Ingeniería Investigación y Tecnología, volumen XV (número 2), abril-junio 2014: 187-198
ISSN 1405-7743 FI-UNAM
(artículo arbitrado)

Modelo formal para la reestructura de marcos orientados a objetos hacia arquitecturas modelo-vista-adaptador

Formal Model for Restructuring of Object-Oriented Frameworks to Architecture Model-View-Adapter

Santaolaya-Salgado René
Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET)
Departamento de Ciencias Computacionales
Cuernavaca, Morelos, México
Correo: rene@cenidet.edu.mx

Fragoso-Díaz Olivia Graciela
Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET)
Departamento de Ciencias Computacionales
Cuernavaca, Morelos, México
Correo: ofragoso@cenidet.edu.mx

Zamudio-López Sheydi Anel
Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET)
Departamento de Ciencias Computacionales
Cuernavaca, Morelos, México
Correo: snzamudio@cenidet.edu.mx

Información del artículo: recibido: junio 2012, aceptado: mayo 2013

Resumen

La reestructura de código legado puede realizarse con fines diferentes, entre los que se encuentran la migración hacia nuevas tecnologías que faciliten el mantenimiento y la reutilización del código. Los marcos orientados a objetos (*frameworks*) cuentan con características que, de cierta manera, limitan el reuso de su código. En este trabajo se propone un modelo formal que describe un proceso de reestructura de código legado de *marcos orientados a objetos* (MOO) hacia código conforme a la *arquitectura modelo-vista-adaptador* (MVA). Este proceso se lleva a cabo aplicando 11 métodos de reestructura, con el objetivo de separar el código de la lógica del negocio (el modelo), la cual es la parte más reusable del marco, del código que implementa la vista y del código que controla el procesamiento específico de la aplicación. Como resultado, el código legado del marco queda preparado para una migración posterior hacia servicios web.

Descriptores:

- marcos orientados a objetos
- reuso de *software*
- reingeniería de *software*
- patrón MVA
- modelo formal
- Teoría de Modelos

Abstract

The restructuring of legacy code can be done for different purposes, among which is the migration to new technologies that facilitate the maintenance and code reuse. The frameworks have features that, in some way, limit the reuse of your code. In this paper, we propose a formal model that describes a process of restructuring legacy code object-oriented frameworks (MOO) to code according to the architecture Model-View-Adapter (MVA). This process is carried out using 11 methods of restructuring, with the aim of separating the code from business logic (the model), which is the most reusable framework, the code that implements the view and the code that handles specific processing of the application. As a result, the legacy code of the framework is ready for a subsequent migration to Web services.

Keywords:

- *frameworks*
- *software reuse*
- *software reeingeniering*
- *MVA pattern*
- *formal model*
- *Model theory*

Introducción

El concepto de *marco orientado a objetos* aportó una nueva dimensión a la noción de reuso de *software*, al considerar un diseño genérico, adaptable a situaciones específicas. Sin embargo, el grado de dependencia de los componentes que integran el marco ocasiona que las clases no puedan sacarse de su contexto y que el marco deba ser reutilizado como un solo componente.

Para elevar el nivel de reuso del código legado de un marco orientado a objetos, se requiere diseñar un proceso de reingeniería basado en operaciones de transformación, que considere la reestructura arquitectónica del código del marco, de manera que sea posible identificar y separar los elementos de código que pueden ser más reutilizables, sin perder la funcionalidad completa ofrecida por el marco.

En este trabajo se propone un proceso de reestructura de código de marcos orientados a objetos hacia código conforme a la arquitectura *modelo-vista-adaptador* (MVA), con el objetivo de separar el código de la lógica del negocio (el modelo), el cual es la parte más reusable del marco, del código que implementa la vista y del código que controla el procesamiento específico de la aplicación.

Este proceso implementa el esquema de solución que se muestra en la figura 1, donde T1 representa el conjunto de operaciones de transformación aplicado al marco orientado a objetos para obtener un marco orientado a objetos con arquitectura MVA.

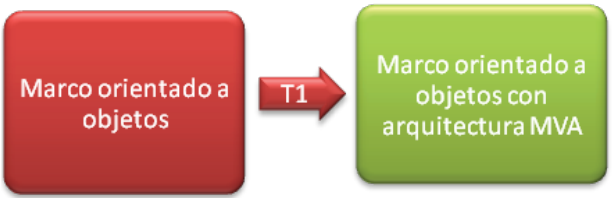

Figura 1. Esquema de solución propuesto para reestructurar marcos orientados a objetos hacia marcos orientados a objetos con arquitectura MVA

El esquema de solución se describe formalmente a través de un modelo construido utilizando elementos de la Teoría de Modelos.

Marcos orientados a objetos

En el contexto de este trabajo, un marco es un conjunto semi-completo de clases en colaboración que incorpora un diseño genérico, el cual puede adaptarse a una variedad de problemas específicos para producir nuevas aplicaciones hechas a la medida Santaolaya (2003).

Generalmente, el marco consiste en clases abstractas que definen la estructura y el comportamiento genérico del marco y forman la base para la aplicación desarrollada a partir del marco. Sin embargo, también puede contener clases concretas e interfaces que sean significativas, porque se utilizan para todas las aplicaciones de un dominio desarrolladas a partir del marco (Froehlich *et al*., 1998). En el contexto de este trabajo consideramos como elementos estructurales de los marcos, a las clases que los conforman, junto con las relaciones entre ellas. La figura 2 muestra el ejemplo de un marco orientado a objetos.

La funcionalidad de los marcos puede definirse como su capacidad para proveer funciones que responden a necesidades expresadas o implícitas cuando el marco se utiliza en condiciones específicas (ISO/IEC 9126-1).

Marcos orientados a objetos con arquitectura MVA

El patrón *Model-View-Adapter* (MVA) es una variante del patrón *Model-View-Controller* (MVC), el cual es un pa-

