Técnicas de adaptación de componentes

José Antonio Flores Lara

Dra. Perla Velasco Elizondo

Índice

Índice de Figuras

Centro de Investigación en Matemáticas CIMAT, 2009

Resumen

El desarrollo de software basado en componentes se refiere al proceso de construir programas, mediante el ensamble o composición de un conjunto de elementos preexistentes, denominados componentes.
Dado que los componentes pueden ser provistos por terceras partes, puede darse el caso de que no todos los componentes identificados para la construcción de un sistema cumplan con los requerimientos en su totalidad. En el ámbito de Sistemas Basados en Componentes (SBC), la adaptación es un proceso mediante el cual se resuelven inconsistencias en los componentes antes de proceder a su ensamble.
En este reporte técnico se describe las técnicas encontradas en el proceso de adaptación de componentes.

Palabras Clave

Adaptación de Técnicas, Componente de Software, Desarrollo Basado en Componentes

Dedicatoria

Esta tesis esta dedicada a mi esposa , quien siempre estuvo en los momentos más difíciles apoyándome , dando palabras de aliciente, también a mis dos hijas que son la inspiración para seguir sobresaliendo en mi carrera.

Agradecimientos

- ✓ Quiero agradecer sinceramente a mi asesora Dr. Perla Velasco por su paciencia y consejos.

- ✓ También quiero expresar mi sincero aprecio a todos mis compañeros de la MIS, quienes me apoyaron, en los momentos difíciles.

- ✓ Y por ultimo quiero agradecer a mi institución el Tecnológico Superior Zacatecas Occidente, donde trabajo por todo el apoyo que me dio para realizar esta maestría.

Capítulo 1. Ingeniería de Software Basada en Componentes y el Concepto de Adaptación

La ingeniería de software basada en componentes es un nuevo paradigma del desarrollo apoyado en la idea de construir sistemas a partir de elementos preexistentes, en donde los elementos preexistentes se conocen como componentes.
En este capítulo se explican los conceptos fundamentales del paradigma con el propósito de dar una introducción, para después pasar puntualmente a la descripción de las técnicas de adaptación de componentes.

1.1 ¿Qué es la Ingeniería de Software Basada en Componentes?

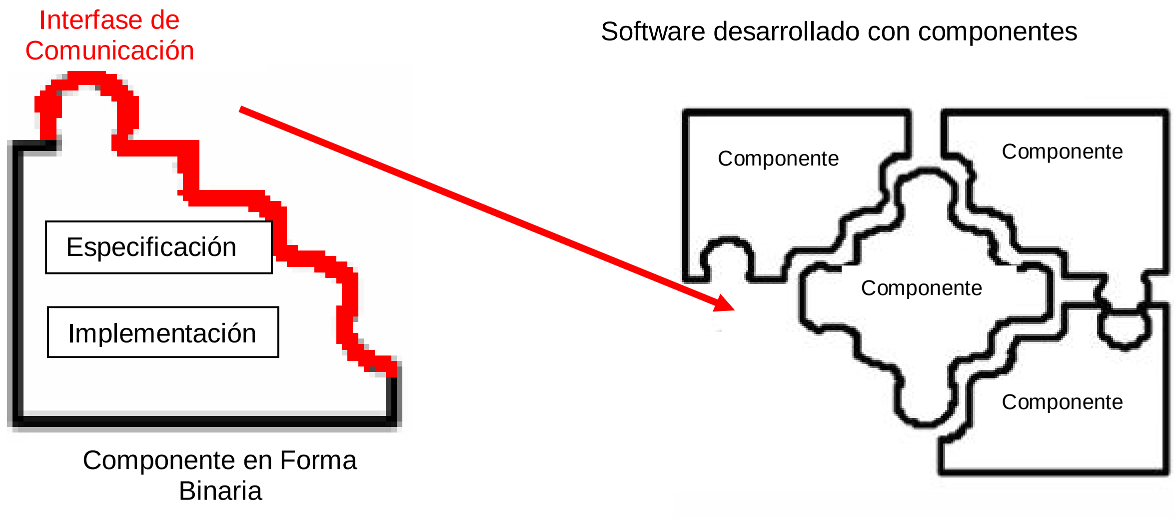
La ingeniería de software basada en componentes es un nuevo paradigma del desarrollo apoyado en la idea de construir sistemas a partir de elementos preexistentes, en donde los elementos preexistentes se conocen como componentes.

1.1.1 ¿Qué es un Componente?

Un componente de software es un una unidad de composición que puede contener múltiples objetos, clases y otros componentes, puede ser desarrolladas por terceros [16].
El término componente de software se podría describir con las siguientes características [7][17]:

- Es independiente y reutilizable, no depende de la aplicación que la utiliza y, se puede emplear en diversas aplicaciones.
- Puede invocar o ser invocado por otro componente de software, esto significa que dos componentes pueden trabajar juntos como un todo ver fig. 1.1.
- Debe de tener una especificación en donde el componente de software debe de ser descrito en un alto nivel de abstracción, estas especificaciones sirven para informar los servicios que provee y espera.

Figura 1.1. Un componente de software



Fuente: Christiansson, , p. 236, 2001

1.2 Ciclo de Vida de Software Basado en Componentes

Como es sabido, el ciclo de vida describe un proceso de desarrollo de software desde su concepción hasta su retiro, en términos de un conjunto de actividades.
Existen dos enfoques para definir un ciclo de vida de software, el secuencial y el evolutivo. El enfoque secuencial define todas las actividades del ciclo de vida seguidas una tras de otra, en donde se tiene que completar una actividad para continuar la siguiente. Algunos ejemplos de este enfoque son el de cascada o el modelo en "V" [10].

El enfoque evolutivo permite un conjunto de actividades en paralelo sin tener un estricto cumplimiento del termino de una actividad para que se empiece la siguiente. Algunos ejemplos de este enfoque son el modelo iterativo y el modelo en espiral.

Independientemente del tipo de modelo utilizado se pueden identificar un conjunto de actividades comunes [9]:

- Especificación de Requerimientos: En esta actividad, se establecen los servicios, restricciones y las metas del sistema. Es decir, una especificación de lo qué el sistema se supone deba de hacer.
- Diseño de Software: Actividad en la cual se aplican distintas técnicas y principios con el propósito de definir un producto de software con el suficiente detalle para permitir su realización.
- Implementación del Sistema: En esta actividad se realiza la traducción del diseño a un lenguaje de programación.

- Pruebas de Software: El objetivo de esta actividad es en donde se tiene que comprobar que el software cumpla con los requerimientos establecidos.
- Mantenimiento: El propósito de esta actividad es mantener y mejorar el software para enfrentar errores descubiertos y nuevos requerimientos.

Sin embargo la construcción de software basado en componentes se diferencia de la construcción de sistemas tradicionales en que, en su ciclo de vida se consideran dos procesos:

a) Proceso de desarrollo para la reutilización.
b) Proceso de desarrollo con reutilización.

En la siguiente sección se explicaran cada uno de ellos ver fig. 1.2.

