisis, identificar similitudes y diferencias, establecer relaciones causa-efecto, hacer comparaciones, identificar secuencias, entre otros.

Resumen

La aplicación de software incorpora un editor especialmente diseñado para escribir el resumen. Corresponde a una estrategia de recuperación de información en donde el aprendiz lector deberá construir una síntesis acerca del texto fuente.

La efectividad de la versión original de e-PELS ha sido evaluada y probada con alumnos de 4to año de enseñanza básica que presentaban problemas de comprensión lectora, los cuales al emplear la herramienta mejoraron significativamente su capacidad de comprensión lectora en comparación a alumnos que no participaron de la utilización del programa [3].

A pesar de que e-PELS ha demostrado su efectividad, nuevos usos de e-PELS en contextos educativos concretos requieren de diversas adecuaciones, derivadas tanto de problemas de usabilidad, funcionalidades adicionales requeridas como también de la incorporación de nuevas estrategias de aprendizaje a las ya implementadas, como por ejemplo, incluir una opción para trabajar palabras individuales del texto fuente (*caja de palabras* en la nueva versión de e-PELS).

Sin embargo, dicha adecuación enfrenta un problema fundamental: e-PELS fue desarrollado bajo una concepción *monolítica de programación*, que no permite una modificación acelerada ni la incorporación de nueva funcionalidades requeridas durante su implementación en diversas situaciones educacionales. La transformación de dicha aplicación consistió en pasar de una aplicación monolítica a una aplicación basada en componentes de software.

Por lo tanto, bajo dicha concepción monolítica, e-PELS no puede ser adaptarlo a requerimientos cambiantes de forma rápida y eficiente. Esto se debe principalmente a que *e-PELS* se encuentra desarrollado en un único módulo; concentrando todas las clases visuales en un archivo editable llamado *EPELS.fla* (Figura 2). Los problemas principales que presenta son:

- El módulo principal de *e-PELS* accede directamente a los atributos de las estrategias lectoras, las cuales no ocultan sus detalles de implementación. Esto produce que los cambios en la estructura de las estrategias lectoras impliquen cambios en la definición del módulo principal.
- La composición de *e-PELS* no ocurre en tiempo de ejecución debido a que las instancias de los módulos son definidas en tiempo de compilación (Figura 2). Esto

obliga a realizar una intervención a nivel de código fuente para incorporar una estrategia lectora.

- El módulo principal de *e-PELS* no provee servicios mediante una interfaz de componentes, esto imposibilita que *e-PELS* pueda interoperar en otros entornos de trabajo.

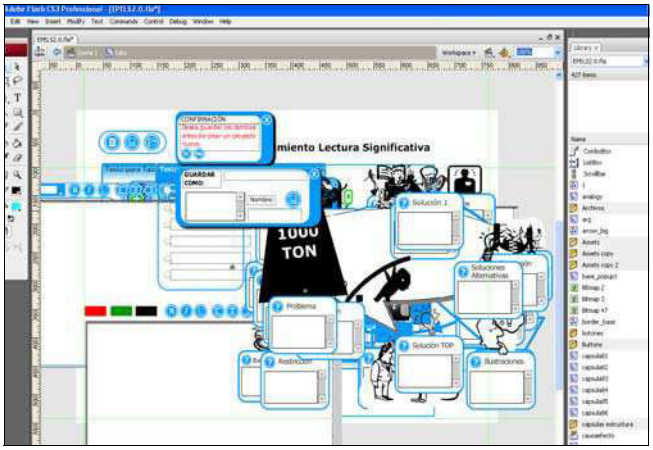

FIGURA 2: Versión monolítica de e-PELS

Para resolver los problemas que presenta la versión monolítica de e-PELS y permitir su adaptación a nuevos requerimientos educativos, se decidió desarrollar una nueva versión utilizando una arquitectura basad en componentes de software.

MARCO CONCEPTUAL

A continuación se presentan los principales aspectos teóricos de las componentes de *software*, que fueron considerados en el desarrollo de una nueva versión de *e-PELS* basada en componentes de *software*.

Definición

Aunque existe una variedad de definiciones respecto a lo que es una componente de *software*, existe cierto consenso para definirlo como un conjunto de objetos que cumplen una función específica, especifican interfaces y funcionan en forma independiente. Los componentes de software pueden estar compuestos por otros componentes y ser utilizadas para crear sistemas de software de mayor complejidad [5,6]. Sus características principales se resumen a continuación [7]:

- **Composición:** Es la capacidad que permite integrar componentes para conformar una componente de mayor granularidad. Para ello, las interacciones deben ocurrir a nivel de interfaces de componente.
- **Encapsulamiento:** La componente debe ser capaz de ocultar detalles de su implementación y funcionar como una verdadera caja negra.
- **Interoperabilidad:** La componente es interoperable cuando puede trabajar de forma independiente y puede interactuar con otras componentes para implementar una funcionalidad de mayor complejidad.
- **Multiplataforma:** Para favorecer la reutilización, es deseable que la componente pueda trabajar independiente del *hardware* y del sistema operativo.
- **Auto-contenido:** La componente es auto contenida cuando depende lo menos posible de otras para cumplir con su propósito**.** Para ello, la componente debe presentar un bajo acoplamiento y una alta cohesión.