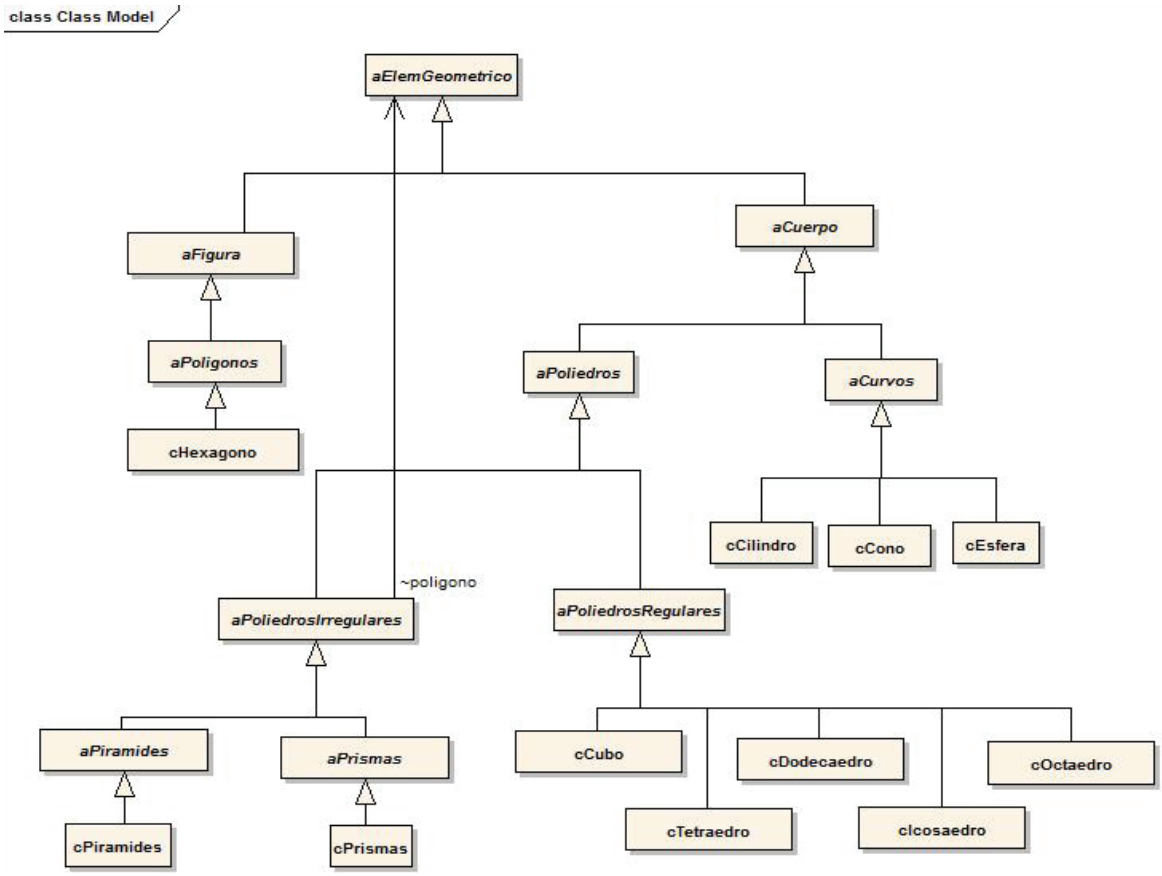

Figura 2. Diagrama de clases de un marco orientado a objetos

trón arquitectónico que separa la lógica del negocio (el modelo) de la parte de presentación de datos (la vista) (Hintze, 2009).

En el contexto de este trabajo, un marco de aplicaciones orientado a objetos con arquitectura MVA es un marco reestructurado por medio de un conjunto de operaciones de transformación para implementar el patrón MVA, sin afectar la funcionalidad original del marco orientado a objetos. En la figura 3 se muestra un ejemplo de una arquitectura de clases conformada al patrón MVA.

Descripción del modelo formal

Se utilizó la Teoría de Modelos para describir formalmente el esquema de solución propuesto.

Definición del lenguaje de primer orden

Sean P y Q sistemas. Definimos un lenguaje de primer orden $\mathcal{L}$ adecuado a P y Q, tal que su alfabeto cuenta con los símbolos siguientes:

Variables: a, $m, a_1,\ldots,a_n, m_1,\ldots,m_n, x, y, r, x_1,\ldots,x_n, y_1,\ldots,y_m$
Relatores: $\langle R_1 \rangle_{1\in J}$, $\langle R_2 \rangle_{2\in J}$, $\langle R_3 \rangle_{3\in J}$, $\langle R_4 \rangle_{4\in J}$, $\langle R_5 \rangle_{5\in J}$, $\langle R_6 \rangle_{6\in J}$ e =, como relator con rango 2 que denota la relación de igualdad o equivalencia.
Descriptor: |
Signos lógicos: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow, \forall, \exists$
Signos relacionales: $\in, \ni, \neq$
Signos binarios: $\cup, \times$
Otros signos: $\psi, \delta, \varnothing, \coloneqq$
Paréntesis: ( ), [ ], { }, < >

En la definición del lenguaje $\mathcal{L}$ se han incluido los signos relacionales $\in, \ni, \neq$ que corresponden a las operaciones relacionales de la teoría de conjuntos "*pertenece a*", "*contiene como elemento*" y "*no es igual a*", respectivamente; el signo binario $\cup$ que corresponde a la operación binaria "*unión*" y el signo binario $\times$ que corresponde a la operación binaria "*producto cartesiano*". También se incluyen otros signos como $\psi, \delta, \varnothing, \coloneqq$ que corresponden a "*configuración*", "*funcionalidad*", "*conjunto vacío*" y "*subsistema de*", respectivamente.