1.2.1 Proceso de Desarrollo para la Reutilización

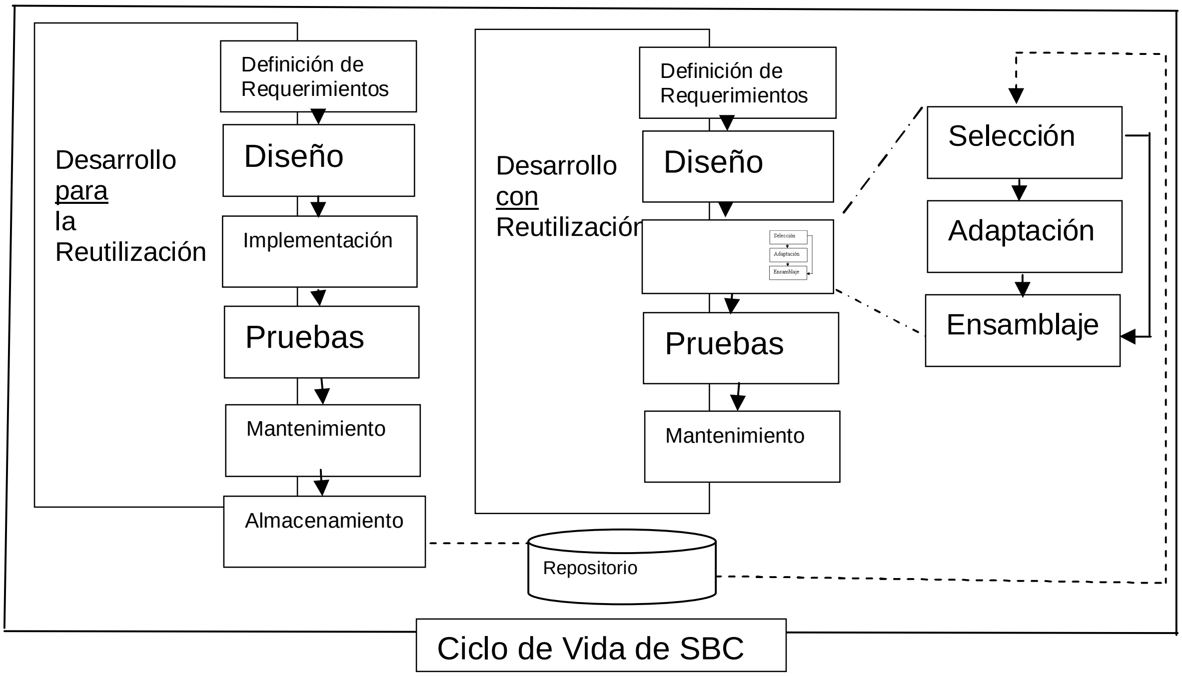
El desarrollo para reutilización es el proceso mediante el cual se incluyen a todas las actividades necesarias para crear un componente de software, que será reutilizado por el creador(es) o por terceros en el proceso b).

En este tipo de proceso de desarrollo de software existen todas las actividades de desarrollo de un ciclo de vida que anteriormente se mencionaron. Sin embargo en el desarrollo para el reutilización se incorpora una actividad de almacenamiento, que describe el echo de almacenar los componentes producidos en una biblioteca llamada repositorio ver fig. 1.2.

1.2.2 Proceso de Desarrollo con Reutilización

La principal idea en la construcción de sistemas basados en componentes es la construcción de sistemas a partir de componentes pre-existentes. Este supuesto tiene severas repercusiones en el ciclo de vida tradicional en donde la actividad de implementación es substituida por un conjunto de actividades particulares: selección, adaptación y ensamblaje [6], ver fig.1.2.
A continuación se describen estas actividades brevemente:

Figura 1.2 Ciclo de Vida de Software Basado en Componentes



I. <u>La selección de componentes:</u>

La "selección de componentes" es un proceso que determina qué componentes ya desarrollados pueden ser utilizados para la construcción del sistema a de desarrollar. Existen dos fases:

- <u>Fase de búsqueda</u>: se identifican las propiedades de un componente que satisfagan los requerimientos del sistema a construir, ejemplo la funcionalidad del componente (qué servicios proporciona) y otros aspectos relativos a la interfaz de un componente (como el uso de estándares).

- <u>Fase de evaluación</u>: en esta fase se determina el mejor componente(s), a partir de un diagnostico, esto es analizando sus partes, funciones, procesos que satisfagan mejor los requerimientos, existen técnicas relativamente maduras para efectuar el proceso de selección. Por ejemplo ISO (International Standards Organization) describe criterios generales para la evaluación de productos ISO/IEC-9126.

II. La adaptación de componentes. :

Como los componentes son creados para satisfacer diferentes necesidades basadas en el contexto de un dominio de aplicación donde se crearon, estos son adaptados solo si no satisfacen los requerimientos del sistema para el cual van a hacer desarrollados. En función del grado de accesibilidad a la estructura interna de un componente, podemos encontrar diferentes enfoques de adaptación [20]:

- Caja Negra.
- Caja Gris.
- Caja Blanca.

Una vez que los componentes son adaptados se procede al ensamblaje.

III. El ensamblaje de los componentes en el sistema.

Esta actividad describe la manera de cómo se va integrar los componentes preseleccionados en el nuevo sistema y como serán ensamblados en una unidad. Para “ensamblar” los componentes en el sistema existe una infraestructura de estilos. Los estilos más conocidos son el bus de mensajes MOM (Message-Oriented Middleware) y la tecnología ORB (Object Request Broker).

1.2.3 Ventajas y Desventajas del Proceso de Software Basado en Componentes

Como en cualquier ciclo de vida de software el desarrollo de software basado en componente tiene ventajas y desventajas a continuación se describen brevemente.

Ventajas:

- Reutilización del software.
  La idea es reutilizar componentes de software en lugar de tener que desarrollarlos desde el principio.
- Simplifica las pruebas.
  Permite que las pruebas se ejecuten probando cada uno de los componentes de manera aislada ,antes de probar el conjunto completo de componentes ensamblados.
- Simplifica el mantenimiento del sistema.
  Cuando existe un débil acoplamiento entre componentes, el desarrollador de aplicaciones es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.
- Mayor calidad.
  Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.
- Ciclos de desarrollo más cortos.

La adición de un componente pre-existente, con la funcionalidad deseada para ser implementada en el sistema, disminuirá el desarrollo del sistema en días en lugar de meses ó años [19].

- Mejor retorno de inversión.
  Usando correctamente la estrategia de la reutilización de componentes, el retorno sobre la inversión puede ser más favorable que desarrollando los componentes uno mismo.

Desventajas

- Desconocimiento del código: Las personas que desarrollaron el componente no siempre estarán presentes en las futuras modificaciones así que muchas veces solo se cuenta con un binario para utilizar en el programa.

Una vez que el componente ya esta seleccionado del repositorio, a sido evaluado, y si el resultado de la evaluación no fue 100% satisfactoria puede utilizarse la adaptación de componentes.

1.3 Adaptación de Componentes

Como se menciono anteriormente en la sección 1.2.2, la adaptación es un proceso que debe de tenerse en consideración ya que los componentes pueden provenir de terceras partes y uno puede encontrarse el problema de que no cumplan con los requerimientos del sistema en su totalidad.
Actualmente existen diferentes técnicas de adaptación las cuales, en términos generales pueden ser clasificadas de acuerdo a la estructura interna del componente [20]:

- Caja Negra: Este tipo de técnicas son aplicables a componentes en los casos en los que solo el código binario del componente es accesible
- Caja Gris: Este tipo de técnicas se aplican a componentes cuando el código fuente del componente no esta disponible para ser modificado, pero el componente provee su propia interfase de aplicación del programa “API” para realizar dichas modificaciones.
- Caja Blanca: Esta tipo de técnicas son aplicables a componentes en donde se tiene el código fuente del componente. Por lo tanto el componente puede ser significativamente modificado.

La naturaleza de la técnica dictamina en gran medida el tipo de cambios que se pueden realizar a un componente.
En términos generales los cambios que se pueden realizar por medio de una técnica de adaptación incluyen:

- Cambio de atributos adición o renombrado del atributo.
- Cambio de firmas de una operación definida en la interfase: cuando es renombrada alguna operación o cuando el parámetro o valor definido a regresar requiere un cambio.

- Una adición de una operación definida en la interfase: cuando se añade un nueva operación a la interfase en donde la implementación interna del método debe de ser añadida para satisfacer esta modificación en la interfase.
- Cambio de un método: cambio en algún comportamiento definido ya existente.

El momento en el que se da la adaptación podría clasificarse, como en la fase en donde se realiza la adaptación del componente.

1.3.1 Tiempos de Adaptación

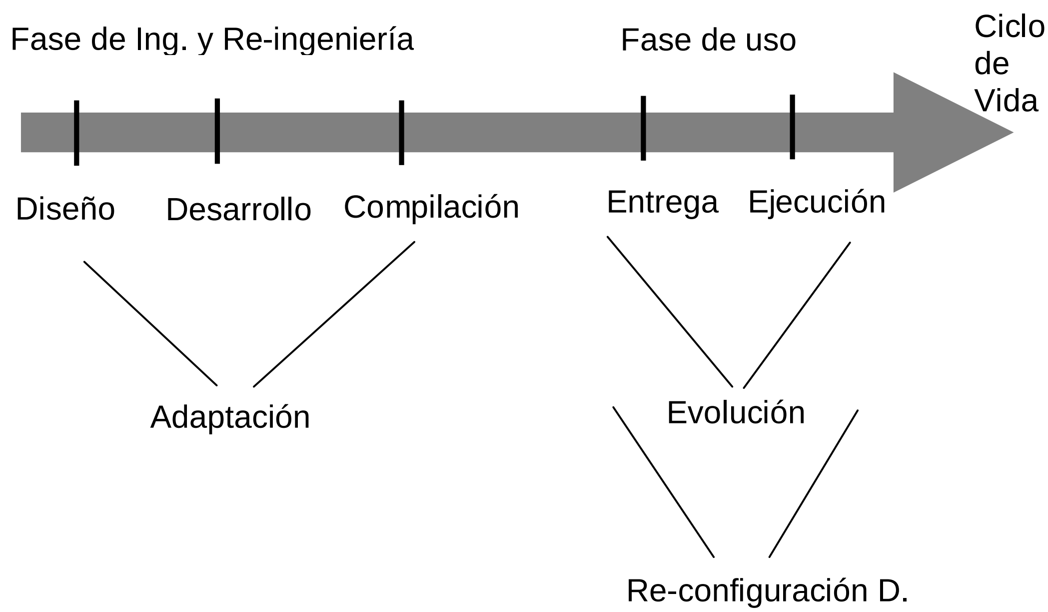
Dependiendo de la técnica de adaptación utilizada, la adaptación de componentes puede ser desarrollada durante diversas fases del desarrollo de sw.

