Cómo generar interfaces gráficas de usuarios en aplicaciones web usando Arquitectura Dirigida por Modelos

How to generate graphical user interfaces in web applications using model-driven architecture

Efraín Francisco Ruíz Zamora

Nemury Silega Martínez

Liliana Simón Figueredo

Universidad de las Ciencias Informáticas. La Habana. Cuba

Universidad de las Ciencias Informáticas. La Habana. Cuba

Universidad de las Ciencias Informáticas. La Habana. Cuba

Resumen

En los centros de desarrollo se buscan constantemente ideas que agilicen el proceso de desarrollo de software. Las herramientas de generación de código juegan un papel primordial en la producción de software, pues permiten agilizar el desarrollo de un producto y disminuir las posibilidades de errores en su elaboración; pero estas herramientas se basan en una única tecnología de desarrollo, por lo que son prácticamente inadaptables a las nuevas tecnologías. En este trabajo se propone la construcción de una herramienta sencilla, práctica y adaptable a los nuevos cambios tecnológicos; mediante la cual se puede generar el código fuente para construir las interfaces gráficas de usuario de las aplicaciones web de gestión a partir de una clase entidad que toma como entrada. La aplicación se desarrolló usando la Arquitectura Dirigida por Modelos y se utilizó el Lenguaje de Transformación Atlas para las trasformaciones de los metamodelos de orígenes a destinos; esta aplicación permite disminuir el tiempo de desarrollo de las interfaces gráficas.

Palabras clave: generación de código, interfaces gráficas, metamodelos, transformaciones

Abstract

In the development centers are search constantly ideas to speed up the software development process. The tools of generation of code play a fundamental role in the production of software, because they allow to speed up the development of a product and to diminish the possibilities of errors in its elaboration; but these tools are based on a one technology of development, so they are virtually unadaptable to new technologies. In this work propose the construction of a simple, practical and adaptable tool to the new technological changes; by means of which the source code can be generated to construct the graphical user interfaces of the web applications of management from an entity class that takes as input. The application was developed using the Model-Driven Architecture and was used the Atlas Transformation Language for the transformations of metamodels from origins to destinations; this application allows to decrease the development time of graphic interfaces.

Keywords: *code generation, graphical interfaces, metamodels, transformations*

Introducción

Las empresas de desarrollo deben tratar de producir software de calidad a un costo y en un tiempo adecuado; un reto difícil de lograr si se tiene en cuenta que las demandas de los clientes en muchos casos son superiores a las capacidades productivas de las empresas y que la producción de software de forma industrial es solo un mito que pocas entidades pueden respaldar (Tarí Guilló, 2000)(ruiz zamora, y otros, 2015), además de que la mala gestión del costo y el tiempo son dos de los principales motivos del fracaso de proyectos informáticos (Díaz Piraquive, y otros, 2015) (Guerra y Bedini González, 2003). Otro de los retos con los que deben lidiar estas empresas es con los constantes cambios tecnológicos, ya sea por las exigencias de los propios clientes o para poder utilizar herramientas que solo soportan estas nuevas tecnologías (Reina, y otros, 2004). Para dar solución a estos cambios han surgido propuestas novedosas entre las que se encuentra el Desarrollo Dirigido por Modelos (MDD, por sus siglas en inglés), la que permite minimizar los costes de adaptación de nuevos en-tornos tecnológicos y generar código fuente a partir de la transformación de uno o varios modelos.

Dentro del proceso de desarrollo de software la interfaz gráfica juega un papel primordial, ya que es con la que interactúa el usuario final del sistema (Guzmán Ojeda, y otros, 2013). Las interfaces gráficas de usuario son decisivas en el desempeño de cualquier aplicación informática, pues más del 80% de los sistemas fallan por problemas de usabilidad (Ujueta, 2017). Además, el desarrollo de interfaces gráficas constituye una tarea exhaustiva y compleja pues comprende el 60% de las líneas de código del total y el 50% promedio del tiempo de implementación de un sistema de información interactivo (Albornoz, y otros, 2013). Para solventar o minimizar el efecto de estas condiciones características en los softwares se han utilizado herramientas generadoras de código, enfocadas al desarrollo de las interfaces gráficas, pero estas herramientas son dependientes de una única tecnología por lo que con el transcurso del tiempo y el desarrollo tecnológico se vuelven en gran medida inutilizable. Otra vía, aunque menos abordada, que se ha empleado para contrarrestar dichas situaciones es a partir del Desarrollo Dirigido por Modelos, emplear la Arquitectura Dirigida por Modelos (MDA, por sus siglas en inglés). En la literatura se documenta

los alentadores beneficios que ha generado la adopción de propuestas basadas en diversos dominios (de Almeida Monte-Mor, y otros, 2011) (Silega, Loureiro y Noguera, 2014).