Interfaces de componentes

Un aspecto esencial en la construcción de componentes de software son las interfaces de componentes (API). Estos son el mecanismo que permite la comunicación e interconexión de componentes que permiten su interoperabilidad [7]. Para ello, las interfaces de componentes presentan una "interfaz proporciona", que declara un conjunto de servicios que la componente implementa, una "interfaz requiere" que especifica los servicios requeridos para que la componente pueda operar, declarando un conjunto de eventos que la componente puede emitir (Figura 3).

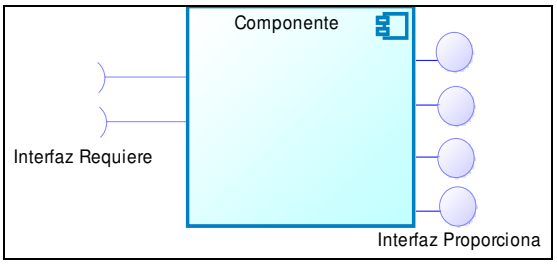

FIGURA 3: Interfaz de componentes

Los eventos permiten comunicar asincronamente una respuesta frente a un estímulo externo o un cambio en el estado interno de la componente. En la interfaz de la componente se especifica la signatura del evento, junto a la condición que gatilla su emisión. Hay que indicar que no se debe explicitar las entidades que consumirán dicho evento.

Composición de componentes

De acuerdo a *Sametinger* la composición de componentes consiste en el proceso de construir aplicaciones mediante la interconexión de componentes de *software* a través de sus interfaces [8]. Visto de manera práctica, este proceso puede entenderse como una relación cliente-servidor. El componente cliente solicita un servicio que se encuentra ofrecido dentro de las operaciones definidas en la interfaz del componente servidor. Luego, el componente servidor ejecuta la operación requerida y devuelve los resultados al cliente de manera síncrona. Desde el punto de vista de la interacción entre las interfaces de las componentes se puede distinguir tres tipos de composición de componentes.

- **Composición secuencial:** Se presenta cuando existe incompatibilidades entre las interfaces de componentes a interconectar. Para ello, se requiere de una componente Adaptador que permita reconciliar dichas interfaces.

- **Composición jerárquica:** Se presenta cuando una componente realiza una solicitud directamente a los servicios proporcionados por otra componente, no existiendo incompatibilidades entre ellas.
- **Composición aditiva:** Ocurre cuando las interfaces de dos o más componentes se unen para crear una nueva componente de mayor granularidad.

DESARROLLO BASADO EN COMPONENTES DE SOFTWARE

OMT++ es una de las metodología de análisis y diseño orientada a objeto, más maduras, eficientes y ampliamente utilizada en la actualidad para el desarrollo de proyectos de *software*. Para aprovechar los beneficios de esta metodología y las ofrecidas por las componentes de *software,* los autores *Laitkorpi y Jaaksi* presentan una modificación de la metodología OMT++ para formalizar la incorporación del desarrollo de componentes de *software* en el proceso de desarrollo de *software* [9]. A continuación, se detalla las principales fases, de manera secuencial, para el proceso OMT++ modificado (Figura 4).

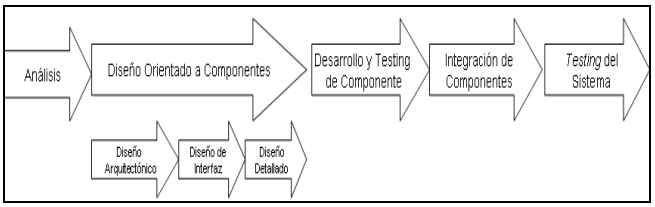

FIGURA 4: Proceso de desarrollo orientado a componentes

Este proceso presenta modificaciones con respecto al desarrollo de *software* tradicional principalmente para las fases de Análisis, Diseño y Desarrollo.

En la fase de Análisis se introducen las actividades de búsqueda, selección y adaptación de componentes existentes para su posterior reutilización en el desarrollo orientado a componentes.

En la fase de Diseño Orientado a Componentes, se introduce una nueva fase denominada Diseño de Interfaces en la cual se clarifica la existencia de interfaces, se identifica las interfaces que pueden ser reutilizadas y se descubren nuevas interfaces y operaciones.

La fase de Desarrollo queda dividida en los procesos de: Desarrollo de componentes e Integración de componentes. El proceso de desarrollo de componentes se realiza en caso de existir componentes que no sean aptas para ser reutilizadas y requerimientos que no han podido ser satisfechos por el repositorio de componentes.

El proceso de integración de componentes se realiza a partir de un conjunto de componentes tratando siempre de maximizar su reutilización y reducir así el número de componentes que requieran ser desarrolladas desde un inicio. Para lograr esto, se debe disponer de repositorios de componentes que sean reutilizables, confiables y que actúen de acuerdo a sus especificaciones. Esta integración se realiza considerando las interfaces de componentes y los métodos de composición ya descritos.

DESARROLLO DE E-PELS

En este artículo se exponen los productos de las fases más relevantes del proceso de desarrollo de *e-PELS* orientado a componentes de *software*.

Durante el desarrollo de la nueva versión de e-PELS, realizaron varios *focus group* con alrededor de 65 profesores de enseñanza básica, a quienes se les presentó la versión actual de *e-PELS*. Esto, con el objetivo de establecer las estrategias lectoras precisas para el nivel 2 (NB2) a ser implementadas en la nueva versión. A continuación, se listan los acuerdos establecidos en los *focus group* (tabla 1).

