INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE COMPUTO
SUBDIRECCIÓN ACADÉMICA

No. de Serie: TT0291 Serie: Amarilla 13-Mayo-2002
Proyecto Terminal

"HERRAMIENTA DE GENERACIÓN DE CÓDIGO A PARTIR DE PATRONES DE DISEÑO O. O."

Garcés Rodríguez Benito Joel
Edif. 93-2 U.H. La Patera Vallejo
Delegación Gustavo A. Madero, D.F.
C.P. 07710, Tel. 53686823
benitogr@servidor.unam.mx

Martínez García David Alejandro
Cuicatecas # 25 Col. Tezozomoc
Delegación Atzcapotzalco, D.F.
C.P. 02450, Tel. 53837490
damtz@prodigy.net.mx

Morales Durán Carlos David
Cóporo # 31-516 Fraccionamiento Rincón de los Bosques
Atizapán, Estado de México
C.P. 52967, Tel. 58240351
skatovato@hotmail.com

Sánchez Santana Alejandra
Gráfico # 1127 Col. Prensa Nacional
Tlalnepantla, Estado de México
C.P. 54170, Tel. 53681719
carlejandra@go.com

RESUMEN

El paradigma orientado a objetos ha probado ya con creces sus bondades. Al construir y diseñar bajo este paradigma, se pueden modelar los sistemas de una manera más clara, concreta y lógica. Buenos diseños, conducen a sistemas implementados de mejores características. Los sistemas OO están construidos primordialmente por *Clasificadores*, cuyas instancias representan al sistema en ejecución. A este conjunto de clasificadores le llamamos *arquitectura de clases*.

La creación de buenas arquitecturas es una tarea difícil para los diseñadores principantes. Los patrones de diseño guían a los diseñadores hacia soluciones probadas, más robustas y reutilizables. ExempLUM es una aplicación para diseño e implementación de sistemas OO codificados con JAVA. Permite describir arquitecturas mediante diagramas de clases, usando el estándar del OMG, UML. Además, incorpora el concepto de patrón de diseño tanto en la funcionalidad de la propia herramienta, como en el diseño y arquitectura de la aplicación misma. ExempLUM facilita la transición hacia la etapa de implementación de los sistemas, ya que posee funcionalidad para generar código basado en los diagramas creados.

Palabras clave: Objeto, Clase, Asociación, Patrón, Diseño, Modelo, UML, Código.

ADVERTENCIA

“Este informe contiene información desarrollada por la Escuela Superior de Cómputo del Instituto Politécnico Nacional a partir de datos y documentos con derecho de propiedad y por lo tanto su uso queda restringido a las aplicaciones que explícitamente se convengan”.

La aplicación no convenida exime a la escuela de su responsabilidad técnica y da lugar a las consecuencias legales que para tal efecto se determinen.

Información adicional sobre este reporte técnico podrá obtenerse en la Subdirección Académica de la Escuela Superior de Computo del Instituto Politécnico Nacional, sita en Avenida Juan de Dios Batíz s/n, teléfono: 729-6000 extensión 52012.

AGRADECIMIENTOS

Primeramente agradezco a mis padres por todo el apoyo que me han brindado durante mi carrera y a lo largo de mi vida para lograr todas mis metas.

A mi hermano por su apoyo incondicional y por animarme a seguir adelante en los peores momentos.

A Dios por permitirme lograr cada uno de los objetivos que me propongo cada día.

Finalmente, me gustaría agradecer a todas aquellas personas que de alguna manera han influido en mi vida para ser una mejor persona.

Benito Joel

IV. RESUMEN

El diseño y programación Orientados a Objetos han probado ser enfoques simples pero poderosos para construir aplicaciones complejas.

El apegarse al paradigma Orientado a Objetos produce aplicaciones más claras, concretas y lógicas. Todo esto es obtenible solamente a través de la construcción de buenos diseños.

La necesidad de diseños sólidos ha traído consigo la creación de una notación para los diseñadores, analistas y desarrolladores. El Lenguaje Unificado de Modelado es esta misma notación. UML es un estándar de la indstria desarrollado por el OMG para uniformizar la notación usada para el modelado de sistemas.