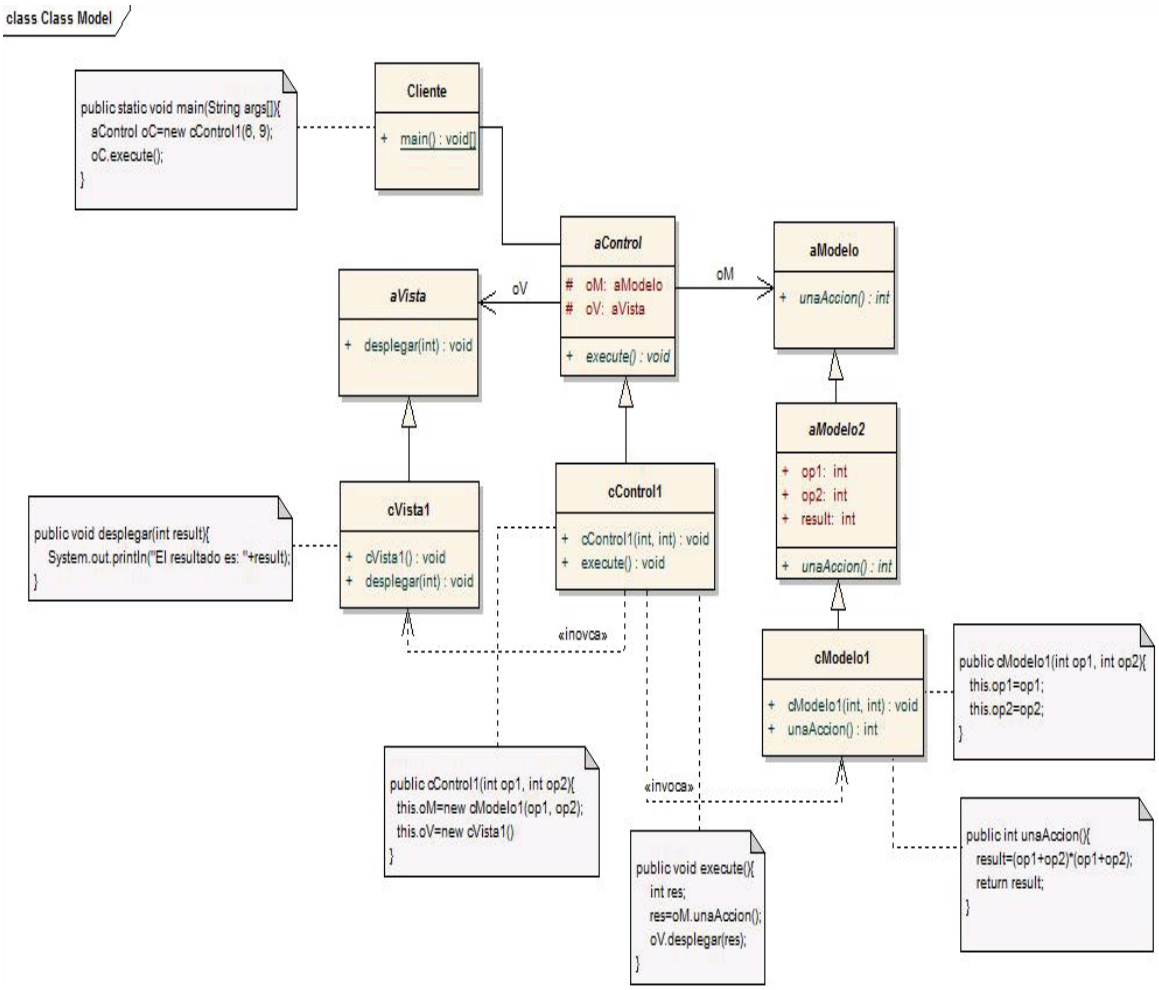

Figura 3. Ejemplo de una arquitectura de clases conformada al patrón MVA

Definición de una clase

Sea *AT* el conjunto de atributos y *MT* el conjunto de métodos, es posible definirlos de la siguiente manera:

$AT = \{a \mid \forall\ a\ A(a)\}$ donde $A(a) =$ "*a es un atributo de clase*" (1)

Se define ahora al conjunto de métodos *MT*:

$MT = \{m \mid \forall m\ (M(m) \wedge (Mi(m) \vee Ma(m)))\ \}$ (2)

donde:
$M(m)$ = "m es un método"
$Mi(m)$ = "m es un método implementado"
$Ma(m)$ = "m es un método abstracto"

Una clase es un conjunto de atributos y métodos, que puede definirse como:

$C = \{a_1, \ldots, a_m, m_1, \ldots, m_n\}$ donde $C \subseteq AT \cup MT$ (3)

El conjunto de clases que conforman un marco se define como:

$C = \cup_{i=\varnothing}^{n} C_i$ (4)

Definición de relación

Una relación es una conexión entre elementos de conjuntos. Podríamos definir una relación como:

$R = \{\langle x, y \rangle \mid \langle x, y \rangle \in \boldsymbol{C} \times \boldsymbol{C}\}$ (5)

Sin embargo, en un marco existen diferentes tipos de relaciones, por lo que es necesario entonces, definir cada una de las relaciones existentes en un marco.

Definición de las relaciones de asociación

Sea α la fórmula que define las relaciones de asociación existentes en un marco, tal que:

$$\alpha = \exists x \exists y R_1(x,y) \text{ donde } R_1(x, y) = \text{“}x \text{ está conectado con } y\text{”} \quad (6)$$

Definición de las relaciones de dependencia

Sea γ la fórmula que define las relaciones de dependencia existentes en un marco, tal que:

$$\gamma = \exists x \exists y R_2(x,y) \text{ donde } R_2(x, y) = \text{“}x \text{ depende de } y\text{”} \quad (7)$$

Definición de las relaciones de herencia

Sea β la fórmula que define las relaciones de herencia existentes en un marco, tal que:

$$\beta = \exists x \exists y R_3(x,y) \text{ donde } R_3(x, y) = \text{“}x \text{ hereda de } y\text{”} \quad (8)$$

Definición de las relaciones de agregación

Sea o la fórmula que define las relaciones de agregación existentes en un marco, tal que:

$$o = \exists x \exists y R_4(x,y) \text{ donde } R_4(x, y) = \text{“}x \text{ es parte de } y\text{”} \quad (9)$$

Definición de las relaciones de composición

Sea ρ la fórmula que define las relaciones de composición existentes en un marco, tal que:

$$\rho = \exists x \exists y R_5(x,y) \text{ donde } R_5(x, y) = \text{“}x \text{ existe si } y \text{ sólo si existe } y\text{”} \quad (10)$$

Definición de las relaciones de implementación

Sea η la fórmula que define las relaciones de implementación, existentes en un marco, tal que:

$$\eta = \exists x \exists y R_6(x,y) \text{ donde } R_6(x, y) = \text{“}x \text{ implementa } y\text{”} \quad (11)$$

Modelo del marco orientado a objetos

Sea P un sistema relacional, tal que

$$P = \langle \mathbf{A}, R_1, R_2, R_3, R_4, R_5, R_6 \rangle \text{ donde } \mathbf{A} = \cup_{i=1}^{n} C_i \quad (12)$$

La configuración (ψP) representa las combinaciones específicas de clases y relaciones existentes en el marco orientado a objetos y se define como:

$$\psi P = \{A \cup H \cup D \cup Ag \cup Co \cup I\} \quad (13)$$

donde:

$$\boldsymbol{A} = \{\langle x,y\rangle \in \boldsymbol{A} \mid x = y \vee x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P[xy] sat\ \alpha\} \quad (14)$$

$$H = \{\langle x,y\rangle \in \boldsymbol{A} \mid x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P\ [xy] sat\ \beta\} \quad (15)$$

$$D = \{\langle x,y\rangle \in \boldsymbol{A} \mid x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P[xy] sat\ \gamma\} \quad (16)$$

$$Ag = \{\langle x,y\rangle \in \boldsymbol{A} \mid x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P[xy] sat\ o\} \quad (17)$$

$$Co = \{\langle x,y\rangle \in \boldsymbol{A} \mid x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P\ [xy] sat\ \rho\} \quad (18)$$

$$I = \{\langle x,y\rangle \in \boldsymbol{A} \mid x \neq y\} = \{\langle x,y\rangle \in \boldsymbol{A} \mid P[xy] sat\ \eta\} \quad (19)$$

La funcionalidad (δP) está soportada por la configuración de las clases y relaciones existentes en el marco orientado a objetos y se define como

$$\psi P \rightarrow \delta P \quad (20)$$

Si tenemos ciertas clases y ciertas relaciones en el marco orientado a objetos, entonces estas combinaciones específicas de clases y relaciones soportarán la funcionalidad ofrecida por el marco orientado a objetos.