- Tiempo de Diseño: La adaptación del componente se manifiesta como parte del diseño del sistema, para cumplir con algún requerimiento de sistema.
- Tiempo de Compilación: La adaptación del componente es realizada durante la compilación del componente, aquí es llevada por el compilador con alguna modificación especial como en el caso de superimposition.
- Tiempo de Ejecución: Este tiempo se lleva acabo cuando el componente esta ejecutándose, y se lleva acabo debido a que existe algún cambio en un servicio que no fue detectado tanto en diseño o compilación.

1.4 Adaptación vs Evolución, Substitución, Re-configuración Dinámica.

Con el fin de clarificar la noción de adaptación en este trabajo se mencionan las siguientes diferencias, la adaptación en un concepto que ha sido utilizado en diversas fuentes bibliográficas para referirse a un proceso diferente al que se tiene en este trabajo, fig 1.3.

Figura 1.3 Fases Ciclo de Vida de Software Basado en Componentes



Adaptación vs Evolución
Por ejemplo, la adaptación se utiliza como termino para referirse a evolución en donde evolución es cuando el diseñador de componentes modifica el componente que el diseño, es decir la evolución se da cuando el componente ya esta integrado en el sistema [19].

Adaptación vs Substitución
Cuando se hable de substitución se referirá de un componente remplazado por otro, y este empata tanto sintacticamente como semánticamente.

Adaptación vs Re-configuración Dinámica

En este trabajo se referirá a re-configuración dinámica cuando se hable de sistemas cerrados y se refiera a la poca o nula modificación del código de una manera novedosa en el cambio del ambiente del componente.

Ahora que se han definido los conceptos base de adaptación en el siguiente capitulo se explicaran las técnicas de adaptación.

Capítulo 2. Estado del Arte de las Técnicas de Adaptación

Existen varias técnicas por medio de las cuales se puede adaptar un componente de software, en este capitulo, se hace una revisión bibliográfica de las técnicas de adaptación propuestas por diferentes autores, tales como Binary Component Adaptation[13] , Superimposition[5] y wrapper[].

2.1 Binary Component Adaption

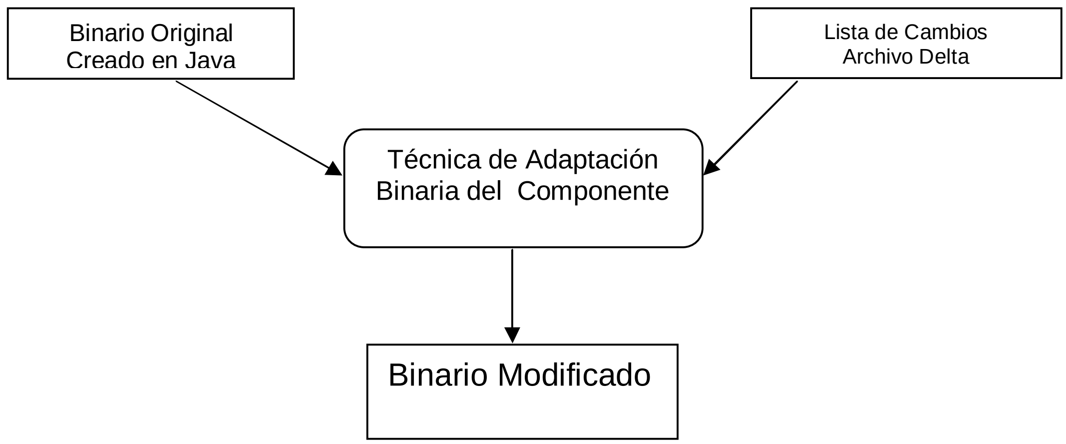
Binary component adaptation (BCA) es una técnica de adaptación propuesta por Keller y Höler,[13]. La técnica es de tipo black box ,aunque el autor no lo menciona supongo que la técnica solo es desarrollable en aplicaciones para escritorio, debido el ejemplo desarrollado, también permite realizar un gran rango de modificaciones posibles como por ejemplo: adición de métodos, renombrar métodos y campos, extender interfases, cambiar herencia. Todo esto de una manera transparente para la persona que va a usar el componente.
Aunque en [13] solo se mencionan específicamente las modificaciones antes mencionadas, el autor cita que con la adaptación binaria se puede realizar todas las modificaciones que pueda permitir el lenguaje java, también menciona en donde se puede llevar acabo la adaptación en el diseño o al vuelo.

En esta técnica de adaptación se realiza de manera casi manual mediante el uso de un envoltorio, pero existe una investigación en donde se menciona de la implementación automática de una herramienta utilizando dicha técnica [1]. Dicho envoltorio se utiliza como puente entre los cambios realizados en las interfaces del componente y el componente. Las interfases contienen una especificación de los servicios funcionales del componente, los cuales son expresados mediante una serie de firmas de métodos y nombres de campos.

La adaptación del componente toma lugar después de que el componente fue entregado al desarrollador de aplicaciones, después se realiza la adaptación para modificar la estructura interna del componente para reflejar los cambios requeridos.

Esta técnica de adaptación utiliza dos entradas y produce una salida. Específicamente requiere el archivo binario original del componente, así como la lista de cambios a realizar y produce un archivo binario que corresponde al componente modificado o adaptado ver Fig.2.3.

Figura 2.3 Adaptación de Componente a través BCA



Fuente: Binary Component Adaption , Keller y Höler, 1997

La técnica permite la realización de adaptaciones flexibles debido a que utiliza el archivo delta este archivo esta escrito en lenguaje java, en el archivo es en donde se guardan todas las modificaciones a aplicar y es compilado a formato binario. El archivo delta es único para el componente que fue desarrollado y para ser re-utilizado se necesitaría de un gran esfuerzo para implementarlo.

Una de las características de esta técnica es que el componente desarrollado puede adaptarse n veces por la técnica.

La adaptación binaria de componentes también lleva a algunos gastos indirectos en el tiempo de carga, esto es por el aumento en el tiempo de carga del componente antes de este ser ejecutado, debido la adición del envoltorio en si se analiza o hace “parsed” dos veces, una el envoltorio con las modificaciones y otra el componente adaptado. Así mismo, la técnica es muy específica a un lenguaje de programación, “java”.

2.1.1 Ejemplo

A continuación se ilustra un ejemplo muy simple de la utilización de BCA paso por paso.
Suponga que el componente a adaptar es la clase java.lang.String. Específicamente se quiere añadir la funcionalidad para la encriptación la cual el componente original carece.
Utilizando BCA tendríamos que generar un archivo delta con nombre “StringEncryption.delta” conteniendo el código de dicha adaptación:

//Se añade la funcionalidad de la encriptación “encrypt”

```
delta StringEncryption adapts class java.lang.String
{
  add method public java.lang.String encrypt()
  {
    char[] buf = this.toCharArray();
    for (int i=0; i<this.length(); i++)
      {
      buf[i] = (char) (buf[i] ^ 0x56);
       }
  return new String(buf);
};
```

//Se añade la funcionalidad para desencriptar “decrypt”

```
  add method public java.lang.String decrypt()
  {
    char[] buf = this.toCharArray();
    for (int i=0; i<this.length(); i++)
      {
      buf[i] = (char) (buf[i] ^ 0x56);
      }
      return new String(buf);
  };
}
```

Archivo StringEncryption.delta

Como puede observarse el código de adaptación contenido en el archivo delta permite añadir dos métodos para encriptar y desencriptar en el componente java.lang.String (`encrypt()`,`decrypt()`).
Una vez creado este código se procede a su compilación utilizando el archivo delta invocándo lo siguiente:

```
bcadfc StringEncryption.delta
```

Esto genera un archivo de nombre StringEncription.df en este archivo se guarda la modificación “encrypt(), decrypt()”que se pretende adaptar al componente java.lang.String.
Para poder utilizar la adaptación antes mencionada, generaremos un sistema llamado Main el cual utiliza el componente adaptado es decir los métodos agregados (encrypt(), decrypt()) .

```
public class Main
{
  public static void main(String[] args)
    {
      String s = “***secret***”;
      String encrypted = s.encrypt();
      System.out.println(“encrypted: “ + encrypted);
      String decrypted = encrypted.decrypt();
      System.out.println(“decrypted: “ + decrypted);
    }
  }
}
```

Se compila Main.java por invocacion:

```
bcajavac -deltas StringEncryption.df Main.java
```

bcajavac requiere el archivo delta StringEncryption donde se ponen las modificaciones de encriptar “encrypt” y “decrypt” en la clase Main.java.