Los sistemas orientados al objeto están constituidos por *clasificadores* y *relaciones*. Un *clasificador* es un grupo o categoría de objetos, personas, etc, basado en sus cualidades, llamadas *atributos* y sus comportamientos, llamados *métodos* u *operaciones*. Cuando un clasificador se conecta de manera conceptual, la conexión recibe el nombre de *relación*. Las relaciones también se llaman asociaciones. El estándar UML define algunos casos específicos de la asociación, cada uno con un significado especial, a saber: generalización (o herencia) para extender una clase, realización para implementar una interfaz, agregación para asociar conjuntos de instancias y composición (como un caso especial de la relación de agregación).

Los diagramas de clases UML usan clasificadores -clases e interfaces- y relaciones para modelar la vista estática de un sistema. Los clasificadores y las asociaciones se reflejan en algún momento durante la transición hacia la fase de implementación del ciclo de desarrollo. En realidad, si la aplicación se escribe en algún lenguaje orientado a objetos, como JAVA, existe una relación uno a uno entre una clase o interfaz JAVA y el clasificador en el diagrama UML. Así, esta relación puede usarse para la generación de código de características inherentes a los diagramas.

La construcción de aplicaciones grandes es una tarea compleja; hay muchos clasificadores y muchas asociaciones entre éstos, se forman grandes jerarquías de clases y el llevar cuenta de todos estos elementos se vuelve una tarea difícil. También aparecen disyuntivas en cierto momento, disyuntivas que los analistas, diseñadores y desarrolladores tienen que considerar basándose en las ventajas y desventajas de los distintos diseños que pueden construirse. Los diseñadores principiantes pueden tener dificultades tomando una decisión. No conocen las consecuencias de la elección que hagan. Los diseñadores experimentados saben qué camino seguir a partir de experiencias pasadas. Los Patrones de Diseño capturan este 'conocimiento' en una descripción del entorno en el que un problema ocurre y alguna arquitectura de clases que resuelva el problema. Los Patrones de Diseño representan soluciones encapsuladas en piezas que sirven para construir aplicaciones complejas de manera confiable.

Consideramos estos dos temas: la Construcción de Software Orientado a Objetos y los Patrones de diseño como dos de las ramas más importantes de la Ingeniería de Software. Representan soluciones y paradigmas para la implementación y diseño de mejor software. ExempLUM es un programa orientado a objetos escrito en lenguaje JAVA que combina estos conceptos en una aplicación útil para la creación de software.

ExempLUM posee muchas características que lo hacen una herramienta auxiliar en todo tipo de entorno de diseño Orientado a Objetos:

Algunas aplicaciones son:

- Para generar documentación de sistemas. Puede exportar diagramas como archivos de imágen GIF para poderlos incrustar en documentos o páginas web.
- Para generar el esqueleto o estructura básica a partir de la arquitectura definida en un diagrama.
- Como herramienta para introducirse al uso de Patrones de Diseño.
- Como herramienta didáctica para diseño orientado a objetos.
- Para explotar las características de aplicaciones CASE líderes de la industria como Rational Rose o Together Control Center por medio de la exportación de modelos con formato XMI.

V. *ABSTRACT*

Object-Oriented programming and design have shown to be simple yet powerful means to develop complex computer applications.

Developing under the Object-Oriented paradigm yields clearer, more concrete and logical applications. All this, of course, can be acquired only through the construction of profitable designs.

Solid design needs have brought in the creation of a notation for designers, analysts and developers. The Unified Modeling Language (UML) provides this notation. UML is an industry standard developed by the Object Management Group (OMG) to standardize the system of signs and figures used to model computer systems.

Object-oriented systems are made up of *Classifiers* and *Relationships. A classifier* is a group or category of objects, persons, etc., based on their qualities which are named *attributes*, and their behaviors, named *methods* or *operations*. When classifiers get connected in a conceptual way, the connection is known as *relationship*. Relationships are also known as associations. The UML standard defines some particular cases of the *association* connection, each one standing for a special meaning, namely: generalization (or *inheritance*) to extend a class, realization to implement an interface, aggregation to associate sets of instances and composition (as a very special case of the aggregation relationship).

UML Class diagrams use classifiers -classes and interfaces- and relationships to model the static view of a system. Classifiers and associations get reflected in some moment in the transition to the implementation phase of the development cycle. Actually, if the application is being written with object-oriented languages, such as JAVA, there is a one-to-one relationship between a JAVA class or interface and a classifier in the UML class diagram!. Thus, this relationship can be used for the code generation of the characteristics inherent to diagrams.