Modelo del marco orientado a objetos con arquitectura MVA

Formalmente un marco orientado a objetos con arquitectura MVA puede describirse como el sistema relacional Q, definido como

$$Q = \langle \mathbf{B}, \varsigma, \vartheta, R_1, R_2, R_3, R_4, R_5, R_6 \rangle \text{ donde } \mathbf{B} = \cup_{i=1}^{m} C_i \quad (21)$$

En la estructura del patrón arquitectónico MVA existe una conexión entre las partes del *adaptador* y de la *vista*, y también entre el *adaptador* y el *modelo*, como se muestra en la figura 3. En el contexto de este trabajo, estas conexiones se implementan como relaciones de composición y se consideran individuos destacados del sistema Q, que se describen formalmente como:

$$\varsigma = \exists x \in A \exists y \in V\ R_5(x,y) \quad (22)$$

$$\vartheta = \exists x \in A \exists z \in M\ R_5(x,z) \quad (23)$$

Las partes arquitectónicas del marco orientado a objetos con arquitectura MVA (el modelo, la vista y el adaptador) pueden describirse formalmente como los subsistemas M, V y A, respectivamente, de manera que:

$M = \langle \boldsymbol{M}, R_1, R_2, R_3, R_4, R_5, R_6 \rangle$ tal que M es un subsistema de Q: $M = Q$ (24)

$V = \boldsymbol{V}, R_1, R_2, R_3, R_4, R_5, R_6$ tal que V es un subsistema de Q: $V = Q$ (25)

$A = \langle \boldsymbol{D}, \varsigma, \vartheta, R_1, R_2, R_3, R_4, R_5, R_6 \rangle$ tal que A es un subsistema de Q: $A = Q$ (26)

Como en el marco orientado a objetos, la configuración (ψ Q) representa las combinaciones específicas de clases y relaciones existentes en el marco MVA y se define como:

$$\psi Q = \{A \cup H \cup D \cup A_g \cup C_O \cup I\} \quad (27)$$

donde:

$$A = \{\langle x, y \rangle \in \boldsymbol{B} \mid x = y \vee x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat\ \alpha\} \quad (28)$$

$$H = \{\langle x, y \rangle \in \boldsymbol{B} \mid x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat \beta\} \quad (29)$$

$$D = \{\langle x, y \rangle \in \boldsymbol{B} \mid x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat\ \gamma\} \quad (30)$$

$$Ag = \{\langle x, y \rangle \in \boldsymbol{B} \mid x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat\ o\} \quad (31)$$

$$Co = \{\langle x, y \rangle \in \boldsymbol{B} \mid x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat\ \rho\} \quad (32)$$

$$I = \{\langle x, y \rangle \in \boldsymbol{B} \mid x \neq y\} = \{\langle x, y \rangle \in \boldsymbol{B} \mid Q[xy] sat\ \eta\} \quad (33)$$

La funcionalidad (δQ) se define como

$$\psi Q \rightarrow \delta Q \quad (34)$$

Descripción formal del esquema de solución

Sean P y Q sistemas relacionales, definidos en las dos secciones anteriores, respectivamente. Tenemos un esquema *T1* de solución si

$$T1(P) \rightarrow Q \text{ tal que } (\psi P \neq \psi Q) \wedge (\delta P = \delta Q) \quad (35)$$

En el cual, *T1* representa un conjunto de operaciones de reestructura.

Descripción del proceso de reestructura

Para integrar el conjunto de operaciones *T1*, se diseñó un proceso de reestructura para transformar la arquitectura original del marco orientado a objetos hacia una arquitectura MVA. El proceso consta de 11 métodos (R_1, …, R_{11}) que realizan 2 actividades principales y cada una se describe por su intención, precondiciones, procedimiento, suposiciones, poscondiciones e interacción de los métodos:

A1. Analizar el código del MOO original y etiquetar los enunciados, funciones y clases de acuerdo a las partes del patrón MVA a las que pertenecen

Intención: identificar a qué parte del patrón MVA corresponde cada uno de los enunciados de código, las funciones y las clases del MOO.

Precondiciones: tener acceso al código del conjunto de clases del MOO original y haber realizado la clasificación de los enunciados del lenguaje en que está escrito el MOO.

Procedimiento:

Paso A1.1 Analizar los enunciados, las funciones y las clases del MOO en estudio, con la finalidad de:

1. Identificar cada enunciado de código del MOO.
2. Etiquetar los elementos de código de acuerdo a la parte del patrón a la que pertenecen.
3. Generar información sobre las clases y las funciones del MOO en estudio, que apoyarán las siguientes actividades del proceso de reestructura.

Suposiciones: a partir de identificar y etiquetar los elementos de código, es posible llevar a cabo la reestructura del MOO en estudio.

Poscondiciones: los elementos del código del MOO en estudio quedan etiquetados de acuerdo a la parte del patrón MVA a la que pertenecen. Se obtiene información de tipos de clases, funciones abstractas, funciones concretas y tipos de variables.

Interacción de los métodos de reestructura: los pasos de esta actividad se llevan a cabo aplicando el método de reestructura R1 y no interacciona con otros métodos.

A2. Reestructurar el código legado

Intención: separar y reubicar los elementos de código del marco orientado a objetos original, de acuerdo a la parte del patrón que les corresponde.

Precondiciones: los enunciados de código, las funciones y las clases del marco orientado a objetos origi-

nal deben estar etiquetados de acuerdo a la parte del patrón MVA a la que pertenecen. Las tablas "Tipos de clases", "Funciones abstractas" y "Funciones concretas" deben estar creadas.

Pasos a seguir en el desarrollo de esta actividad:

Paso A2.1 Crear las tablas "Clases Modelo", "Clases Vista", "Clases Adaptador", "Funciones abstractas MVA", y "Funciones concretas MVA".

Paso A2.2 Obtener la jerarquía de clases del marco orientado a objetos original, con la finalidad de conservarla en la parte del modelo.

Paso A2.3 Siguiendo el orden de la jerarquía de clases del marco orientado a objetos original, crear las clases del nuevo marco orientado a objetos con arquitectura MVA. Este paso implica:

1. Crear las clases del modelo, la vista y el adaptador que se requieren (Paso A2.3.a, figura 4).

Paso A2.4 Si la clase en estudio es especializada en alguna parte del patrón MVA, reubicar a los miembros de la clase en la parte del patrón a la que pertenecen. Este paso implica:

1. Reubicar los constructores de la clase (Paso A2.4.a, figura 4).
2. Reubicar las funciones de la clase (Paso A2.4.b, figura 4).
3. Reubicar los atributos de la clase (Paso A2.4.c, figura 4).
4. Redefinir las llamadas a las funciones ya reubicadas que se encuentren en el código (Paso A2.4.d, figura 4).