Ahora se corre el Main por invocación:

```
bcajava -deltas StringEncryption.df Main
```

El Main produce la siguiente salida:

```
encrypted: |||%35$3”|||
decrypted: ***secret***
```

2.1.2 Ventajas y Desventajas

La adaptación binaria de componentes ofrece las siguientes ventajas:

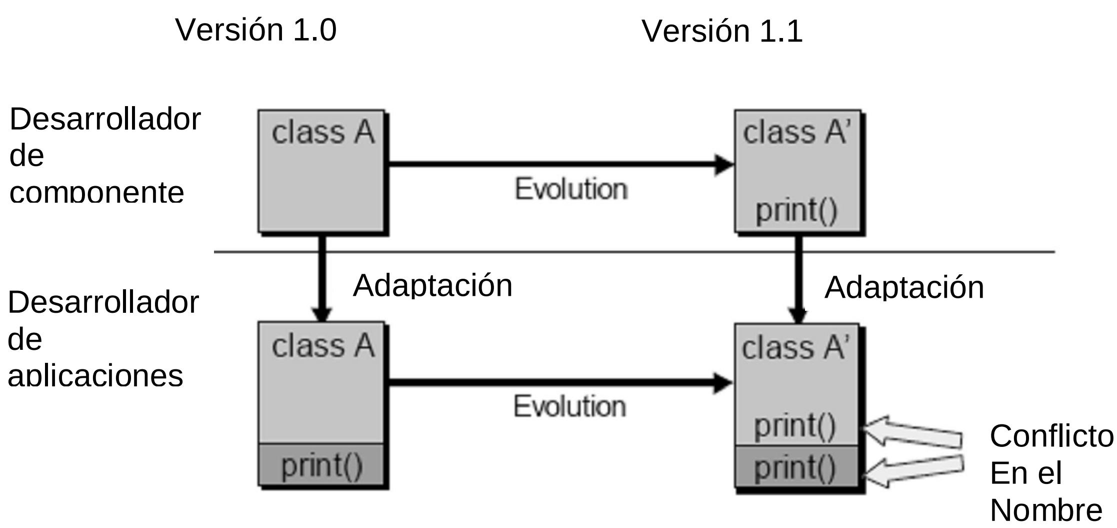
Muy flexible: Permite un gran rango de modificaciones posibles.

No necesita código fuente: BCA permite adaptación de componentes hechos en código java, sin necesidad de código fuente así que se puede utilizar con librerías de terceros. BCA utiliza la información contenida en el archivo .class realizado en java.

Modificaciones directas sobre el componente: La estructura interna de un componente se modifica sobre el componente creando envoltorios de clases que incorporan los cambios. Esto significa que la clase original es la que resulta ser modificada.

Y las siguientes Desventajas:

- Incremento en el Rendimiento o performance: puede tener un incremento en el tiempo de ejecución, durante la fase de carga, pero el código se ejecuta a una completa velocidad después de esto. El incrementa en el tiempo de carga del componente es aumentado debido a la adición del envoltorio.
- Dependiente de un lenguaje: Es muy específica a un lenguaje de programación “java”, en los articulo revisados solo se habla del lenguaje java la versión JDK1.1.5. [12][14]
- No maneja los conflictos de versiones, esto significa que un método o un campo pueden tener conflictos al momento de que se haya hecho la adaptación del componente ver fig.2.4, la técnica no resuelve los conflictos. Por ejemplo, si al momento de desarrollar un componente versión 1.0 , las adaptaciones necesarias como agregar el método print() fueron implementadas con éxito , pero al momento de que existe una evolución en el componente A’ versión 1.1 el cambio introducido por la adaptación anterior al componente A coincide con los cambios realizados por la evolución en el componente A’, ocurrirá un error en el acoplamiento[15].



Fuente: Binary Component Adaption User Guide, Keller y Höler, 1998
Figura 2.4 Adaptación de Componente a través BCA

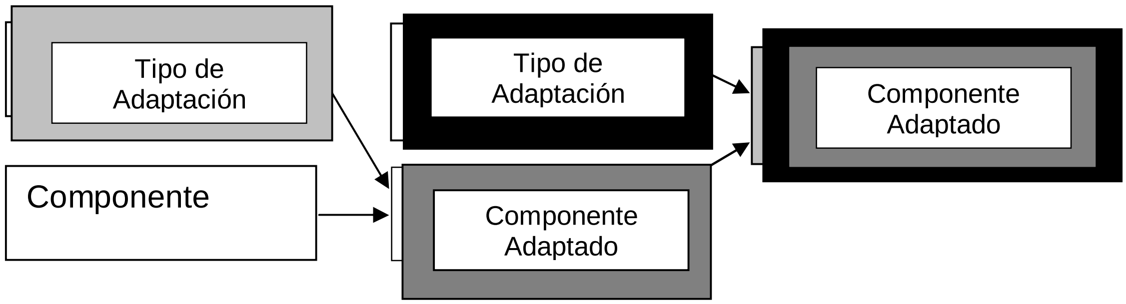
2.2 SuperImposition

SupperImposition es una técnica de adaptación basado en black box propuesta por Bosch [5]; en donde súper impone el comportamiento extendiéndolo hacia un componente de software.
Se basa en el principio que un componente y su funcionalidad de adaptación son diversas entidades que se integran apropiadamente.
En esta técnica se requiere que exista la interfaz del componente, la adaptación de un componente es apoyada por la capacidad de cambiar la especificación de la interfaz en los métodos en cuestión.

SuperImposition es implementada en (LayOM) un lenguaje de modelo de objetos basados en capas, en donde las ventajas de las capas sobre los envoltorios o wrappers es que las capas son transparentes, esto significa que proveen reutilización y personalización del comportamiento de la adaptación.

Esta técnica de adaptación es transparente como se muestra en la Fig.2.5. ,en donde se muestra que un componente adaptado que utilizo esta técnica puede tener n adaptaciones, si que el componente lo sepa.

Figura 2.5. Adaptación de Componente a través de SuperImposition



Fuente:SuperImposition:A Component Adaptation Technique,Bosch ,1999 pag7

La adaptación del componente por esta tecnica puede ser llevada un numero de n veces , debido a que el componente adaptado no se distingue entre otros componentes y la tecnica es configurable porque la adaptacion es definida en una parte especifica, esto se puede hacer en cada instancia de la adaptacion.

Con la técnica de adaptación se puede realizar lo siguiente: renombrado de clases, renombrado de métodos y campos, adición de métodos, restricción de interfaces, selección de que campo o método se quiere acceder.

2.2.1 Ejemplo

A continuación se ilustra un ejemplo muy simple de la utilización de Superimposition. Existe un sistema, desarrollado en un lenguaje orientado a objetos específicamente en C++ e implementado en LayOM.

En el lenguaje LayOM al modelar la adaptación, se genera una capa de tipo adaptador que provea la funcionalidad necesaria, esta se crea como parte de la definición de la clase, por que se extiende su comportamiento, en donde la modificación de la clase representa la adaptación a proveer. La sintaxis de la capa llamada “Adapter” es la siguiente:

<id> : Adapter(accept <mess-sel>+ as <new-mess-sel>, accept <mess-sel>+ as <new-mess-sel>, ...);

La semántica del tipo capa específica un mensaje con un seleccionador de mensajes <mess-sel> que especifica en que capa es pasada a un nuevo selector <new-mess-sel>.