Construction of mid and large-sized computer applications is a complex task; there are lots of classifiers and many associations between them, large class-hierarchies appear and managing all this elements becomes a difficult task. Also decision trees turn up in some moment, forks the analysts, designers or developers have to consider based on the trade-offs of the various designs that can be constructed. Begginer designers can have hard times making up their minds. They don't know the consequences of the choices they make. Experienced designers know from previous designs (sometimes actually *mistakes*, hehe) which way to go. Design Patterns capture these 'wisdom' in a convenient description of the environment in which a problem occurs besides some proven class-architectures that solve the problem. Design Patterns represent solutions as pieces to build tangled or intricate applications in a reliable way.

We consider these two topics: Construction of Object-Oriented Software and Design Patterns as two of the most important branches in the Software Engineering field. They represent solutions and paradigms for better software design and implementation. ExempLUM is an object-oriented program written in the JAVA language which merges these concepts in a useful application to create software.

ExempLUM has many features that make it a helpful tool in all ranges of Object-Oriented design environments. Some uses are:

- To generate systems documentation. It can export diagrams as GIF image files to be embedded in documents or html files.
- To generate the skeleton of a system based on the architecture defined in a diagram.
- As a tool for getting into the use of Design Patterns.
- As a teaching tool for object-oriented design.
- To exploit features of some of the industry leading CASE applications such as Rational Rose or Together Control Center through the exportation of models with XMI format.

III. OBJETIVO.

Desarrollar una herramienta de software capaz de genera código (Java) basado en patrones de diseño OO, que permita la definición e incorporación de nuevos patrones sin tener que modificar su código. Dicho sistema tendrá, como medio para interactuar, una GUI que proporcionará al usuario un entorno amigable para facilitar la construcción de la arquitectura de clase de una aplicación.

IV. JUSTIFICACIÓN.

El proyecto, cuyas características se describirán más profusamente en secciones posteriores, pretende facilitar la escritura de código en el proceso de programación basándose en el reciente paradigma de patrones de diseño en la POO.

La adecuada aplicación de patrones de diseño requiere conocer detalladamente Diseño Orientado a Objetos (DOO) y POO

Se ha fijado como meta que el software deberá contar con parámetros de robustez, interoperatibilidad, alto grado de reutilización y extensibilidad; lo cual tiene como consecuencia, llevar al pie de la letra la Ingeniería de Software.

El desarrollo de la herramienta, dada su complejidad, requiere de la integración de distintas disciplinas de cómputo:

- Análisis Orientado a Objetos (AOO)
- Diseño Orientado a Objetos (DOO)
- Patrones de Diseño
- Programación Visual
- Programación Orientada a Objetos (POO)
- Calidad en el Software

Tradicionalmente, la aplicación de patrones de diseño se hace manualmente. La herramienta es innovadora porque facilita el uso de patrones por medio de una interfaz gráfica.

Al proporcionar un esquema predefinido por alguno de los patrones que se integrarán a la herramienta, el usuario (novel o experimentado) no se verá en la necesidad de comenzar un diseño desde cero.

V. ANTECEDENTES.

"*...El diseño de objetos es difícil, y el diseño de objetos reutilizables es aún más difícil. Debe encontrar los objetos correctos, unirlos en clases de la granularidad correcta, definir interfaces de clases y jerarquías de herencias, y establecer relaciones clave entre ellas..*"

"... Un Patrón nombra, abstrae e identifica los aspectos clave de una estructura de diseño común que lo hacen útil (al patrón) para la creación de diseños reutilizables orientados a objetos.."

Design Patterns: Elements of Reusable Object-Oriented Software;
Gamma, Helm, Johnson, Vlissides.

Al enunciar el nombre de "Herramienta de Generación de Código a partir de Patrones de Diseño Orientados al Objeto" podemos identificar inmediatamente los tres conceptos medulares sobre los que se sustenta esta herramienta: Orientación al Objeto, Patrones de Diseño, y la Generación de Código. Pretendemos en esta sección definir (al menos delimitar alguna definición ya existente) estos conceptos y ubicar los tres en el contexto del Trabajo Terminal 291.