Estrategia Lectora	Habilidad deseada	Disponible en e-PELS actual
Destacado y Parafraseo	Selección de información relevante explícita	Si
Tipo de Texto	Reconocimiento de la estructura subyacente para texto acorde al nivel NB2	No
Organizadores Gráficos Interactivos	Organización gráfica de la información	Parcialmente
Resumen	Síntesis y Compresión del texto	Si
Caja de Palabras	Incorporación de vocabulario	No

Tabla 1: Requerimientos de estrategias

Además, se establecieron los siguientes cambios a e-PELS:

- Los tipos textuales "Expositivo" y "Conversacional" no pertenecen a la estrategia de tipología textual para NB2, por lo que no deberán ser implementados en la nueva versión de e-PELS.
- La nueva versión trabajará solamente con las tipologías textuales y no con las micro-estructuras, por lo que los Organizadores Gráficos de Definiciones y Problema-Solución no serán considerados dentro de la estrategia tipo de texto.
- La estrategia "Tipo de Texto" deberá incorporar las siguientes tipologías textuales: Narrativo, Dramático, Poético, Noticioso e Informativo de acuerdo a las exigencias impuestas por el Ministerio de Educación.

En los focus group se acordó el desarrollo de la estrategia “Caja de Palabras”, que es transversal a las demás estrategias lectoras y que no se encuentra implementada en la versión actual de e-PELS. Esta estrategia permite al estudiante definir las palabras desconocidas, a medida que va procesando el texto. Para ello, la Caja de Palabras deberá interactuar con las estrategias "Destacado y Parafraseo" y "Resumen", con el objetivo de que éstas puedan suministrar las palabras a definir. Los profesores señalaron que el Organizador Gráfico "Definiciones" es indicado para que el alumno pueda desarrollar definiciones de las palabras desconocidas

El proceso involucró el desarrollo, reutilización y adaptación de componentes que fueron integradas a través de una serie sucesiva de versiones mediante los mecanismos de composición. Se realizaron pruebas de funcionalidad y usabilidad con cerca de 1000 alumnos de distintos colegios.

Diseño arquitectural de e-PELS

Para obtener las componentes y las interfaces de componentes se consideraron los principios propuestos por los autores *Laitkorpi y Jaaksi* [9], que permiten deducir del diagrama de clases (Figura 5) su correspondiente diagrama de componentes (Figura 6).

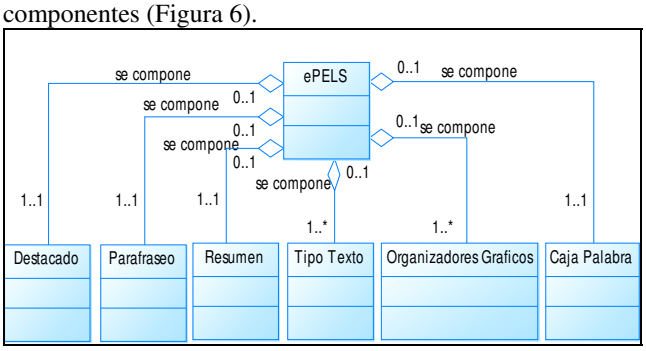

FIGURA 5: Diagrama de clases de e-PELS

Una vez que el diagrama de clases fue implementado, se procedió a establecer el conjunto de componentes que servirían como los bloques estructurales de la nueva versión de e-PELS (ver figura 6).

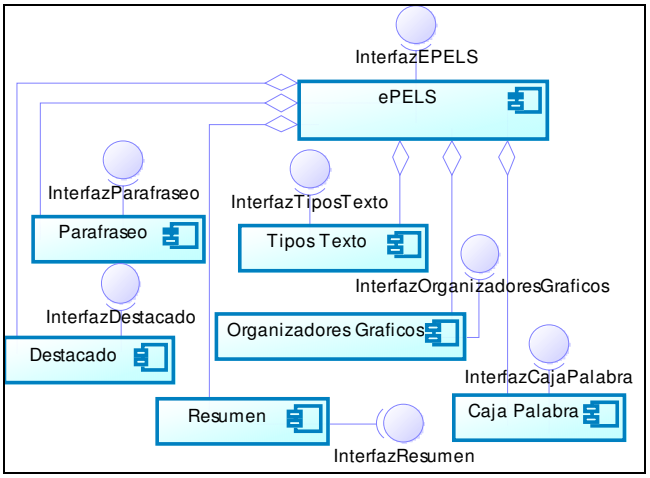

FIGURA 6: Diagrama de componentes inicial de e-PELS

Durante el proceso de desarrollo orientado a componentes, el diagrama de componentes inicial sufrió las siguientes modificaciones:

- Como las estrategias de Destacado, Parafraseo y Resumen quedan implementadas a través de un Editor de Texto, las componentes de Destacado, Parafraseo y Resumen serán implementadas por la componente Editor de Texto.
- Como las componentes TiposTexto y Organizadores Gráficos tienen el propósito general de despliegue e implementación de sub-estrategias lectoras, se decidió que el despliegue de las sub-estrategias lo realizará una componente especial denominada GestorEstrategia y la implementación de las sub-estrategias quedará a cargo de las componentes Tipo Texto y los OGIs 3.0 a reutilizar.
- Se introduce la componente Adaptador que permite reconciliar la interfaz de las componentes OGIs 3.0 para que puedan ser reutilizados por la componente GestorEstrategia.
- La componente Caja Palabra utilizará la componente TipoTexto para poder definir las palabras ingresadas.