La adaptador tipo capa permite más de un selector de mensajes los cuales van a ser transformados en nuevos selectores de mensajes. La capa transformara ambos mensajes enviados al objeto encapsulado por la capa y los mensajes enviados al objeto.
Abajo un ejemplo de la clase adapter:

```
class adapter

    layers

        adapt : Adapter(accept mess1 as newMessA, accept mess2, mess3 as
newMessB);

        inh : Inherit(Adaptee);

end; // class adapter
```

El ejemplo clase adaptador transforma el mensaje “mess1” en el mensaje “newMessA” y el mensaje “mess2” o “mess3” en “newMessB”. Los métodos “newMessA” y “newMessB” son implementados por la clase “Adaptee” y la capa de herencia se redirige hacia los otros instancias de los mensajes de la clase “Adaptee” que es contenida con la capa

La adaptación de un nivel del objeto puede ser alcanzada encapsulando el objeto con una instanciación de una capa adicional.

En este ejemplo la adaptación solo será efectiva para esta instancia en particular y no para ninguna instancia de la misma clase.

A continuación se muestra una parte del código de un ejemplo de un objeto adaptado.

```
...

// object declaration

adaptedAdaptee : Adaptee with layers

    adapt : Adapter(accept mess1 as newMessA, accept mess2, mess3 as newMessB);

end;

...
```

2.2.2 Ventajas y Desventajas

Ventajas:

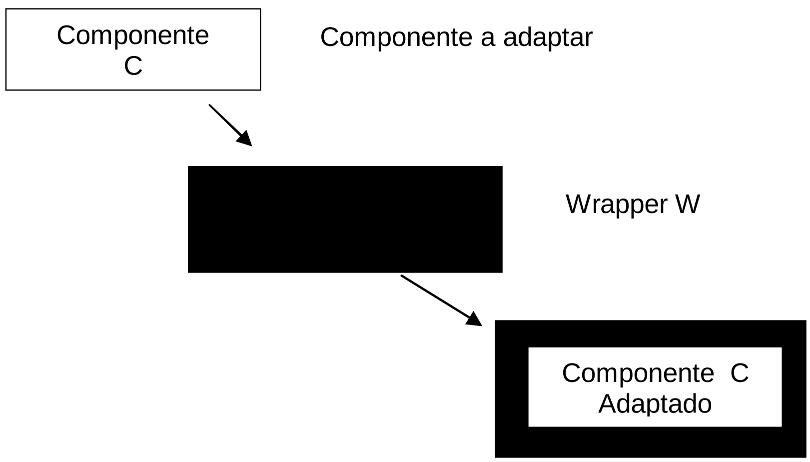
- Con la técnica de adaptación se puede realizar lo siguiente: renombrado de clases, renombrado de métodos y campos, adición de métodos, restricción de interfaces, selección de que campo o método se quiere acceder.
- Reutilización de programas informáticos.
- Herencia prevé el código que existe en un solo lugar.
- Permite al ingeniero de software de adaptación de un componente mediante una serie de comportamientos predefinidos de adaptación que pueden ser configurados para el componente específico.
- Las capas son transparentes y facilita la reutilización

Desventajas:

- Es dependiente de un lenguaje de programación:
- de LayOM que es un modelo extensible de objetos basada en la noción de DCO “delegating compiler objects(DCOs)”
- Para implementar la técnica requiere un compilador extensible , una sintaxis especial y código definido por DCO.
- Ingeniero de software puede gastar un esfuerzo considerable en la comprensión de los componentes antes de poder adaptarla.

2.3 Wrapping

Es una de las técnicas de adaptación de componentes más comúnmente utilizadas, aunque diferente a superimposition y a BCA porque no existe un autor que la haya generado.
Para realizarla no se necesita el código fuente es decir es de tipo Black-box, también conocida como “containment” en la literatura de COM [3].
Un Wrapper o envoltorio es un contenedor de un objeto que encapsula totalmente un componente C y provee una interfase que puede restringir o extender la funcionalidad del componente C. En este simple caso el envoltorio W puede ser utilizado para adaptar la interfase del componente C, el envió de llamadas al envoltorio W hace que se apropie de los métodos correspondientes de C como se muestra en la sig. fig., esto significa que la técnica soporta cambio de interfaces específicamente de nombres también soporta adición de métodos diferentes del componente original, es más cualquier cosa permisible con el lenguaje de programación con el que se implemento el componente.



Componente C encapsulado totalmente y provee una interfase que restringe o extiende la funcionalidad

Desde que el wrapper encapsula al componente a adaptar ,el componente no manda directamente los mensajes a quien lo utiliza , siempre necesitan pasar por el wrapper, esto significa que el wrapper tiene que lidiar con todos los mensajes aun con los que no necesariamente fueran a adaptarse.
El componente adaptado por el wrapper puede ser adaptado n veces por la misma técnica, pero al momento de realizar el wrapper este solo funciona para esa adaptación en particular, esto es decir no es configurable y solo puede ser reutilizable en los casos en donde se utilice el mismo comportamiento a adaptar.
Otra característica de los wrapers es que son independientes del lenguaje de programación, debido a que la técnica de adaptación es una solución que se provee con el lenguaje en que se implementará el componente, se puede realizar un wrapper en java, c++ o en otros lenguajes.

2.3.1 Ejemplo

Se asume que se ha escrito una aplicación utilizando los componentes A,B,C cada uno de ellos fue comprado a diferentes vendedores. A continuación se muestran una vista simplificada de las interfaces ofrecida por los componentes.

```
type BaseA {
  method print();
  methods mbase_a1 ,mbase_a2..
}

type SubA:BaseA{
  methods  msub_a1,msub_a2,…
}

type BaseB {
  method printPart1();
  method printPart2();
  methods mbase_b1, mbase_b2,...
}

type SubB:BaseB{
  methods msub_b1, msub_b2,...
}

type BaseC{
  method drucke ();
  methods mbase_c1, mbase_c2,...
}

type SubC:BaseC{
  methods msub_c1, msub_c2,...
}
```

"BaseA es un supertipo de todo los objetos en el componente A"

"Uno de muchos subtipos de BaseA"

"BaseB es el supertipo de todos los objetos en el componente B""

"Uno de muchos subtipos de BaseB"

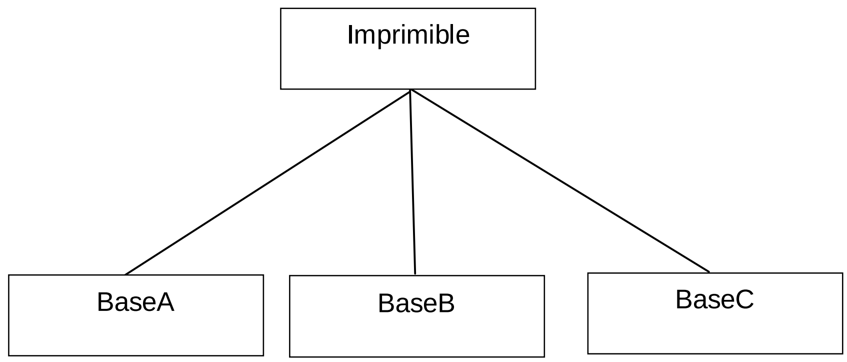
"BaseC es un supertipo de todos los objetos en el component C" , "drucke" es en alemán para imprimir

"Uno de muchos subtipos de BaseC"

Problemática de los Componentes inconsistentes

Supóngase que se necesita integrar los objetos derivados de BaseA,BaseB y BaseC en nuestra aplicación. Por ejemplo, queremos escribir un método que tome cualquier objeto de la aplicación e imprima en pantalla. Tendríamos que crear un supertipo común "imprimible" para BaseA, BaseB y BaseC como se muestra en la sig. fig.:

Figura 2.6. Clase SuperTipo Imprimible



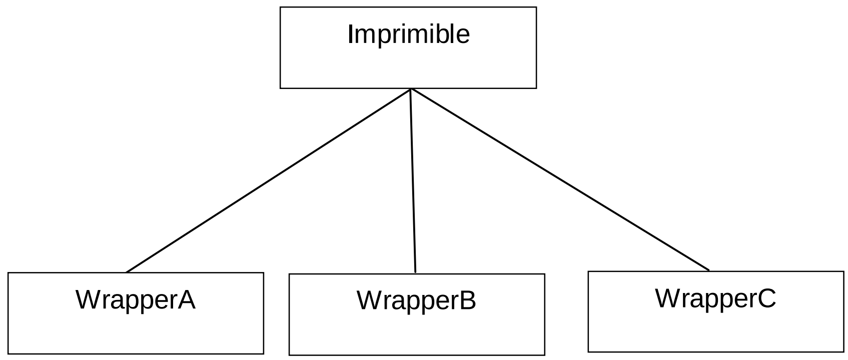
Fuente: Integrating Independently-Developed Components in Object-Oriented Languages, Urs Höslzle ,1993 pag39