Retomemos primero el concepto de Orientación al Objeto. La Programación Orientada a Objetos proporciona una visión de una aplicación consistente en objetos con un estado propio dotados de funcionalidad. Los objetos se comunican entre sí y cada uno tiene una forma propia de respuesta, que viene determinada por una serie de procedimientos que son asociados a cada objeto. El objetivo de esta tecnología es obtener un software más consistente, robusto, portable y reutilizable; más fácil de desarrollar, codificar, verificar, mantener, refinar y extender. El paradigma orientado a objetos representa un paso más en la dirección de acercar el lenguaje de las soluciones informáticas al lenguaje en que se plantean los problemas.

El Paradigma Orientado a Objetos afecta a la herramienta del Trabajo Terminal 291 por 2 razones:

- El análisis, diseño e implementación (e incluso tal vez mantenimiento) se realizarán utilizando metodologías orientadas al objeto. Particularmente se puede hablar del estándar de el OMG llamado UML (*Unified Modeling Language*, Lenguaje Unificado de Modelado), que proporciona un método estandarizado para modelar sistemas de información, y que tiene por objetivo lograr modelos que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos más fácilmente por los clientes, usuarios o de aquello que se modela.
- La herramienta proporciona un punto de acceso al mundo del modelado y construcción de software orientado a objetos: se basa en las técnicas y algunas de las prestaciones que este paradigma ofrece. Esta característica invita (¿u obliga?) al usuario a familiarizarse con los conceptos de la Orientación al Objeto, paradigma que definitivamente modela una problemática real de una mejor manera.

  La orientación a objetos trata de cumplir las necesidades de los usuarios finales, así como las propias de los desarrolladores de productos de software. Estas tareas se

realizan mediante la creación de modelos del mundo real. El soporte fundamental es el *modelo objeto*. Los cuatro elementos (propiedades) más importantes de este modelo son:

- Abstracción. Propiedad que permite representar las características esenciales de un objeto, sin preocuparse de las restantes características (no esenciales).
- Encapsulamiento. Esta propiedad permite asegurar que el contenido de la información de un objeto está oculta al mundo exterior: el objeto A no conoce lo que hace el objeto B.
- Modularidad. Es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas *módulos*), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en si y de las restantes partes.
- Jerarquía. Esta propiedad permite una ordenación de las abstracciones. Las dos jerarquías más importantes de un sistema complejo son: estructura de clases (jerarquía "es-un": generalización/especialización) y estructura de objetos (jerarquía "parte-de": agregación).

Existe una quinta propiedad que no suele ser considerada como fundamental, pero por su importancia, no tiene sentido considerar un *objeto modelo* que no soporte esta propiedad. Esta propiedad es el *polimorfismo*, que literalmente indica la posibilidad de que una entidad tome *muchas formas*. En términos prácticos, el polimorfismo permite referirse a objetos de distintas clases con el mismo elemento de programa y realizar una operación de diferentes formas, según sea el objeto que se referencia en ese momento.

La propiedad de jerarquía es la propiedad que más explota (tal vez sólo igualada por la abstracción) la herramienta. Se aprovecha el hecho de que esta propiedad establece una relación semántica y de comportamiento entre los objetos que la poseen. Así tenemos que en la una agregación de piezas (un rompecabezas), en una agregación de videocasetes (una videoteca) y en una agregación de prendas de vestir (un guardarropa) seguramente se encontrarán comportamientos (que en el software se modelan como métodos de clase) por parte de los contenedores (el rompecabezas, la videoteca y el guardarropa) para agregar a sus contenidos (las piezas, los videos y las prendas), o algún otro método, por ejemplo para deshacerse de alguno de los objetos contenidos, o para iterar sobre ellos, entre algunas otras. Estos comportamientos se imprimen primeramente en el diagrama de clases que el usuario de la herramienta podrá editar, y se reflejará finalmente en el código a generar asociado a ese diagrama.

Pasemos ahora al concepto de Patrón. En palabras de Christopher Alexander:

> *"Cada patrón describe un problema que aparece una y otra vez en nuestro entorno, y después describe la esencia de la solución a ese problema, en forma tal que se puede usar esta solución un millón de veces sin tener que volver a hacerlo igual dos veces".*
>
> "A Pattern Language",Christopher Alexander, Sara Ishikawa, Murria Silverstein; 1977

En los tres últimos años, usted quizá haya escuchado el término patrón de diseño. Un patrón de diseño es una descripción de un problema que aparece una y otra vez en el diseño de sistemas.