Estas modificaciones permitieron obtener el siguiente diagrama de componentes final de *e-PELS* (Figura 7).

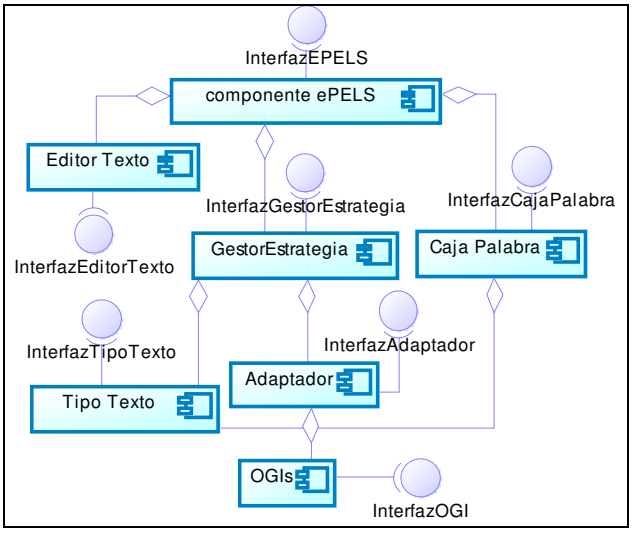

FIGURA 7: Diagrama de componentes final de e-PELS

Búsqueda y selección de componentes

Para el desarrollo de la nueva versión de e-PELS, el equipo de desarrollo contaba con un repositorio de componentes de software desarrollado y otros en desarrollo. Entre los componentes disponibles, se encontraban los Organizadores Gráficos Interactivos (OGIs) [10,11], versión 3.0. Las características que presentan los OGIs 3.0 son las siguientes:

- Presentan servicios y eventos definidos mediante una interfaz de componente. Esta característica los hacen aptos para ser reutilizados dentro de *e-PELS*.
- Se encuentran desarrollados en lenguaje *ActionScript3* al igual que *e-PELS*. Esta característica facilita la integración de los OGI dentro de *e-PELS*.
- Poseen una arquitectura basada en el modelo MVC++
- Emplean formato XML para guardar las sesiones al igual que *e-PELS*. Esta característica facilita la definición de las sesiones dentro de *e-PELS*.

Adaptación de componentes

Como los OGIs 3.0 presentaban incompatibilidades de operación, incompleta y de parámetro; se hizo necesario desarrollar una componente Adaptador que permitió reconciliar dichas incompatibilidades

La incompatibilidad incompleta se hizo presente ya que los servicios *getTag* y *start* no eran suministrados por medio de la interfaz de componente que definen los OGIs 3.0. Para resolver esta incompatibilidad el Adaptador implementó estos servicios requeridos, los cuales fueron invocados de forma transparente por la componente GestorEstrategia.

La incompatibilidad de operación se hizo presente ya que existían métodos equivalentes entre los OGIs y *e-PELS* pero que diferían en su nombre. Para resolver esta incompatibilidad el Adaptador invoca el servicio del OGI a reconciliar, para luego pasar el resultado obtenido de esta invocación a la componente GestorEstrategia.

La incompatibilidad de parámetro se hizo presente ya que las sesiones de los OGIs poseían un formato XML que comenzaba con el tag "<IGO>", pero *e-PELS* requiere que las sesiones de los OGIs comiencen con un *tag* único e irrepetible. Para resolver esta incompatibilidad el Adaptador agrega a la sesión del OGI un tag equivalente con el nombre del OGI.

Diseño de interfaces de e-PELS

En base al diseño arquitectural, todas las componentes implementadas y por implementar deberán ser accedidas mediante su interfaz de componentes y ellas estarán obligadas a especificar los siguientes servicios:

- **start (conf:XML): void:** Este servicio permite configurar las componentes contenidas en *e-PELS* mediante una configuración en formato XML que se desglosa a medida que se desciende en la granularidad de la componente.
- **setSesion (sesion: XML): void, getSesion (): XML: reset (): void:** Estos servicios permiten la manipulación de las sesiones de la componentes.
- **getId (): Int, getDescription (): String:** Estos servicios permiten identificar a las componentes en cuanto a su id y descripción
- **READY_COMPONENT, :** Evento que notifica el éxito de la carga de las sub-componentes de una componente
- **FAILURE_COMPONENT:** Evento que notifica el fallo de la carga de las sub-componentes de una componente

Diseño detallado de e-PELS

La *componente e-PELS* es la pieza de *software* fundamental para la implementación de la nueva versión de *e-PELS* debido a que se encarga de la carga, configuración y despliegue de las estrategias: Destacado y Parafraseo; Tipo de Texto; Organizadores Gráficos; Resumen; y Caja de Palabras. Además, esta componente interopera con la plataforma que lo contiene para apoyar a las funcionalidades para abrir, guardar, crear, exportar e imprimir una sesión. A continuación se presenta la modelación interna de la componente *e-PELS*, para ello se utilizará la nomenclatura utilizada en el modelo de componente *JavaBeans* (Figura 8).