Los tres componentes solo son superficialmente incompatibles por que la funcionalidad deseable ya esta presente: BaseA tiene un método print, BaseB tiene dos métodos print-Part públicos y BaseC tiene también un método print, descrito en alemán llamado “drucke”.
Una posible solución seria crear tres wrappers, cada uno conteniendo un apuntador a un objeto de su respectivo tipo.
Por ejemplo , WrapperA deberá contener un objeto BaseA:

```
type WrapperA: Printable {
   a:BaseA;              “Contiene al objeto real creado por el componente A”
   method print();        “Invoca a a.print ”

}
```

En vez de arreglar los objetos de los componentes del árbol en una relación del subtipo, arreglamos los wrappers en su relación apropiada:



Fuente: Integrating Independently-Developed Components in Object-Oriented Languages, Urs Höslzle ,1993 pag39

Cada envoltura transmite los mensajes deseados al objeto componente real, el wrapperC renombra el método “drucke”. Sucede algo similar en el WrapperB sintetiza el método print con printPart1 y PrintPart2.
En esencia , los wrappers que se generan de manera manual , aíslan el núcleo de nuestras aplicaciones , es decir ocultan los objetos reales de la aplicación , para solucionar los problemas de inconsitencias entre los componentes.

2.3.2 Ventajas y Desventajas

Ventajas

<u>No necesita código fuente</u>: no necesita de código fuente así que se puede utilizar con librerías de terceros, solo utiliza la información contenida en el archivo componente a adaptar.

<u>Modificaciones directas sobre el componente</u>: La estructura interna de un componente se modifica sobre el componente creando envoltorios de clases que incorporan los cambios.

Desventajas

- Las Desventajas de utilizar envoltorios es el elevado desempeño que se incurre por la llamada al componente adaptado tiene que ejecutarse también el wrapper.

- Los wrappers introducen redundancia adicional en el sistema debido a ellos duplican partes de las interfaces del componente adaptado, esto significa que si la interfase del componente cambia, se necesita un trabajo adicional para adaptar el programa.

2.4 Structure Fragmentation

Casi todos los trabajos que hablan de adaptación de componentes se enfocan a adaptar los servicios, en donde para realizar la adaptación de componentes lo que comúnmente se realiza es el reemplazo de un algoritmo por otro (reemplazo de código), casi ningún trabajo se enfoca a la adaptación de la estructura del componente, aclarando que estructura es la estructura externa del componente (por ejemplo puertos, interfaces o servicios), o la estructura interna del componente (por ejemplo sub-componentes, conexiones internas)[3].

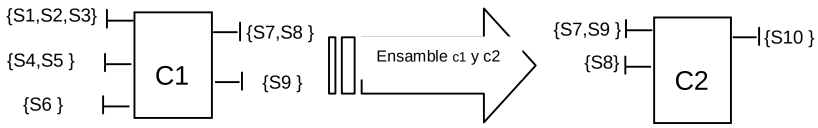
La adaptación estructural de un componente de software consiste en la capacidad de modificar la estructura de este mientras conserva su comportamiento y sus servicios. De hecho la adaptación estructural esta enfocada más bien en reorganizar la estructura del componente, es más, la estructura del componente a hacer adaptada es una lista estructural de elementos definidos en el componente así como sus enlaces.

La adaptación estructural puede consistir en la actualización de la lista de las interfaces del componente, modificando el contenido de cada una de estas interfaces o reestructurando el contenido de los puertos del componente en cuestión; también la adaptación estructural puede consistir en la descomposición de un componente de software en algunos sub-componentes donde un conjunto de servicios provistos deben de ser los mismos que se proveían antes de que sucediera la adaptación.

Esta técnica se puede decir que es del tipo de black box o white box debido a como sea llevada acabo, la adaptación, el articulo [] muestra que la técnica esta enfocado a sw empotrado.

Por ejemplo la figura 2.6 muestra que los componentes C1 y C2 no pueden ser adaptados, debido a que los puertos requeridos por C1 no empatan con los puertos provistos por C2.

Figura 2.7. Adaptación Estructural.



Imposible Empate de Componentes

Fuente:, Adapting software components by structure fragmentation , pag 1751

Debido a la falta de soporte (acceso a código) por parte de la técnica , no puede desarrollar los ejemplos , tal como se venia desarrollando en el trabajo. A continuación muestro un ejemplo resumido del ejemplo que incluye el paper [3].

2.4.1 Ejemplo

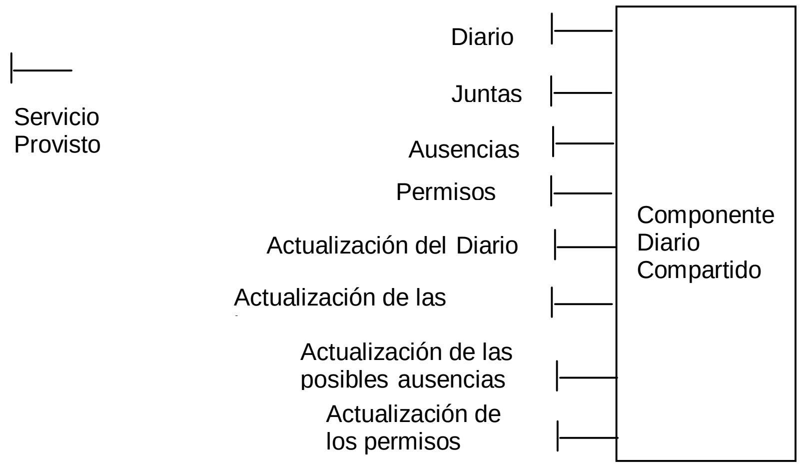
Se quiere compartir un sistema que maneja un diario compartido por múltiples usuarios, en donde se implementaría este componente utilizando la plataforma java en el modelo fractal.

El componente a adaptar en un componente monolítico lo que significa que solo es un solo componente y el cual provee los siguientes servicios ver fig 2.6:

- <u>Administración del uso personal del diario</u>, aquí se incluye lo que es autentificación, consultas a eventos, etc. Estos servicios son provistos a través de la interfase “Diario”.
- <u>Organización de las juntas</u>: permite la posibilidad de organizar juntas en donde la fecha y las listas de las personas con las cuales se llevaran acabo las juntas. Este servicio será provista a través de la interfase “Juntas”.
- <u>Administración de las posibles ausencias</u>, aquí se da la posibilidad de añadir una posible ausencia, o poder consultar todas las ausencias existentes en un periodo. Este servicio será provista a través de la interfase “Ausencias”.
- <u>Administración de los permisos</u>: aquí es donde se da los permisos y se administran los permisos. Este servicio será provista a través de la interfase “Permisos”.
- <u>Actualización del diario</u>: actualización de las fechas de las reuniones, la fechas de ausencia Este servicio será provista a través de la interfase

“Actualización del Diario”, “Actualización de las juntas”, “Actualización de las posibles ausencias”, “Actualización de los permisos”.

Figura 2.6. Adaptación Estructural.