Los patrones de diseño saltaron a la fama cuando "la pandilla de los cuatro" (Gamma, Helm, Johnson y Vlissides) publicaron su famoso libro "Design Patterns" (Addison Wesley, 1994). Este libro presenta un catálogo de patrones en el ámbito de objetos. Los patrones, ahí presentados, describen soluciones simples y elegantes a problemas de diseño de objetos de una forma muy práctica y útil.

Pero la historia no empieza ahí. Es con el arquitecto Christopher Alexander y sus colegas, que proponen la arquitectura de edificios y ciudades basándose en un lenguaje de patrones. Un lenguaje de patrones consiste en una colección de plantillas, todas con un formato similar - nombre del patrón, descripción del patrón, problema que resuelve, relaciones con otros patrones, consecuencia del patrón -, que describen soluciones a un entorno arquitectónico.

Por ejemplo, un lenguaje de patrones específico para problemas de arquitectura podría tener una plantilla correspondiente a una alberca abierta con los vestidores en cierta disposición para mantener a los nadadores cerca de la alberca. Otra plantilla del mismo lenguaje de patrones, podría describir el estacionamiento de un lugar, con una distribución de cajones para los carros de discapacitados, que requieren estar cerca de la entrada del lugar.

Ahora imagínese a usted mismo como un arquitecto novato que se enfrenta a un escenario con múltiples problemas de distribución bajo un presupuesto apretado. ¿Cómo saber si el diseño propuesto no tendrá consecuencias poco agradables? ¿Está inventando el hilo negro? Es en este ámbito donde es útil tener un catálogo de soluciones ya experimentadas, es decir un lenguaje de patrones.

En resumen, los lenguajes de patrones de diseño permiten a los novatos en diseño, aprovechar la experiencia previa de expertos en esta área de conocimiento, ayudándolos tanto a identificar problemas específicos como sus soluciones respectivas.

En general, un patrón posee cuatro elementos esenciales:

1. **El Nombre del Patrón** es una referencia que podemos usar para describir un problema de diseño, sus soluciones, y consecuencias en una o dos palabras. El nombrar a un patrón inmediatamente incrementa nuestro vocabulario de diseño. Nos

permite diseñar en un nivel de abstracción más alto. El tener un vocabulario para patrones nos permite hablar acerca de ellos con nuestros colegas, en nuestra documentación, e incluso a nosotros mismos. Hace más fácil pensar acerca de diseños y comunicarlos junto con los compromisos que conllevan (*trade-offs*) a otras personas.

2. **El Problema** describe cuándo aplicar el patrón. Explica el problema y su contexto. Debería describir problemas de diseño específico tales como la forma de representar algoritmos como objetos. Debería describir estructuras de clases u objetos que sean indicadores de defectos de un diseño inflexible. Algunas veces el problema incluirá una lista de condiciones que deben cumplirse antes de que tenga sentido aplicar el patrón.

3. **La Solución** describe los elementos que forman al diseño, sus relaciones, responsabilidades y la forma en que colaboran. La solución no describe un diseño o implementación particular, porque un patrón es como una plantilla que puede ser aplicada en muchas situaciones distintas. En su lugar, los patrones proporcionan una descripción abstracta de un problema de diseño y cómo un ordenamiento general de elementos (clases y objetos en nuestro caso) lo resuelven.

4. **Las Consecuencias** son los resultados y compromisos (*trade offs*, lo que se gana y lo que se pierde) del patrón que se aplica. Aunque las consecuencias usualmente no se mencionan cuando se describen las decisiones de diseño, son críticas para la evaluación de alternativas de diseño y para la comprensión de costos y beneficios de la aplicación del patrón.

La incorporación de los Patrones de Diseño a la Herramienta consiste en proporcionar al usuario una forma de trabajar con los Patrones más básicos, útiles y genéricos (tal vez hasta célebres): los patrones de la Pandilla de Los Cuatro (*Gamma et. al*.). Estos patrones se presentan al usuario en forma de un diagrama de clases utilizando los nombres de los colaboradores originales de la Pandilla de Los Cuatro. El usuario así podrá utilizar como plantillas las estructuras de los patrones propuestas en el libro de *Gamma* para su uso en diseños propios.

No olvidemos además que estos Patrones dejan implícito también un comportamiento asociado a los objetos que colaboran en él. Como podemos imaginar, es posible representar esta interacción como métodos de clases y algunos atributos.