El presente trabajo aborda el desarrollo de una aplicación informática adaptable a los constantes cambios tecnológicos, guiado sobre el enfoque de la Arquitectura Dirigida por Modelos, que permite la generación de código para disminuir el tiempo de desarrollo de interfaces gráficas de usuario en las aplicaciones web de gestión, a partir de las transformaciones de modelo-modelo y modelo-código con el uso de las reglas de transformación que ofrece el lenguaje ATL. El efecto del método fue demostrado a partir de un pre-experimento realizado con un solo grupo.

Materiales y métodos

Arquitectura Dirigida por Modelos (MDA)

MDA es una propuesta de MDD definida por el Object Management Group (OMG). Separa la aplicación y la lógica de negocio de los aspectos tecnológicos (de Almeida Monte-Mor, y otros, 2011). Promueve el uso del Lenguaje de Modelado Unificado (UML, por sus siglas en inglés) para visualizar, almacenar e intercambiar diseños y modelos de software. Promueve la creación de modelos muy abstractos y legibles por máquina que se desarrollan independientemente de la tecnología de implementación y se almacenan en repositorios estandarizados (Kleppe, y otros, 2003). Basa su funcionalidad en tres modelos (Miller, y otros, 2003). Un Modelo Computacional Independiente (CIM, por sus siglas en inglés) que es una vista del sistema que no muestra detalles de su estructura y se le puede conocer como el modelo del dominio (Pons, y otros, 2010). Luego, se define un Modelo Independiente de la Plataforma (PIM, por sus siglas en inglés) a través de un lenguaje específico para el dominio en cuestión e independiente de cualquier tecnología. El modelo PIM puede traducirse a uno o más Modelos Específicos de la Plataforma (PSM, por sus siglas en inglés). Los PSM, pueden estar basados en lenguajes específicos del dominio o lenguajes de propósito general. Luego de definir cada uno de estos modelos, se realiza la implementación del código fuente a partir de cada uno de los PSM desarrollados (Texier, y otros, 2012).

Atlas Transformation Language (ATL)

Es un lenguaje de transformación de modelos desarrollado por el grupo Atlas sobre la plataforma Eclipse. ATL proporciona formas de producir un conjunto de modelos de destino a partir de un conjunto de modelos de origen y tienen como objetivo facilitar el desarrollo de las transformaciones ATL (ATLAS group, 2005). Es un lenguaje de transformación híbrido. Contiene una mezcla de construcciones declarativas e imperativas. El estilo animado es declarativo. Las transformaciones de ATL son unidireccionales, operan en modelos fuente de solo lectura y producen modelos de destino solo de escritura. Los modelos de origen y destino para ATL se pueden expresar en el formato de serialización XMI OMG (Jouault, y otros, 2005).

Herramienta de soporte

La aplicación se implementó siguiendo el enfoque de MDA, a partir del diseño de metamodelos que se pueden transformar en uno o varios metamodelos. Para el desarrollo de la aplicación se utilizó el Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) Eclipse; se escogió esta plataforma porque permite

la inclusión de un plugin para el desarrollo con el lenguaje ATL y otro para el diseño de metamodelos (EMF). ATL brinda la posibilidad de implementar un conjunto de reglas y funciones de ayudas (helpers) que facilitan las transformaciones de un metamodelo a otro, por eso se decidió utilizar este lenguaje.

En el trabajo se parte de un fichero XMI, que se obtiene a partir de la extracción mediante el uso de un algoritmo de los datos de una entidad del negocio, en este caso una entidad PHP generada a partir del Mapeo Relacional de Objetos (ORM, por sus siglas en inglés) Doctrine. Este fichero sirve como entrada para el metamodelo que representa a la entidad. Luego se transforma mediante las reglas de transformación de ATL en otro metamodelo que representa a la vista genérica de un CRUD de una aplicación web de gestión, sin tener en cuenta las especificidades de ningún lenguaje de programación para la creación de las interfaces gráficas. Posteriormente se transforma usando igualmente un conjunto de reglas de ATL en un metamodelo que representa el desarrollo de una vista de un CRUD ya en un lenguaje específico, para este caso EXT JS 4. Entonces la salida de este último meta-modelo, que es un fichero XMI, es usado para extraer de él los valores de las etiquetas y convertirlos con la ayuda de un algoritmo en el código fuente. El código quedará desglosado en diferentes ficheros que corresponden al desarrollo de las interfaces gráficas de usuario para la aplicación web de gestión en cuestión, que expresa a la entidad tomada como entrada.