Fuente:, Adapting software components by structure fragmentation , pag 1751

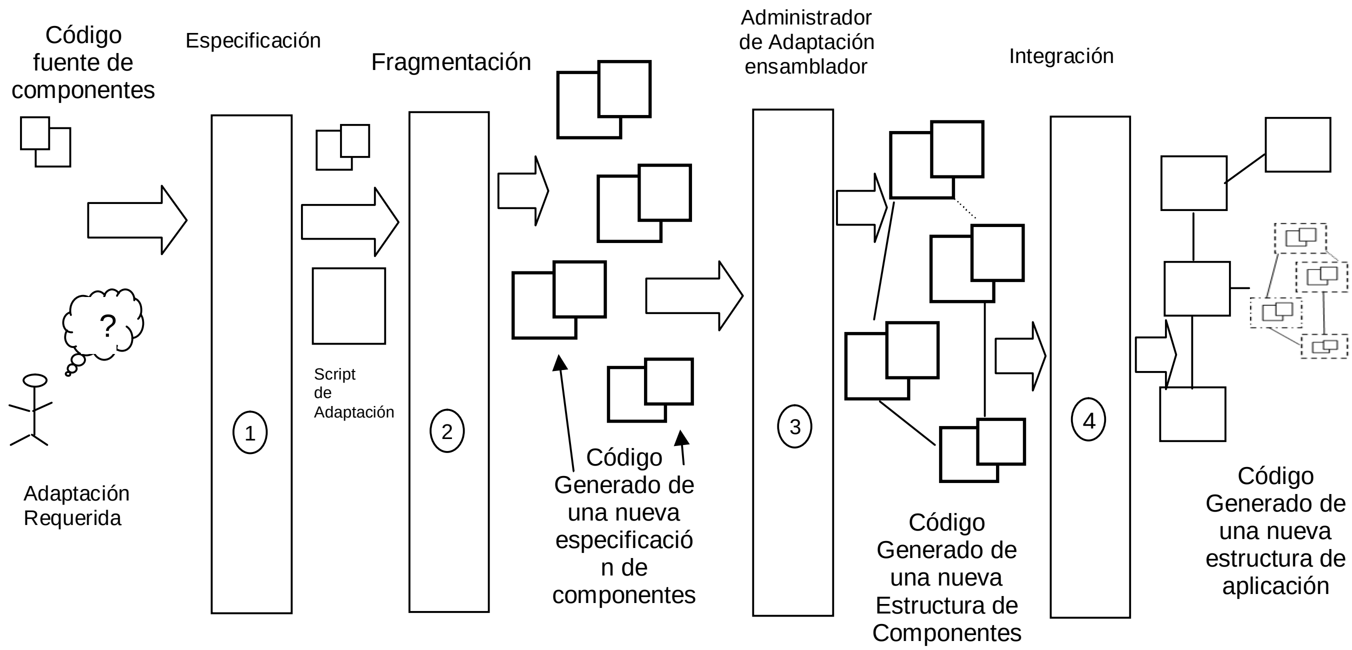
La adaptación de componente se llevo acabo en cuatro escenarios, el siguiente es el proceso sig. fig :

1. Especificación: La especificación es creada a través de un script en donde se especifica la adaptación utilizada, en un formato cercano al lenguaje XML, para cada nuevo componente se especifica sus puertos y para estos puertos sus interfaces asociadas.
2. Implementacion: a partir de la especificación se genera la implementación ,la adaptación estructural puede consistir en la descomposición de un componente de software en algunos sub-componentes donde un conjunto de servicios provistos deben de ser los mismos que se proveían antes de que sucediera la adaptación.

3. El ensamble. Este ensamble refleja los enlaces en los cuales existen entre estos componentes que se van a adaptar .
4. Integración. Consiste en que la conexión del nuevo componente creado con la otra aplicación que lo utiliza, de una manera transparente, de hecho la aplicación que utilice el componente adaptado debe de funcionar de igual manera, antes de usar el componente adaptado.

La meta de esta técnica es reorganizar clases, variables y métodos en una nueva jerarquía , para poder facilitar el futuro de la adaptación, la técnica implementada utilizando un framework llamado Julia, y desarrollado en java.

Figura 2.7. Adaptación Estructural.



Fuente:, Adapting software components by structure fragmentation , pag 1751

2.4.2 Ventajas y Desventajas

Ventajas

- Es una nueva forma de adaptar los componentes ya generados, que se basa en modificar la estructura del componente

Desventajas

- Esta enfocada o sw empotrado

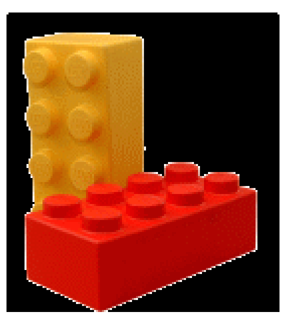
Capítulo 3. Análisis de las Técnicas de adaptación de componentes

3.1 Con respecto a sus requerimientos

Normalmente cualquier nueva técnica de adaptación propuesta trata de demostrar que cumple con los requerimientos funcionales usuales, estos requerimientos proporcionan un marco que se pueda utilizar para evaluar técnicas de adaptación de componentes convencionales, que cualquier técnica debería de satisfacer para llamarse técnica de adaptación, a continuación muestro los requerimientos y que técnicas cumplen con que requerimientos..

- Transparente: La adaptación de un componente debe de ser tan transparente como sea posible. Significa que el desarrollador quien usa el componente original antes de la adaptación o el desarrollador que utilizo el componente después de la adaptación, ninguno se dio cuenta de que el componente fue adaptado y además todos los aspectos del componente que no fueron adaptados , deben de ser accesibles si ningún esfuerzo de la adaptación.
- Black-box: la adaptación del los componentes es limitada solo a través de las interfaces.

- Composable: La técnica de la adaptación debe de ser facil de implementar al componente a adaptar, es decir ninguna redefinición del componente debe ser requerida. Es más, el componente adaptado debe ser tan composable con otros componentes como estaba sin la adaptación. Finalmente, la adaptación debe ser composable con otras adaptaciones.



• Configurable: Se refiere a que debe de tener la técnica alguna forma de configurar lo que va a adaptar en el componente.

• Reusable: Que la técnica de adaptación pueda rehusarse n veces, esto significa que el ingeniero de sw no tenga que volver a desarrollar la técnica para volverla a implementar.

- Independiente del lenguaje: la técnica de adaptación no debe de depender del lenguaje en el que se vaya a implementar.

Análisis de wrapper con respecto a los requerimientos son los siguientes:

- Transparente: Desde que el wrapper encapsula al componente a adaptar ,el componente no manda directamente los mensajes a quien lo utiliza , siempre se necesita pasar por el wrapper, esto significa que el wrapper tiene que ligar con todos los mensajes aun con los que no necesariamente fueran a adaptarse.

- Black-box: El wrapper no tiene forma de acceder a los elementos internos del componente adaptado.

- Composable: Un componente adaptado por un wrapper puede ser envuelto por otro wrapper, este proceso puede ser recursivo

- Configurable: al momento de realizar el wrapper, este solo funciona para esa adaptación en particular, no es configurable esto significa que al momento de querer cambiar algo mas en el wrapper , se necesita un trabajo de programación muy laborioso.

- Reusable: El wrapper solamente puede ser re-utilizado en los casos en donde se utilice exactamente el mismo comportamiento a adaptar. Pero como el wrapper no es configurable, cualquier diferencia de la adaptación original hace imposible la reutilización, esto generaría un nuevo wrapper.

- Independiente del lenguaje: Es independiente debido a que la solución que provee no esta ligada con el lenguaje en que se implementa, se puede realizar un wrapper en java, c++ y en otros lenguajes.

Análisis de supperimposition con respecto a los requerimientos son los siguientes:

- Transparente: La adaptación que se súper impone es totalmente transparente, solo el comportamiento que necesita ser adaptado es afectado por la adaptación.

- Black-box: Superimposition es de tipo black-box y no depende de la implementación actual del componente a adaptar , por que esta no depende de quien utilice este componente a adaptado.

- Composable: es libremente composable debido a un componente adaptado puede ser adaptado n veces y el componente adaptado no se distingue de otros componentes.

- Configurable: La adaptación se lleva acabo de una parte específica, en donde la parte específica puede ser definida para cada instancia de la adaptación haciendo que la técnica sea configurable.

- Reusable: como la técnica especifica lo que se quiere adaptar, esto hace que sea reusable.

- Independiente del lenguaje: no es independiente del lenguaje porque el autor especifica que la técnica utiliza LayOM.

Análisis de BCA con respecto a los requerimientos son los siguientes

- Transparente: Los componentes creados a partir de esta técnica poseen diversos identificadores del componente que fueron generados, dado que se genera una nueva interfaz que es proporcionada después de envolver el componente, permitiendo su uso sin saber que el componente se a adaptado.

- Black-box: BCA soporta la adaptación de componentes con un mínimo entendimiento, porque no necesita del código fuente.

- Composable: El componente adaptado se puede adaptar una y otra vez con el mismo proceso de la adaptación.

- Configurable: no debido a que el archivo delta solo puede utilizarse en el componente para el cual fue creado.

- Reusable: como solo puede ser utilizado el archivo delta para el componente que se adapto, el o los componentes en donde se quisiera reutilizar deberían de tener la misma interfaz , del componente adaptado.