Llegamos finalmente al concepto de generación de código.
Ya hemos mencionado que un diagrama de clases representa ciertos conceptos y algunos (no todos) comportamientos intrínsecos que quedan implícitos al establecer relaciones entre las clases que participan en él. Por otro lado tenemos el comportamiento de que dotan los

Patrones a una estructura de clases. Este comportamiento se refleja en el código como métodos y atributos, así como jerarquías de interfaces y clases que las realizan.

El objetivo primordial de un generador de código es automatizar la escritura de secciones de código (libres de error) que aparecen invariablemente en un programa que posee algún comportamiento ya conocido.

En conclusión, la herramienta ayudará a extraer los métodos y atributos que imprimen un comportamiento específico a una estructura de clases, para su conversión automática en código JAVA.

1. INTRODUCCIÓN

La *POO* presenta para los años 90 lo que la Programación Estructurada fue para los 70: un nuevo e importante paradigma para mejorar la construcción, mantenimiento y utilización de software. Los métodos tradicionales de programación tienden a ver los programas como un conjunto de procedimientos que se llaman unos a otros. Cada procedimiento tiene asociados datos pasivos sobre los que opera. La *POO* cambia esta visión por otra en la que una aplicación está computa por objetos con estado propio dotados de funcionalidad. Los objetos se comunican entre sí y cada uno tiene una forma propia de respuesta, que viene determinada por una serie de procedimientos que son asociados a cada objeto. El objetivo de esta tecnología es obtener un software más consistente, robusto, portable y reutilizable; más fácil de desarrollar, codificar, verificar, mantener, refinar y extender. El paradigma orientado a objetos representa un paso más en la dirección de acercar el lenguaje de las soluciones informáticas al lenguaje en que se plantean los problemas.

Aún cuando la *POO* permitió superar las carencias de la Programación Estructurada pare el correcto diseño de software, dicha tecnología también trajo consigo deficiencias que el impidieron dar una respuesta adecuada a *toda* problemática de diseño. Incluso, el origen de tales deficiencias radica en el hecho de que *POO*, al igual de la Estructurada, no garantiza el *reuso* de soluciones al diseñador a iniciar proyectos partiendo desde cero, cuando bien podría apoyarse de soluciones que ya han sido desarrolladas.

A pesar de ello, con frecuencia se ha observado que existen buenas soluciones para cierto problemas que aparecen repetidamente en los buenos diseños. En lugar de crear un diseño completamente nuevo desde el principio, en cada oportunidad los diseñadores ahorran una cantidad de trabajo incorporando en sus diseños esta soluciones "comunes". Dichas soluciones estándar son diseño útiles que han sido comprobados por años de experiencia. Estas "soluciones en espera de problemas " son llamadas **patrones**.

Christopher Alexander, un arquitecto, propone por primera vez un lenguaje de patrones para poder diseñar ciudades, barrios, grupos de edificios, plazas, edificios, habitaciones, etc. Él describe a un *patrón* como un *esquema* "presenta un problema que aparece una y otra vez en nuestro entorno y luego, describe la esencial de la solución al problema de tal manera de que se la pueda utilizar de forma indefinida".

Para esta autor, cada esquema está conectado con algún otro de nivel superior (más amplio) y, además, con otros de nivel inferior (más reducidos). Cada nivel de patrones ayuda a completar un nivel superior y a su vez se completa por un nivel inferior.

Los proponentes del diseño OO han encontrado en los *esquemas* de Alexander una forma efectiva de solucionar los problemas que se presentan en el diseño de software y, al mismo tiempo, una manera de disminuir las deficiencias de paradigma OO. Esto dio como resultado la aparición de una serie de propuestas sobre patrones de diseño dentro de ámbito OO durante los últimos 35 años. Pero no fue, sino hasta mediados de la década de los noventa, cuando dicha tecnología iniciara su proceso de consolidación.

Tal vez la mayor promoción en la idea de patrones se encuentra en el libro *"Design Patterns, Elements of Reusable Object-Oriented Software"*, de E.Gamma, R.Helm, R.Jonson y J.Vlissides, publicado en 1995, el cual se ha convertido en un *estándar* dentro de la *POO*. El objetivo de los autores es rescatar la experiencia de los diseños exitosos de sistemas creados con el modelo de objetos y, presentarla en forma de un catálogo de patrones. Este catálogo puede ser utilizado por los diseñadores novatos, igual que por los experimentados, para reutilizar las micro – arquitecturas de clases que han comprobado su utilidad en otras ocasiones y, de esta manera, fomentar realmente el reuso dentro el modelo de objetos y evitar al máximo descubrir el hilo negro.