Validación de la investigación

Para la validación de la presente investigación se realizó un pre-experimento, mediante el diseño de pre-prueba y post-prueba con un solo grupo; para comprobar cómo se comportaba el tiempo de desarrollo de las interfaces gráficas, analizando un antes (sin el uso de la herramienta) y un después (con el apoyo de la herramienta).

Resultados y discusión

Para la puesta en práctica de la nueva herramienta generadora de código se tomó como entrada una entidad generada por Doctrine 2.0 y se realizaron las transformaciones necesarias mediante el conjunto de reglas de ATL para transformar de un metamodelo a otro y luego generar el código fuente de las interfaces de un CRUD en EXT JS 4.

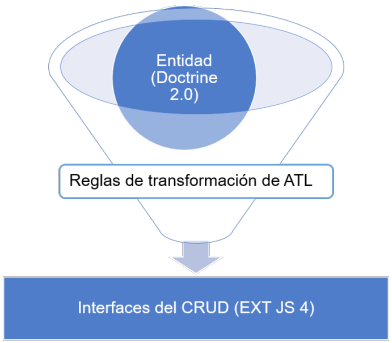

Figura 1. Entrada y salida de la herramienta generadora de código

En la estructura de una entidad generada por Doctrine 2.0, los valores que son utilizados por la herramienta desarrollada son los nombres de los atributos y sus tipos de datos. Estos valores se representan en la entidad dentro de las anotaciones del propio ORM como se puede observar en la línea de código señalada en la figura que se muestra a continuación. En este caso se utilizó como ejemplo una entidad nombrada "Empresa" con los atributos ruc, descripción, actividad, teléfono y dirección.

```php
<?php
namespace AppBundle\Entity;
use Doctrine\ORM\Mapping as ORM;
/**
 * Empresa
 * @ORM\Table(name="empresa")
 * @ORM\Entity(repositoryClass="AppBundle\Repository\EmpresaRepository")
 */
class Empresa
{
    /**
     * @var int
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;
    /**
     * @var int
     * @ORM\Column(name="ruc", type="integer")
     */
    private $ruc;
    /**
     * @var string
     * @ORM\Column(name="descripcion", type="string", length=255)
     */
    private $descripcion;
    /**
     * @var string
     * @ORM\Column(name="actividad", type="string", length=255)
     */
    private $actividad;
    /**
     * @var int
     * @ORM\Column(name="telefono", type="integer")
     */
    private $telefono;
    /**
     * @var string
     * @ORM\Column(name="direccion", type="string", length=255)
     */
    private $direccion;
```

Figura 2. Estructura de entidad generada por Doctrine 2.0

Mediante el algoritmo de extracción de datos desarrollado se obtiene el fichero XMI a partir de la entidad PHP, este fichero se utiliza para cargar el metamodelo que representa a la entidad. El algoritmo implementado convierte la entidad en una cadena de texto, luego la descompone en tokens mediante la función de PHP "token_get_all" y posteriormente extrae de la misma los atributos y tipos de datos que en ella aparecen.