Paso A2.5 Si la clase en estudio es No-especializada, fragmentar la clase para reubicar a sus elementos de código en la parte del patrón a la que pertenecen. Este paso implica:

1. Crear los constructores de las clases MVA (Paso A2.5.a, figura 4).
2. Crear nuevas funciones para reubicar los enunciados de código que pertenezcan a la vista y/o al adaptador (Paso A2.5.b, figura 4).
3. Reubicar los atributos a los que acceden las funciones reubicadas (Paso A2.5.c, figura 4).
4. Redefinir las llamadas a las nuevas funciones (Paso A2.5.d, figura 4).

Suposiciones: la parte del modelo mantiene la arquitectura del marco original y no tiene ningún elemento de la vista. La vista contiene solamente los elementos de código que pertenezcan a esta parte del patrón. El adaptador gestiona las interacciones entre el modelo y la vista.

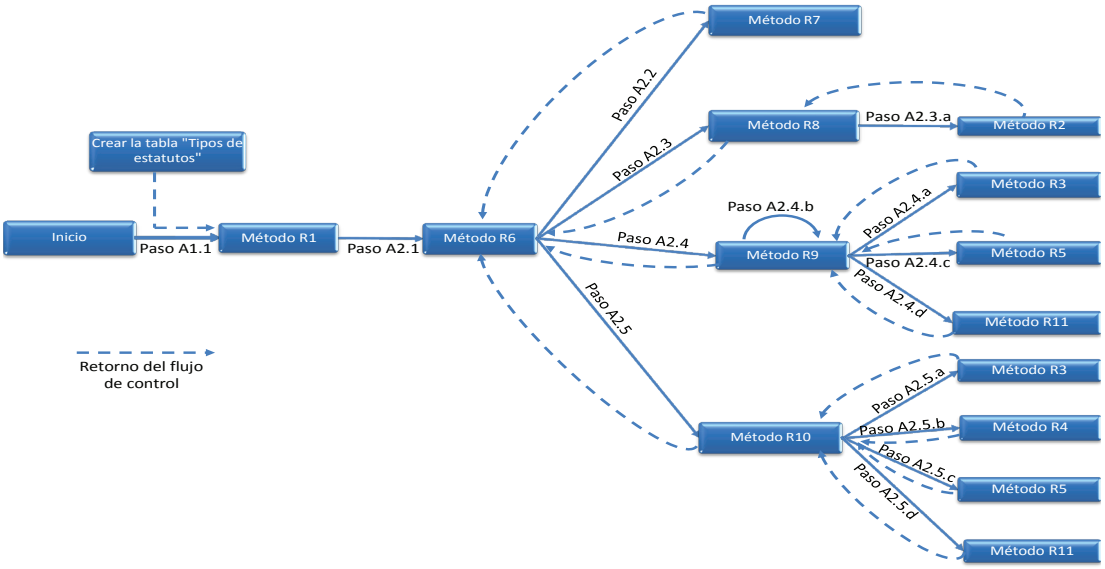

Figura 4. Proceso de reestructuración para obtener el marco orientado a objetos con arquitectura MVA

Poscondiciones: se obtiene un marco orientado a objetos con arquitectura MVA.

Interacción de los métodos de reestructura: después de analizar y etiquetar el código fuente del marco orientado a objetos original, se inicia la reestructuración de la arquitectura del marco, aplicando el método R6. Este método coordina el proceso de reestructuración llamando a los métodos R7, R8, R9 y R10.

La figura 4 muestra gráficamente este proceso, el cual termina cuando todas las clases del marco orientado a objetos original han sido reubicadas en la arquitectura MVA.

Resultados

Para realizar las pruebas al proceso de reestructuración para obtener un marco orientado a objetos con arquitectura MVA, se construyó un marco orientado a objetos del dominio de la geometría. De la arquitectura original de este marco, se seleccionó el caso práctico cuadrado-rectángulo cuya arquitectura se muestra en la figura 5.

El marco orientado a objetos cuadrado-rectángulo está formado por 7 clases (la clase cliente no es parte del marco orientado a objetos), que están relacionadas entre sí a través de relaciones de herencia. De acuerdo al modelo formal presentado en la sección de modelo del marco orientado a objetos, la descripción formal del marco orientado a objetos cuadrado-rectángulo se define como:

$H = \langle \mathbf{H}, R_3 \rangle$ donde:
$\mathbf{H} = \{\langle aFigura, aFiguraGeometrica, aPoligonos, aCuadrilatero, aParalelogramos, cCuadrado, cRectangulo \rangle\}$

La configuración (ψH) se describe formalmente como:

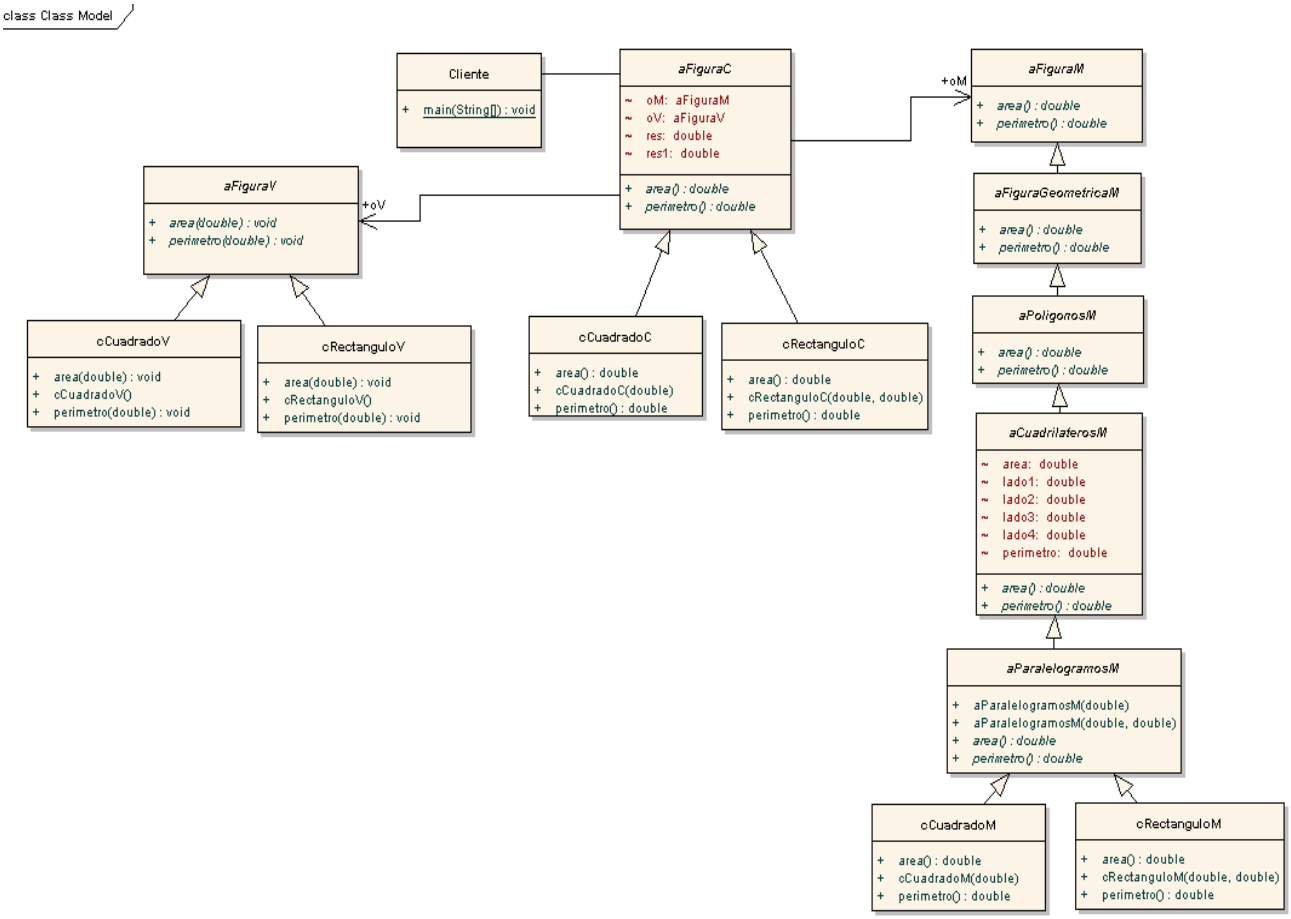

Figura 5. Arquitectura de clases del marco orientado a objetos cuadrado-rectángulo

$(\psi H) = \{\langle aFiguraGeometrica, aFigura, aPoligonos, aFiguraGeometrica, aCuadrilateros, aPoligonos, aParalelogramos, aCuadrilateros, cCuadrado, aParalelogramos, cRectangulo, aParalelogramos\rangle\}$

La funcionalidad del marco orientado a objetos cuadrado-rectángulo se describe formalmente como: $\psi H \rightarrow \delta H$.

Después de aplicar el conjunto de operaciones que integran el proceso de reestructura que se describe en la sección 5, al marco orientado a objetos cuadrado-rectángulo, se obtiene el marco orientado a objetos con arquitectura MVA (MVA cuadrado-rectángulo), como se muestra en la figura 6.

De acuerdo con el modelo formal presentado en la sección modelo del marco orientado a objetos con arquitectura MVA, el marco MVA cuadrado-rectángulo se describe formalmente como:

$\mathbf{I} = \langle \mathbf{I}, \varsigma, \vartheta, R_3 \rangle$ donde:

$\mathbf{I} = \{\langle aFiguraV, cCuadradoV, cRectanguloV, aFiguraC, cCuadradoC, cRectanguloC, aFiguraM, aFiguraGeometricaM, aPoligonosM, aCuadrilaterosM, aParalelogramosM, cCuadradoM, cRectanguloM\rangle\}$

$\varsigma = R_5\ (aFiguraC, aFiguraV)$
$\vartheta = R_5\ (aFiguraC, aFiguraM)$

La configuración (ψI) se describe formalmente como:

$\psi I = \{\langle aPoligonosM, aFiguraGeometricaM, cRectanguloV, aFiguraV, cCuadradoC, aFiguraC, cRectanguloC, aFiguraC, aFiguraGeometricaM, aFiguraM, cCuadradoV, aFiguraV, aCuadrilaterosM, aPoligonosM, aParalelogramosM, aCuadrilaterosM, cCuadradoM, aParalelogramosM, cRectanguloM, aParalelogramosM\rangle\}$

Una de las características que se debe cumplir para que un proceso de reestructura se considere exitoso, es que