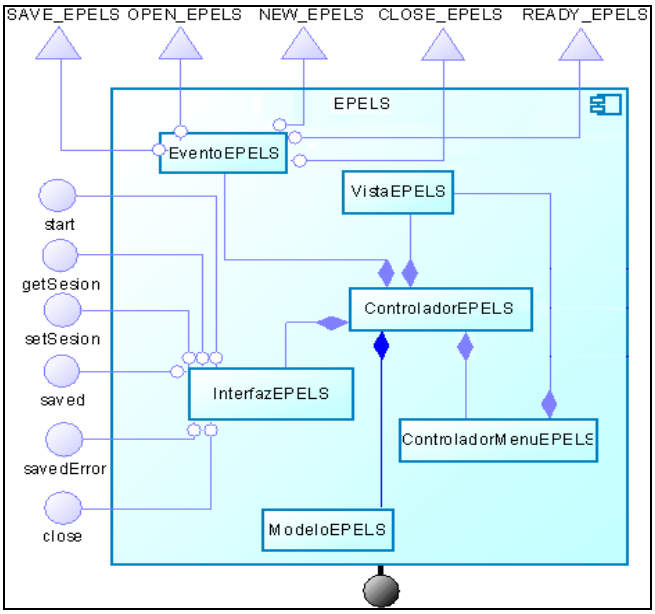

FIGURA 8: Modelación interna de la componente e-PELS

El mecanismo de composición de componentes queda definido por el servicio *start* que se encuentra implementado por la clase "ControladorEPELS". Este servicio recibe como parámetro la configuración en formato XML que detalla aspectos de carga y configuración de las componentes contenidas en *e-PELS* (Figura 9).

```
public function start(conf:XML=""):void{
    this.conf = conf;
    this.cargarComponentesEpels();
}
```

FIGURA 9: Implementación del servicio start

El servicio *start* ejecuta el método *cargarComp* que se encarga de cargar cada uno de las componentes definidas en la configuración XML. Para ello se obtiene desde la configuración XML la ruta de la componente y luego se accede a la clase *modeloEPELS* para realizar la carga de la componente desde el repositorio de componentes (Figura 10).

```
private function cargarComp():void{
  var ruta =   conf.estrategias.estrategia[cont]
               .direccion;
  modeloEPELS.cargarComp(ruta);
}
```

FIGURA 10: Método cargarComp de controladorEPELS

El método *cargarComp* invoca al método *loader* quien se encarga de cargar la componente como un objeto *MovieClip* (Figura 11).

```
private function cargarComp (ruta:String){
        loaderContext= new LoaderContext();
        componenteEpelsLoader = new Loader();
        componenteEpelsLoader.load(new
        URLRequest(ruta,loaderContext);
}
```

FIGURA 11: Método cargarComp de ModeloEPELS

La función *load* realiza la carga de la componente empaquetada dentro de un *MovieClip* en modo asíncrono. Una vez que la máquina virtual JVM realiza la carga de la componente, este emite un evento que es escuchado por la clase *modeloEPELS* mediante la definición del escuchador *initHandler* (Figura 12).

```
private function initHandler(event:Event):void {
        comp  = event.target.content;
        controladorEPELS.setEstrategia(comp);
}
```

FIGURA 12: Método initHandler

Una vez cargada la componente se realiza un llamado al método *setEstrategia* el cual se encarga de colocar dos *listeners* para poder controlar la carga de las componentes. Luego se invoca al servicio *start* de la componente cargada suministrando como parámetro la configuración XML relacionada a la estrategia lectora que ésta implementa (Figura 13).

```
privatefunction setEstrategia (comp:MovieClip):void
{
this.compActual = comp;
comp.addEventListener("READY_COMPONENT",exitocarga)
;
comp.addEventListener("FAILURE_COMPONENT",falloCarg
a);
comp.start(configuracion.estrategias.estrategia[con
t]);
}
```

FIGURA 13: Método setEstrategia

Si las componentes de la estrategia lectora fueron cargadas exitosamente, se activa el escuchador *exitoCarga*, el que se encarga de almacenar la componente en un arreglo de componentes para luego continuar con la carga normal de las siguientes componentes definidas en la configuración XML (Figura 14).

```
public function exitoCarga(e:Event):void{
        arregloComponente.push(compActual);
        continuarCargaComponentes(true);
}
```

FIGURA 14: Método exitoCarga

En caso de existir un error en la carga de la componente de tipo estrategia lectora, se activa el escuchador *falloCarga* el cual se encarga de continuar con la carga de las siguientes componentes (Figura 15).

```
public function falloCarga(e:Event):void{
        continuarCargaComponentes(false);
}
```

FIGURA 15: Método falloCarga

Integración de componentes

La integración de las componentes en *e-PELS* se realiza mediante la composición de componentes que ocurre en tiempo de ejecución (Composición Tardía) [8]. Esta composición es posible gracias a que la componente *e-PELS* se configura mediante un archivo XML el cual define *tags* de tipo "<estrategia>" para definir las estrategias lectoras. Esta configuración permite a *e-PELS*:

- Crear dinámicamente *tabs* en la parte superior para definir el acceso a la estrategia lectora mediante la definición de los *tags* <nombre> y "<descripcion>" (Figura 16).
- Componer en tiempo de ejecución las componentes de *software* que implementan las estrategias lectoras mediante la definición del *tag* "<direccion>", permitiendo así una inserción escalable de estrategias lectoras (Figura 16).