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns="entidades">
  <entidad nombre_entidad="Empresa">
    <atributos nombre_atributo="ruc" tipo_dato_atributo="int" complejo="false"
    exp_regular="" vinculado="null" />
    <atributos nombre_atributo="descripcion" tipo_dato_atributo="string" complejo="false"
    exp_regular="" vinculado="null" />
    <atributos nombre_atributo="actividad" tipo_dato_atributo="string" complejo="false"
    exp_regular="" vinculado="null" />
    <atributos nombre_atributo="telefono" tipo_dato_atributo="int" complejo="false"
    exp_regular="/^([0-9]{5})+((-{1})*)+([0-9]{6})$/" vinculado="null" />
    <atributos nombre_atributo="direccion" tipo_dato_atributo="string" complejo="false"
    exp_regular="" vinculado="null" />
  </entidad>
</xmi:XMI>
```

Figura 3. Fichero XMI obtenido de la entidad

Basado en MDA se definieron 3 metamodelos, uno que representa a la entidad del negocio generada por Doctrine 2.0 y que es el utilizado como entrada de las transformaciones, otro metamodelo PIM que representa a la vista genérica de un CRUD para una aplicación web de gestión y uno PSM que representa a la vista del CRUD para el lenguaje específico EXT JS 4 en este caso y del que se extraen los valores para

transformarlos en el código fuente de las interfaces gráficas; pudiéndose representar tantos metamodelos PSM como lenguajes de desarrollo de interfaces se quieran generar. Los metamodelos identificados y diseñados se muestran a continuación.

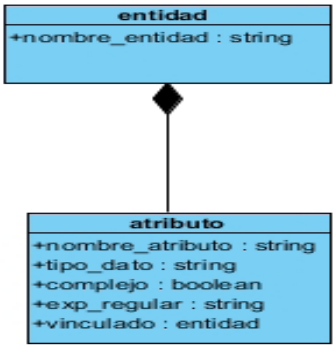

Figura 4. Metamodelo "entidad", representa a la entidad del negocio

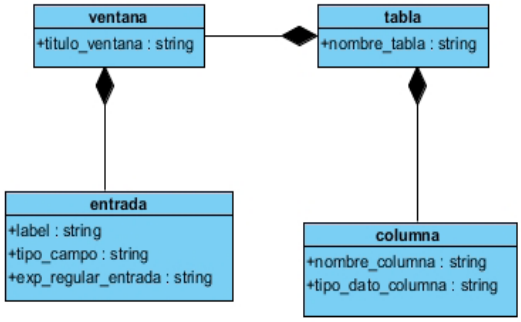

Figura 5. Metamodelo "vista", representa la vista genérica

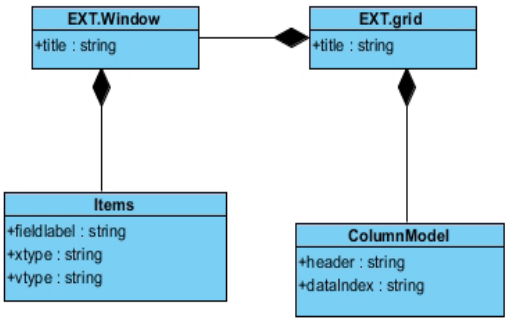

Figura 6. Metamodelo "vista_EXTJS", representa la vista para el lenguaje EXT JS

Una vez diseñado los metamodelos se transforman los modelos usando un conjunto de reglas que se definieron usando el lenguaje ATL. Se implementó un fichero ATL para cada una de las transformaciones de un metamodelo a otro, en cada uno de estos ficheros se especifica el metamodelo origen y el destino, además del conjunto de reglas de transformación. A continuación, se muestra una de las reglas definidas en el fichero ATL que tiene como origen y destino a los metamodelos "entidad" y "vista" respectivamente. Esta regla representa las transformaciones de los atributos de la entidad del negocio en columnas de la tabla de la vista genérica.