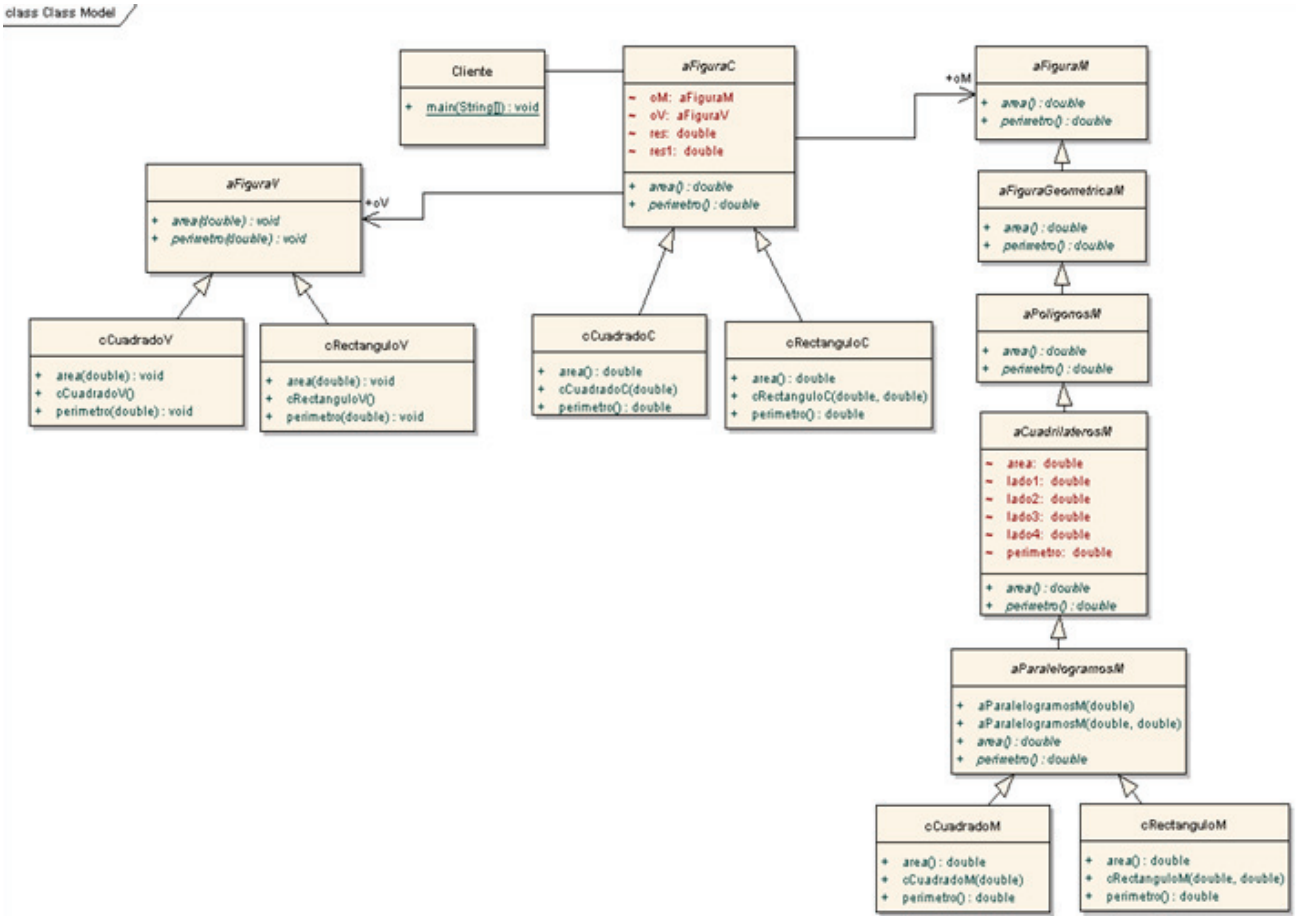

Figura 6. Arquitectura de clases MVA del caso práctico cuadrado-rectángulo

no se pierda la funcionalidad original ofrecida por el marco orientado a objetos. En la descripción formal del esquema de solución, que se presenta en la sección de descripción formal del esquema de solución, esta característica se establece en la restricción:

$$(\psi H \neq \psi l) \wedge \left(\delta H = \delta l\right) \tag{37}$$

Dos conjuntos son iguales sí y sólo sí tienen los mismos elementos (Hintze, 2009). Al analizar los conjuntos ΨH y Ψl observamos que se cumple ($\psi H \neq \Psi l$) debido a que la estructura arquitectónica de los marcos es diferente.

La expresión ($\delta H \neq \Psi$ l) establece que la funcionalidad de ambos marcos debe ser equivalente. Para verificar que esta condición se cumple se construyó una aplicación cliente con solicitudes de servicio similares para cada uno de los marcos. En las figura 7 se muestra el código de estas aplicaciones.

a)
```
public class Cliente
{  public static void main(String[] args)
   {  aFigura c=new cCuadrado(5);
      aFigura r=new cRectangulo(3, 9);
      System.out.println("CUADRADO");
      c.area();
      System.out.println("RECTANGULO");
      r.area();
   }
}
```

b)
```
public class Cliente
{  public static void main(String[] args)
   {  aFiguraC oC=new cCuadradoC(5);
      aFiguraC oR=new cRectanguloC(3, 9);
      System.out.println("CUADRADO");
      oC.area();
      System.out.println("RECTANGULO");
      oR.area();
   }
}
```

Figura 7. Clases Cliente a) para el caso práctico Cuadrado-Rectángulo original y b) para el caso práctico MVA Cuadrado-Rectángulo

Los resultados obtenidos al ejecutar las aplicaciones cliente se presentan en las figuras 8 y 9.

```
CUADRADO
El area del cuadrado es: 25.0
RECTANGULO
El area del rectangulo es: 27.0
```

Figura 8. Resultados de la ejecución de la aplicación Cliente para el caso práctico Cuadrado-Rectángulo original

```
CUADRADO
El area del cuadrado es: 25.0
RECTANGULO
El area del rectangulo es: 27.0
```

Figura 9. Resultados de la ejecución de la aplicación Cliente para el caso práctico MVA Cuadrado-Rectángulo

El resultado de la ejecución de las aplicaciones *Cliente* permite observar que existe una equivalencia funcional entre ambos marcos. La restricción ($\delta H \neq \delta$ l), se cumple para los casos prácticos presentados.

Trabajos relacionados

La migración de *software* legado hacia nuevas plataformas tecnológicas es una de las propuestas de solución más utilizadas para aumentar el nivel de reuso y el mantenimiento de este tipo de código.

Sin embargo, los trabajos que se relacionan más cercanamente con el trabajo que se presenta en este documento son aquéllos que implementan un proceso de reingeniería basado en la reestructuración arquitectónica del código legado hacia una arquitectura de capas (MVC o MVA), como una de las fases o etapas del proceso de migración. Estos trabajos consideran la separación del código de la lógica del negocio (el modelo, donde es posible encontrar más funcionalidades que pueden ser reutilizables) del código de la lógica de presentación (la vista).

En el trabajo que se describe en (Matos y Heckel, 2009), los autores presentan una metodología que se enfoca en el análisis del código fuente para migrar sistemas legados hacia arquitecturas orientadas a servicios. La metodología propuesta transforma la arquitectura del software legado hacia una arquitectura de tres capas que separa la lógica de la aplicación, los datos y la interfaz de usuario; y posteriormente, realiza una descomposición funcional que permite obtener componentes candidatos a servicios web.

El trabajo que se describe en las secciones anteriores, también se enfoca en el análisis del código fuente para realizar la transformación arquitectónica de un marco orientado a objetos hacia una arquitectura de tres capas, aunque difiere del trabajo que se describe en (Matos y Heckel, 2009) en el modelo arquitectónico obtenido, la metodología y las técnicas empleadas para realizar el proceso de reestructuración.

En (Hunold *et al*., 2008), los autores presentan un conjunto de herramientas de ingeniería que dan soporte a un proceso incremental de transformación arquitectónica. Este proceso separa el *software* legado original en varios componentes independientes y reemplaza-

bles, que facilitan la migración del código legado hacia nuevas plataformas.

El proceso de transformación arquitectónica que se describe en Hunold *et al*. (2008) coincide con el proceso de reestructuración que se describe en este trabajo, al considerar el análisis y la categorización del código fuente legado como la base del proceso de transformación. Las diferencias entre ambos trabajos se encuentran en el modelo arquitectónico obtenido, la metodología y las técnicas empleadas para llevar a cabo la reestructuración.

En Pahl y Barret (2004), los autores describen un conjunto de requerimientos básicos para el diseño y la implementación de un proceso de reingeniería de sistemas legados distribuidos y/o basados en el Web, hacia arquitecturas orientadas a servicios. Entre estos requerimientos, los autores plantean la reestructuración arquitectónica del sistema legado hacia una arquitectura de 3 capas (implementada de acuerdo al patrón MVC), como una de las fases que se deben considerar en el diseño del proceso de reingeniería. Este planteamiento concuerda con el enfoque del proceso de reestructuración que se describe en este documento.