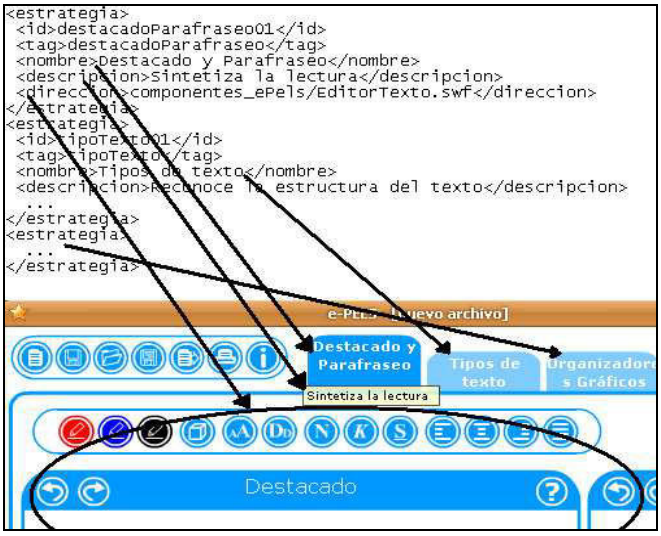

FIGURA 16: Definición de estrategias lectoras en e-PELS

- Componer en tiempo de ejecución las sub-estrategias contenidas dentro de las estrategias lectoras Tipos Texto y Organizadores Gráficos mediante la definición de los tags "<componente>", permitiendo así una inserción escalable de las sub-estrategias lectoras (Figura 17).

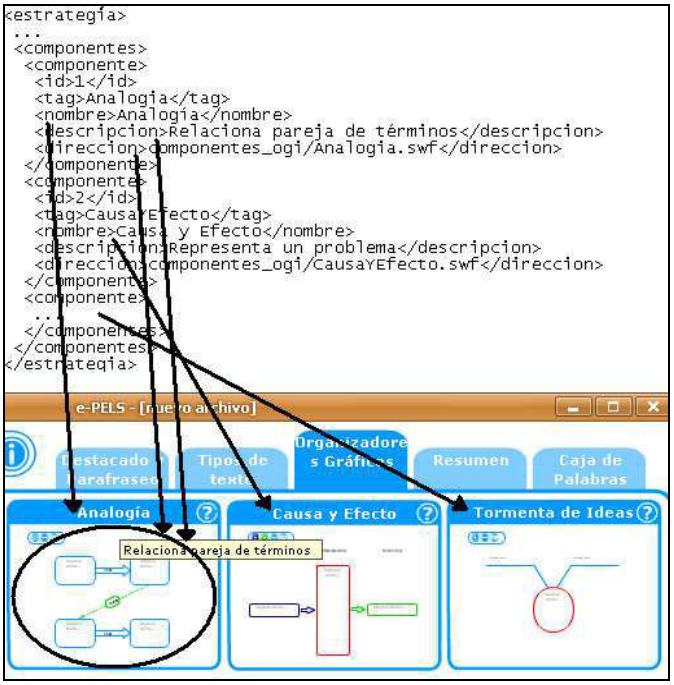

FIGURA 17: Definición de sub-estrategias lectoras en e-PELS

RESULTADOS Y DISCUSIÓN

De los resultados obtenidos se desprende que la nueva versión de *e-PELS* puede ser considerada una componente *basada en componentes de software* debido a que presenta las siguientes características. Analizando estas características existe, se observa una mejora significativa con respecto a la versión anterior.

Composición: Los módulos implementados en la versión antigua de *e-PELS* no son componibles en tiempo de ejecución debido a que los módulos son definidos en tiempo de compilación (Figura 2). En cambio, las componentes implementadas en la versión nueva de *e-PELS* permiten componerse dinámicamente en tiempo de ejecución (Composición tardía) [7] mediante un archivo XML que es suministrado a través del servicio *start.*

Esta composición sumada a la configuración XML suministrada permite incorporar estrategias lectoras solamente realizando una intervención en el archivo de configuración XML. Por ejemplo, si se desea incorporar el OGI Tormenta de Ideas como una estrategia lectora con el objetivo de activar el conocimiento previo que posee el aprendiz acerca la lectura, se tiene que incorporar las siguientes líneas a la configuración XML (Figura 17), obteniendo así la siguiente ejecución (Figura 18).