```
rule atributos2columnas {
    from
        s : entidad!atributo
    to
        t : vista!columnas (
            nombre_columna <- s.nombre_atributo,
            tipo_dato_columna <- s.tipo_dato_atributo
        )
}
```

Figura 7. Regla de transformación de los atributos de la entidad en columnas de la vista

Después de aplicar todas las reglas de transformación de cada uno de los ficheros ATL la herramienta utiliza el XMI de salida que representa al metamodelo "vista_EXTJS", para generar a partir de este el código fuente. Para la generación de código se parte de la extracción de los valores de las etiquetas del fichero XMI y con estos elementos (nombre de la entidad, nombre de los atributos y sus tipos de datos, así como las expresiones regulares (opcionales)) se configuran los parámetros del CRUD, para finalmente generar las interfaces gráficas de usuario en EXT JS 4. En la siguiente figura se muestra como quedarían las interfaces gráficas para la entidad "Empresa" utilizada.

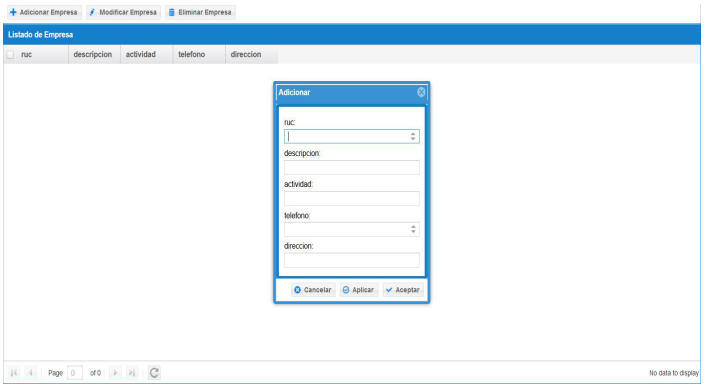

Figura 8. Interfaces gráficas de un CRUD en EXT JS 4 desarrolladas por la herramienta generadora de código

Estas interfaces son guardadas en la estructura de carpetas que requiere el marco de trabajo EXT JS en la versión 4 o superior (estructura MVC), donde la carpeta raíz es el nombre de la entidad y se desglosa de la siguiente forma: un archivo principal "app.js" y cuatro carpetas: "controller", "model", "store" y "view", en las cuales se guardan los controladores, modelos, almacenes y vistas respectivamente; como se observa a continuación.

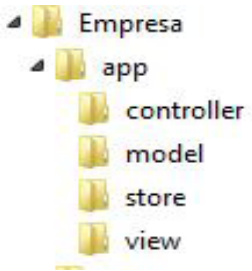

Figura 9. Estructura de carpetas en la que se desglosa el código fuente generado

Para la validación de la investigación se realizó un pre-experimento, teniendo en cuenta para la pre-prueba los resultados obtenidos de una encuesta aplicada a una muestra de 30 programadores, que trabajan o han trabajado con el marco de trabajo EXT JS 4, del Centro de Informatización de Entidades (CEIGE) de la Universidad de las Ciencias Informáticas para medir los tiempos promedios que se demoraban los desarrolladores que utilizan este marco de trabajo para la creación de las interfaces gráficas de un CRUD en una aplicación web de gestión.

La post-prueba fue aplicada a la misma muestra de desarrolladores a la que se le realizó la encuesta; esta prueba consistió en el uso de la herramienta desarrollada, una vez que les fue explicado a cada uno como funcionaba, con el objetivo de medir el tiempo que se demoraban en desarrollar las interfaces con el uso de la herramienta. Para la prueba se tomó como entrada una de las entidades del negocio de su proyecto seleccionada aleatoriamente y el tiempo fue medido en el proceso completo de creación de las interfaces con la aplicación.

Como resultado del pre-experimento se obtuvieron los resultados que se muestran a continuación. Estos resultados expresan el tiempo para el mejor y el peor de los casos en las pruebas desarrolladas antes y después de la implementación de la herramienta generadora de código.

Con la realización del pre-experimento se comprobó como el tiempo de desarrollo de las interfaces gráficas disminuyó marcadamente con el uso de la herramienta desarrollada bajo el enfoque de la arquitectura MDA. Además, con el uso de esta herramienta se garantiza la homogeneidad dentro de las aplicaciones web de gestión del mismo proyecto y se disminuyen los errores humanos que puedan surgir dentro del proceso de desarrollo, al crearse las interfaces gráficas de forma automática. Esta herramienta se adapta de manera fácil ante los cambios de tecnología, pues no es necesario desarrollar completamente las interfaces, solo modificar el metamodelo referente al cambio de lenguaje; además la reutilización se favorece porque podría generarse a partir del mismo modelo origen el código de las interfaces en varios lenguajes de programación.