Finalmente, en el trabajo que se presenta en Ping *et al*. (2004) los autores describen un marco de reingeniería, cuyo objetivo es transformar la arquitectura de aplicaciones Web legadas hacia una arquitectura basada en el patrón MVC.

Concretamente, el proceso de reingeniería propuesto por los autores consiste en la implementación de 3 fases o etapas, en las cuales se analiza el código de la aplicación Web legada para separar los elementos que pertenecen al modelo (accesos y consultas a la base de datos), a la vista (elementos de código escrito en HTML y páginas JSP) y el control (archivos HTML, páginas JSP generadas y objetos Java Beans). La similitud con el trabajo que se describe en este documento se observa en la realización del análisis del código legado como una actividad básica del proceso de reingeniería; así como en la obtención de una nueva arquitectura organizada en los componentes del patrón MVC.

Conclusiones

En la literatura especializada se han presentado trabajos que demuestran que la transformación arquitectónica del código legado hacia una arquitectura de capas permite separar la lógica de presentación, de la lógica del negocio y puede considerarse como una estrategia que facilita la transición del código legado hacia nuevas plataformas tecnológicas.

El proceso de reestructuración que se describió en este documento presenta las siguientes ventajas:

- Permite desacoplar los elementos del marco orientado a objetos sin perder la funcionalidad original.
- Al finalizar el proceso de reestructuración se obtiene un marco orientado a objetos con arquitectura MVA completamente funcional.
- El código del marco queda ordenado y preparado para migrarlo hacia nuevas plataformas tecnológicas o facilitar su mantenimiento.

Un punto importante que se debe considerar al aplicar el proceso de reestructuración propuesto es que existen enunciados del código fuente del marco orientado a objetos que no son fáciles de clasificar, de acuerdo con la parte arquitectónica del patrón MVA a la que corresponden. Tal es el caso de los enunciados de decisión, los enunciados repetitivos, los enunciados de asignación, que pueden utilizarse en las diferentes partes arquitectónicas del patrón MVA y cuya clasificación requiere un análisis más profundo que la simple identificación.

Debido a que el procedimiento de identificación y clasificación de enunciados es la base del análisis y reestructura del código, esta fase del proceso requiere la participación de un experto en el dominio y en el lenguaje.

Sin embargo, los resultados obtenidos al aplicar el proceso de reestructuración nos permiten concluir que representa una estrategia útil que facilita la migración de marcos orientados a objetos hacia nuevas plataformas tecnológicas.

Actualmente, se continúa trabajando en el diseño e implementación de una herramienta de ingeniería que automatice el proceso reestructuración.

Referencias

Froehlich G., Hoover H.J., Liu L., Sorenson P. *Designing Object-Oriented Frameworks, Department of Computing Science*, Canadá, University of Alberta, Edmonton, noviembre 1998.

Hintze B. Bungee Connect and Model-View-Adapter (MVA): The Architectural Pattern for Next Generation Interactive Cloud-Based Web Applications, Bungee Labs, 2009.

Hunold S., Korch M., Krellner B., Rauber T., Reichel T., Rünger G. Transformation of Legacy Software into Client/Server Applications through Pattern-based Rearchitecturing, en: Computer Software and Applications (COMPSAC '08), 32nd Annual IEEE International, 2008.

International Standard ISO/IEC 9126-1. Software Engineering Product Quality, Part 1, Quality Model.

Matos C. Heckel R. Migrating Legacy Systems to Services-Oriented Architectures. *Electronic Comunications of the EASST*, volumen 16, 2009.

Pahl C., Barrett R. Towards a Re-Engineering Method for Web Services Architectures. *Journal of Mathematical Modelling in Physics, Engineering and Cognitive Sciences*, volumen 11, diciembre 2004.

Ping Y., Kontogiannis K., Lau T.C. Transforming Legacy Web Applicattions to the MVC Architecture, en: Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'04), IEEE, 2004.

Rosen K. *Discrete Mathematics and its Applications*, 6a ed., McGrawHill International, 2007, p.113.

Rumbaugh J., Jacobson I., Booch G. *The Unified Modeling Language Reference Manual*, Adisson-Wesley, 1999.

Santaolaya R. *Modelo de representación de patrones de código para la construcción de componentes reusables,* tesis (doctorado), Departamento de Ciencias Computacionales, Centro de Investigación en Computación, Instituto Politécnico Nacional, 2003.

Este artículo se cita:

Citación estilo Chicago

Santaolaya-Salgado, René, Olivia Graciela Fragoso-Díaz, Sheydi Anel Zamudio-López. Modelo formal para la reestructura de marcos orientados a objetos hacia arquitecturas modelo-vista-adaptador. *Ingeniería Investigación y Tecnología*, XV, 02 (2014): 187-198.

Citación estilo ISO 690

Santaolaya-Salgado R., Fragoso-Díaz O.G., Zamudio-López S.A. Modelo formal para la reestructura de marcos orientados a objetos hacia arquitecturas modelo-vista-adaptador. *Ingeniería Investigación y Tecnología*, volumen XV (número 2), abril-junio 2014: 187-198.

Semblanza de los autores

René Santaolaya-Salgado. Es profesor investigador en el área de ingeniería de Software, del Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET). Obtuvo el grado de doctor en ciencias de la computación, por el Centro de Investigación en Computación del Instituto Politécnico Nacional y es miembro del Sistema Nacional de Investigadores. Su área de interés es la ingeniería de Software, específicamente en ambientes integrados para el desarrollo de sistemas, programación visual, reusabilidad del Software y servicios Web.

Olivia Graciela Fragoso-Díaz. Es profesora investigadora en el área de ingeniería de Software, del Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET). Obtuvo el grado de doctor en ciencias de la computación, por el Centro Nacional de Investigación y Desarrollo Tecnológico. Tiene el grado de maestría en ciencias computacionales por el Instituto de Ciencia y Tecnología de la Universidad de Manchester (UMIST-UK). Su área de interés es la ingeniería de Software, específicamente, reingeniería, reusabilidad de Software y servicios Web.

Sheydi Anel Zamudio-López. Es profesora del Sistema Nacional de Institutos Tecnológicos, en el Instituto Tecnológico de Nuevo León. Obtuvo el grado de maestría en ciencias computacionales por el Centro Nacional de Investigación y Desarrollo Tecnológico (CENIDET) y actualmente está inscrita en el programa doctoral en ciencias de la computación que imparte el CENIDET. Su área de interés es la ingeniería de Software, específicamente el análisis y diseño de metodologías de reuso de software, patrones de diseño y desarrollo de proyectos de Software.