Esta opción representa una ventaja importante sobre la programación monolítica, ya que no se interviene el código fuente de las componentes involucradas o en la realización de la integración de dichas componentes [5].

```
<estrategia>
 <id>organizadoresGraficos01</id>
 <tag>TormentadeIdeas</tag>
 <nombre>Tormenta de Ideas</nombre>
 <descripcion>Coloca tus Ideas</descripcion>
 <direccion>componentes_epels/GestorEstrategia.swf
 </direccion>
 <componentes>
  <componente>
   <id>1</id>
   <tag>TormentaDeIdeas</tag>
   <nombre>Tormenta de Ideas</nombre>
   <descripcion>Coloca tus Ideas</descripcion>
   <direccion>componentes_ogi/TormentaDeIdeas.swf
   </direccion>
  </componente>
 </componentes>
</estrategia>
```

FIGURA 17: Incorporando la estrategia lectora Tormenta de Ideas

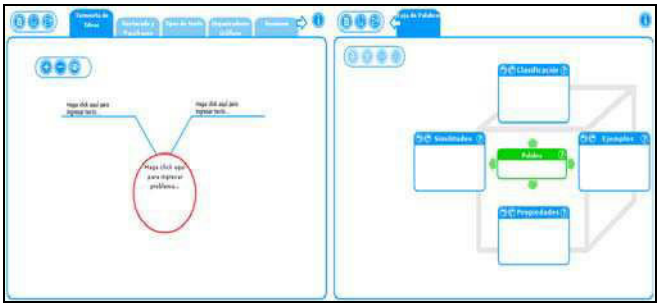

FIGURA 18: Ejecución de e-PELS al incorporar la estrategia lectora Tormenta de Ideas

La desventaja que haber utilizado una composición en tiempo de ejecución es que el aumento en la cantidad de componentes a cargar en *e-PELS* implica un aumento en el tiempo de ejecución de la aplicación. Esto queda constatado al registrar los tiempos de carga por cada incremento en la cantidad de componentes cargadas, prueba que se realizó en un computador con sistema operativo XP de 3.4 GHz. con 1Gb de RAM (Figura 19).

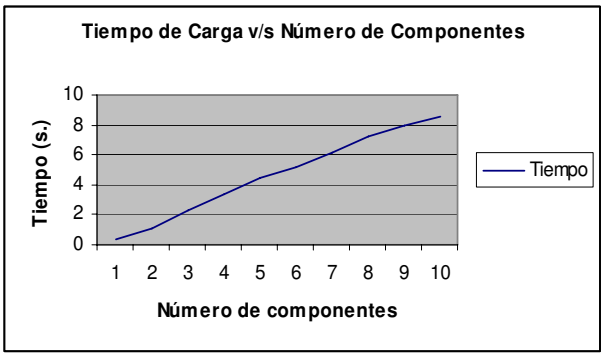

Figura 19: Tiempo de Carga v/s Número de Componentes

Dado los resultados observados en la figura 19, se hace necesario estudiar soluciones de composición de componentes que disminuyan su tiempo de carga. Existen dos opciones, una es la optimización de la carga de la componente

en forma individual, o bien, la administración en la carga de todos los componentes que estructuran la aplicación. Una combinación de ambas opciones también es posible, aunque se requiere su desarrollo e implementación para determinar la mejor opción.

Encapsulamiento: los módulos de la versión antigua de *e-PELS* declaran por defecto sus variables y funciones como públicas, no ocultando su detalle de implementación. En cambio todas las componentes implementadas en la versión nueva de *e-PELS* presentan una definición de interfaz de componente, la cual encapsula sus detalles de implementación del exterior.

Con el desarrollo de *e-PELS*, se demuestran las ventajas del encapsulamiento de las componentes, ya que cada una de ellas puede programarse en forma paralela, visualizándolas como proyectos independientes [7]. Esto debido a que la dependencia entre ellas ocurre a través de sus interfaces de componentes, permitiendo al mismo tiempo el encapsulamiento.

Interoperabilidad: Los módulos implementados en la versión antigua de *e-PELS* no puede interoperar entre sí debido a que éstas no presentan interfaces de componentes que permitan su interacción. En cambio, las componentes en la nueva versión de *e-PELS* interoperan a través de sus interfaces de componentes, lo que permitió, por ejemplo, implementar la opción de enviar palabras desde la componente *editor de texto* a la componente *caja de palabras*.

La interoperabilidad de las componentes permite implementar funcionalidades de mayor complejidad donde se requiere la colaboración entre componentes, no requiriéndose una intervención directa a su implementación [7].

Auto contenido: Las componentes implementadas en la nueva versión de *e-PELS* son auto-contenidas; en cambio, los módulos implementados en la versión antigua no lo son debido a que:

- Las componentes implementadas en la nueva versión de e-PELS presentan **acoplamiento de paso de mensaje** con respecto al acoplamiento de contenido [7], que presenta los módulos de la versión anterior. El acoplamiento de paso de mensaje se presenta cuando las interacciones ocurren a nivel de interfaz de componentes.

- Las componentes implementadas en la nueva versión de *e-PELS* presentan **cohesión de tipo funcional** en comparación con la cohesión procedural [7] que presentan los módulos de la versión anterior. La cohesión funcional se presenta cuando las componentes de *software* presentan un único propósito permitiendo que su reutilización se realiza conociendo únicamente los servicios que provee su interfaz de componentes, obviando así detalles de implementación.

Es importante destacar también, que la nueva versión de *e-PELS* puede ser considerada una componente de *software*, ya que presenta las principales características de las componentes de *software*, a saber:

Composición: La nueva versión de *e-PELS*, a través de la composición, puede participar como componente en la implementación de otra aplicación orientada a componentes, por ejemplo, un software que integre estrategias para distintas disciplinas, utilizando una composición tardía.