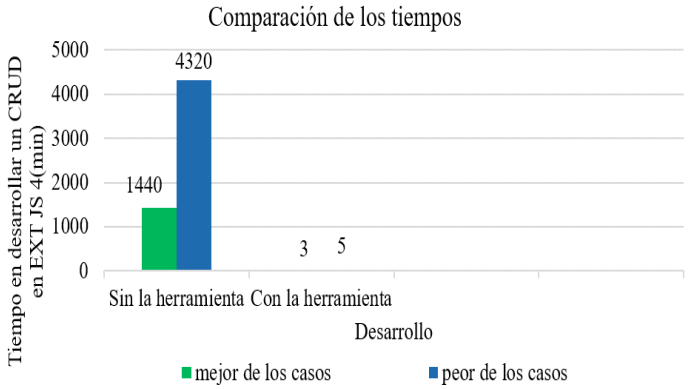

Figura 10. Resultados del pre-experimento realizado

Conclusiones

La aplicación desarrollada se basa en la definición de metamodelos; estos metamodelos son transformados a partir de reglas de transformación del lenguaje ATL de un metamodelo a otro y luego son llevados a código fuente. La misma permite disminuir el tiempo de implementación de las interfaces gráficas de usuario de un CRUD en las aplicaciones web de gestión en los centros de desarrollo. Esta herramienta es adaptable a nuevas versiones de las tecnologías existentes y a la aparición de nuevas tecnologías, para ello solo es necesario incluir nuevos metamodelos PSM que representen estos nuevos cambios y definir el nuevo fichero ATL con el conjunto de reglas de transformación del metamodelo PIM que representa la vista genérica.

Referencias

Albornoz, M. C., Miranda, E. y Berón, M. (2013). Evaluación de Interfaces Gráficas de Usuario usando LSP. Universidad Nacional de San Luis, San Luis, Argentina.

ATLAS group. (2005). ATL: Atlas Transformation Language. LINA & INRIA.

de Almeida Monte-Mor, J., Oliveira Ferreira, E., Fernandes Campos, H., Marques da Cunha, A. y Vieira Dias, L. A. (2011). Applying MDA Approach to Create Graphical User Interfaces. Las Vegas, Nevada, USA. Obtenido de https://www.researchgate.net/publication/220841584_Applying_MDA_Approach_to_Create_Graphical_User_Interfaces

Díaz Piraquive, F. N., Medina García, V. H., González Crespo, R. y Pérez Castillo, J. N. (2015). Motivos de fracaso en los proyectos de Tecnologías de Información y Comunicaciones. Conference: 13th LACCEI Annual International Conference. Santo Domingo, República Dominicana: ResearchGate. Obtenido de https://www.researchgate.net/publication/282647169_Motivos_de_fracaso_en_los_proyectos_de_Tecnologias_de_Informacion_y_Comunicaciones

Guerra, L. y Bedini González, A. (2003). Gestión de Proyectos de Software.

Guzmán Ojeda, J. R. y Betancourt Santana, R. (2013). "Biblioteca JavaScript para el desarrollo de interfaces gráfi-

cas de usuario de RIA". Universidad de las Ciencias Informáticas, FACULTAD 6, La Habana. Cuba.

Jouault, F. y Allilaire, F. (2005). The Atlas Transformation Language (ATL) project. University of Nantes. ATL Eclipse.

Kleppe, A., Warmer, J. y Bast, W. (2003). MDA Explained. The Model Driven Architecture: Practice and Promise. Addison Wesley. Obtenido de www.awprofessional.com/titles/032119442X/

Miller, J. y Mukerji, J. (2003). MDA Guide Version 1.0.1. OMG.

Pons, C., Giandini, R. y Pérez, G. (2010). Desarrollo de Software Dirigido por Modelos. La Plata, Argentina: Mc Graw Hill.

Reina, A. M., Torres, J., Toro, M. y Álvarez, J. A. (2004). Separación de conceptos y MDA: Arquitectura de un framework. Universidad de Sevilla, Departamento de Lenguajes y Sistemas Informáticos, Sevilla, España.

Ruíz Zamora, E. F., Laza Tarafa, A. y Simón Figueredo, L. (2015). Generación de código Javascript para construir interfaces gráficas en EXT JS 4 para aplicaciones web de gestión. Universidad de las Ciencias Informáticas. La Habana: 10ma Peña Tecnológica.

Silega, N., Loureiro, T. y Noguera, M. (2014). Model-Driven and Ontology-Based Framework for Semantic Description and Validation of Business Processes. 12(2).

Tarí Guilló, J. J. (2000). Calidad total: fuente de ventaja competitiva. Alicante: Publicaciones Universidad de Alicante.

Texier, J., De Giusti, M., Oviedo, N., Villarreal, G. L. y Lira, A. (2012). Los Beneficios del Desarrollo Dirigido por Modelos en los Repositorios Institucionales. E-LIS repository. Obtenido de eprints.rclis.org/17970/1/Texier_Beneficios.pdf

Ujueta, R. E. (2017). BBR Capital. Obtenido de "Usabilidad": Éxito o fracaso del emprendimiento tecnológico: http://bbrcapital.com/usabilidad-software/