- Independiente del lenguaje: no es independiente debido a que la técnica solo se puede implementar en Java.

Análisis de Structure Fragmentation con respecto a los requerimientos son los siguientes

- Transparente: El componente generado permite su uso sin saber que el componente se a adaptado.

- Black-box: Literalmente menciona que la técnica puede implementarse ya sea de manera de black-box o de white-box.

- Composable: El paper no menciona y no da ningún indicio que se cumpla este requerimiento.

- Configurable: La única referencia que hace el paper [3] de configurable es la de la utilización del archivo XML como un script en donde se especifica la adaptación utilizada, en un formato cercano al lenguaje XML, para cada nuevo componente pero esto no significa que se cumpla con este requerimiento.

- Reusable: no menciona nada el articulo, y a partir de la información que menciona se supone que no es reutilizable.

- Independiente del lenguaje: no es independiente debido a que la técnica solo se puede implementar en Java.

Todo el análisis con respecto a los requerimientos se podría resumir en la siguiente tabla

	Transparente	Black-box	Composable	Configurable	Reusable	Independiente del lenguaje
BCA	✓	✓	✓	⊘	✓	⊘
Wrapper	✓	✓	✓	⊘	✓	✓
Superimposition	✓	✓	✓	✓	✓	⊘
Structure Fragmentation	✓	✓	?	⊘	?	⊘

? Con la información que se tiene no se puede llegar a una decisión.

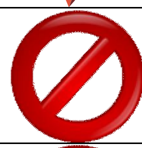
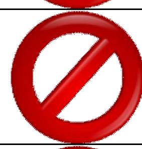
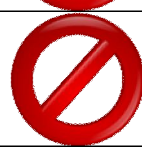
✓ Si cumple con ese requerimiento.

⊘ No cumple con ese requerimiento

3.2 Requerimientos No funcionales detectados

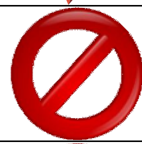
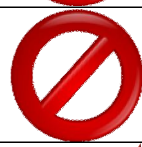
También las técnicas de adaptación atacan otros requerimientos no funcionales, como el de la documentación, BCA se ocupa de la parte de la documentación utilizando un archivo delta que podría utilizarse como documentación para posibles adaptaciones.

Y superimposition, como Structure Fragmentation también manejan una declaración de la adaptación pero es muy difícil considérala como documentación. Y la técnica de wrapper ni siquiera hace nada para documentar la adaptación.

	Documentación
BCA	
Wrapper	
Superimposition	
Structure Fragmentation	

3.3 Con respecto al grado de automatización

Ni wrapper o Supperimposition tienen alguna forma de hacer la adaptación de manera automática, todo el proceso se lleva acabo de manera manual no así BCA y Structure Fragmentation.

	Herramienta para implementar la técnica
BCA	
Wrapper	
Superimposition	
Structure Fragmentation	

3.4 Conclusiones.

Se observo que las técnicas de adaptación aquí analizadas tratan de proveer una manera sistemática, para resolver el problema de la adaptación de componentes…también se asume que la tendencia de las técnicas de adaptación están enfocadas al momento del vuelo debido a estas características encontradas en la literatura:

- Al momento del diseño del sistema no se puede saber todas las posibles cambios que le puedan influir al sistema[21].
- Cada vez mas el desarrollo de aplicaciones enfocadas a sistemas abiertos y distribuidos(Web) dando las situaciones que esto implica. [22]
  - No tener la posibilidad de detener el sistema para realizar una adaptación.
  - Tener el software distribuido.

También se puede observar que todas las técnicas aquí analizadas no proveen una herramienta, no asi el caso de BCA , que aunque no son los autores de la tecnica quienes proveen la herramienta [1].

3.5 Discusión y Trabajos futuros

- Un posible campo de investigación para desarrollar, en el tema de técnicas de adaptación de componentes es enfocadas a Sistemas distribuidos.

También se puede dejar como un tema para próximas investigaciones, los siguientes problemáticas encontradas.

Dentro del estudio de las propiedades del nuevo componente generado a partir de la técnica de adaptación, refiriéndome a propiedades como lo siguiente:

- El componente generado cumple con los requerimientos funcionales del sistema.
- El componente generado tiene en consideración el problema “de la clase base frágil” (FBCP), porque las técnicas aquí mostradas solo checan sintacticamente que empaten las diferencias y no semánticamente.
- El componente generado cumple con los requerimientos no funcionales del sistema, refiriéndose a seguridad, confiabilidad, etc.

Bibliografía

[1]Jeong Ah Kim, Kyung Whan Lee ,2005, Binary component adaptation technique and supporting tool, Software Engineering Research, Management and Applications, 2005. Third ACIS International Conference on p.14-19

[2]Sven Apel and Christian Lengauer , Superimposition: A Language-Independent Approach to Software Composition, Department of Informatics an Mathematics , 2008

[3]Gautier Bastide, Abdelhak Seriai, Mourad Oussalah,Adapting software components by structure fragmentation ,Symposium on Applied Computing archive Proceedings of the 2006 ACM symposium on Applied computing, 2006, pag 1751

[4]Steffen Becker, Antonio Brogi,Ian Gorton , Sven Overhage , Alexander Romanovsky and Massimo Tivoli ,Towards an Engineering Approach to Component Adaption

[5]Jan Bosch,SuperImposition:A Component Adaptation Technique , Technical Report TR, Department of Computer Science and Business Administration, University of Karlskrona/Ronneby, September 1997

[6]Brown, A.W. “Large-Scale, Component-Based Development”. Prentice Hall PTR, 2000.

[7]Christiansoon B., Component-Based Systems Development, In Nilsson A.G. & Pettersson, J.S. (eds.) On Methods for Systems Develpment in Professional Organisations, Studentlitteratur, Lund, 2001

[8]Ivica Crnkovic , Stig Larsson and Michel Chaudron , Component-based Development Process and Component Lifecycle, Journal of Computing and Information Technology 2005

[9]Ivica Crnkovic, Stig Larsson, Michel Chaudron, Component-based Development Process and Component Lifecycle, 1Mälardalen University, Västerås, Sweden Suzanne W. Dietrich, Rohini Patil, Amy Sundermier, Susan D. Urban, Component adaptation for event-based aplication integration using active rules.

[10]Suzanne W. Dietrich, Rohini Patil, Amy Sundermier, Susan D. Urban, Component adaptation for event-based aplication integration using active rules.

[11]Gelernter,D. And N. Carriero , Cordination Languages and their Sigificance , Común ACM 35 p.96. 1992,

[12]George T. Heineman, An Evaluation of Component Adaptation Techniques , In DARPA/OMG Workshop on Compositional Software Architectures, Monterey, CA, January 1998

[13]George T. Heineman, Helgo O Ohlenbusch ,An Experiment in component Adaptation , ICSE 2000,

[14]Ralph Keller and Urs Hölzle,Binary Component Adaptation,Department of Computer Science University of California, 1997

[15]Ralph Keller and Urs Hölzle,The Binary Component Adaptation User Guide,Department of Computer Science University of California, 1998

[16]S. Mahmood ,R. Lai and Y.S. Kim , Survey of component-based software development

[17]Szyperski, C., Gruntz, D., and Murer, S.: ‘Component software –beyond object – oriented programming’ (Addison-Wesley, 2002,2nd edn.)

[18]Pressman, R. “Ingeniería del software. Un enfoque práctico”. Cuarta Edición.McGraw-Hill/Interamaericana de España. 1998.

[19]József Tick,Special Aspects of Component Based Software Development

[20]Valetto, G., Kaiser, G. E.: Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments Proceedings of the 7th IEEE International Workshop on CASE, Toronto, Ontario, Canada, July 10-14, 1995, pp. 40-48

[21] M. Katrib; J. L. Pastrana y E. Pimentel. Adaptación y reutilización de componentes distribuidos**.** En Hernán Astudillo y Carla Taramasco Toro (Eds.). Actas del 8º Workshop Iberoamaericano de Ingeniería de Requisitos y Ambientes Software, IDEAS'05. 2 al 6 de mayo de 2005, Valparaíso, Chile. págs. 203-214. Dpto. de Informática; Universidad Técnica Federico Santa María: Chile, 2005.

[22] Brogi Antonio,Canal Carlos , Pimentel Ernesto , Dipartimento di Informatica, Universit` di Pisa (Italia),Adaptación automática de componentes: una experiencia de desarrollo orientado a aspectos