Encapsulamiento: El módulo principal de la versión antigua de *e-PELS* declara por defecto sus variables y funciones como públicas, no ocultando su detalle de implementación. En cambio la componente *e-PELS* orientada a componentes presentan una definición de interfaz de componente, la cual encapsula sus detalles de implementación del exterior. Esto permite que se integre de manera trasparente a distintos entornos de trabajo.

Interoperabilidad: La versión antigua de *e-PELS* no puede interoperar con otras componentes debido a que no presenta interfaces de componentes que permitan su interacción con otras componentes de *software*. En cambio, la versión orientada a componentes puede interoperar con otros entornos permitiendo apoyar a la implementación de funcionalidades de mayor complejidad, tales como abrir, guardar y crear una sesión de *e-PELS* (persistencia local).

Multiplataforma: La versión antigua de *e-PELS* puede ser ejecutada únicamente en el sistema operativo *Windows*. En cambio, la versión de *e-PELS* orientada a componentes puede trabajar en diversas plataformas debido a que presenta interfaces de componentes que permiten ser integradas a aplicaciones, tales como *e-PELS Visual Basic*, *e-PELS AIR* y *e-PELS Web,* las cuales pueden ser ejecutadas en los sistemas operativos: *Windows 98, 2000, XP* y *Ubuntu*.

Auto contenido: La nueva versión de *e-PELS* es auto-contenida, ya que permite acoplamiento de paso de mensaje y presenta cohesión de tipo funcional.

CONCLUSIONES

De los resultados obtenidos se puede concluir que el desarrollo orientado a componentes presenta varias ventajas para el desarrollo de software educativo, permitiendo la reutilización de activos de software, ya sea aquellos existentes o bien, en el desarrollo de futuras aplicaciones. Por ejemplo, en el desarrollo de *e-PELS* se reutilizaron los componentes de software denominados *organizadores gráficos interactivos*.

Además, los componentes de *e-PELS* pueden ser reutilizados en otros proyectos, por ejemplo las componentes: *editor de texto*, *caja de palabra y tipo texto*, que sirven para implementar estrategias lectoras para otros niveles educacionales. Esta reutilización es factible debido a que presentan interfaces de componentes, que le permiten actuar como “caja negras”.

Importante resultó también el desarrollo de software en paralelo, permitiendo que distintos equipos concentraran sus esfuerzos en los diversos componentes necesarios para implementar *e-PELS*. Esto también facilita la especialización de los equipos de trabajo, aunque se requiere de mayor coordinación para asegurar el cumplimiento de los requerimientos de la aplicación general. Por ejemplo, el diseño de la interfaz gráfica común para las distintas componentes, o bien, la utilización de estándares del almacenamiento de datos (ejemplo: *e-PELS* utiliza XML), entre otros aspectos.

Por último, es importante destacar que la configuración XML, además de ser un mecanismo que facilita la integración de componentes, permite que un desarrollador pueda entender rápidamente la lógica que emplea la aplicación en la integración de componentes, ya que la jerarquía entre componentes se encuentra descrita en dicho lenguaje estándar.

REFERENCIAS

[1] García E., González J., Pérez M., Pérez-Schofield J. B, Valdés V. (2002) ¿Existe una situación de crisis del *software* educativo? VI Congreso Iberoamericano de Informática Educativa. España: Vigo.

[2] García E., González J., Pérez M., Pérez-Schofield J. B, Valdés V. (2002). Una propuesta para la reutilización de componentes en el proceso de desarrollo de *software* educativo. España: Vigo.

[3] Ponce, H., López, M., Labra, J., Brugerolles, J. y Tirado, C. (2007). Evaluación experimental de un programa virtual de entrenamiento en lectura significativa (E-PELS). Revista de Investigación Psicoeducativa, 5(2), pp. 399-432.

[4] Ponce, H., López, M., Labra, J. (2007). Programa de Formación en Estrategias de Aprendizaje Lector. En J. Sánchez (Ed.) Nuevas Ideas en Informática Educativa, Volumen 3, pp. 193-216, Santiago de Chile: LOM Ediciones.

[5] Szyperski C. (1998). Component Software Component Software Beyond Object–Oriented Programming. Edinburgh Gate: Addison–Wesley.

[6] Broy M., Deimel A., Henn J., Koskimies K., Plasil F., Pomberger G., Pree W., Stal M. & Szyperski C. (1998). What characterizes a (software) component? *Software, Concept and Tools*. 19(1) 49-56.

[7] Crnkovic I. & Larsonn M. (2002). Building reliable component-based software systems. Boston: Artech House.

[8] Sametinger J. (1997). *Software* Engineering with reusable components. Berlin: Springer-Verlag.

[9] Laitkorpi M. & Jaaksi A. (1999). Extending the Object-Oriented Software Process with Component-Oriented Design. The Journal of Object Oriented Programming (JOOP).

[10] Ponce, H., López, M., Labra, J. (2008). Organizadores Gráficos Interactivos. En Farias, M y Oblinovic, K. (eds.) Aprendizaje Multimodal-Multimodal Learning, pp. 183-190. Santiago: PUBLIFAHU-USACH.

[11] López, M, Ponce, H., Labra, J, Jara, H. (2008). Organizadores Gráficos Interactivos: Add-in para MS PowerPoint. XIII Taller Internacional de Software Educativo, TISE. Diciembre 2, 3 y 4. Santiago, Chile.