Un **patrón** *es una unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probada a un problema recurrente dentro de un cierto contexto*. El objetivo de los patrones es crear un lenguaje común a una comunidad de desarrolladores para comunicar experiencia sobre los problemas y sus soluciones.

Existen varios patrones de diseño dependiendo del nivel de abstracción, del contexto particular en el cual aplican, ó del proceso de desarrollo. Algunos de estos tipos son:

- de Diseño.
- *Idioms.*
- de Arquitectura.
- de Análisis.
- para Ambientes Distribuidos.
- de Negocios.
- de Proceso y Organizacionales.

En base a lo anterior se puede observar que el empleo de la *POO*, a través de patrones de diseño, es indispensable para la construcción de sistemas complejos, grandes y escalables; donde resulta crítico disponer de soluciones robustas y confiables.

Por lo tanto, disponer de una herramienta que integre dichas características en un entorno gráfico, sería de gran utilidad par los diseñadores. En especial, si se considera que en la actualidad no existen ninguna herramienta que cumpla estrictamente con las características arriba mencionadas.

3. ANÁLISIS

3.1 TERMINOLOGÍA

Definimos a continuación el significado de algunos términos utilizados a lo largo del análisis para evitar ambigüedades o concepciones no adecuadas por parte del lector.

Diseñar
Preparar de manera preliminar la forma y estructura de un sistema de software.

Diseño
Cuando utilizamos la frase "*el diseño*", nos referimos a la organización en subsistemas basados tanto en la estructura del análisis como en una arquitectura propuesta, aplica normalmente a "*el diseño* de ExempLUM". Cuando usamos la frase "*un diseño*" nos referimos a un diagrama de clases construido con ExempLUM.

Editor de Diagramas de clases UML
Herramienta con interfaz gráfica que permite manipular los símbolos de los diagramas de clases definidos en el estándar UML de la OMG. Esta manipulación implica la configuración *directa* de los símbolos utilizados para los diagramas de clases para la construcción de algún diseño específico. La *manipulación directa*, implica que la herramienta permite dotar de características a los diagramas UML manejando únicamente los elementos gráficos del diagrama, sin necesidad de llenar múltiples formularios.

Modelar
Construcción de la representación de una situación del mundo real, utilizando cualquier técnica para modelar (UML, OMT, BOOCH, etc.)

Modelo
Representación de una situación del mundo real. En el ámbito de análisis, diseño e implementación de ExempLUM, este "modelo" será un diagrama de clases UML, o, por extensión, clases JAVA.

Meta-Modelo
El meta-modelo, es un modelo que describe a otros. El meta-modelo más importante de ExempLUM es aquel conformado por las clases (JAVA) que modelan en memoria al diagrama de clases, es decir, las clases que modelan a: una clase, una interfaz, un método, una relación de herencia, etc.

3.2 IDENTIFICACIÓN DE PROBLEMAS

Descripción de Software de manera Gráfica

El software hoy en día puede ser muy complejo en su estructura y arquitectura, pero eso no implica que deba ser difícil de usar. En la industria del hardware, todo está construido a partir de partes más pequeñas. Este enfoque también se puede aplicar en el mundo del software –objetos a un nivel de abstracción mayor se tratan como "piezas de software." Para simplificar aún más el proceso podemos usar imágenes en lugar de descripciones textuales para mostrar las relaciones entre objetos en un sistema complejo. Aunque las imágenes son más eficaces que las descripciones textuales solamente, la experiencia ha probado que el comunicar ideas complejas de manera efectiva requiere algo más que simples diagramas de flujo.

Lenguaje Unificado de Modelado

Las primeras metodologías, tales como la notación Booch, OMT y otras, servían para el mismo propósito: expresar de manera gráfica información sobre la arquitectura del software. Sin embargo, estas metodologías lograban su propósito en formas ligeramente distintas con distintos niveles de éxito. En 1994, Grady Booch, James Raumbaugh e Ivar Jacobson unificaron sus métodos y experiencia. UML fue el fruto de su esfuerzo. UML fue creado con dos objetivos:

- Reflejar las mejores prácticas en la industria y
- Desmitificar el proceso de modelado de sistemas de software.

En resumen, UML *proporciona símbolos gráficos normalizados de las aplicaciones de software y permite a los desarrolladores adquirir conocimiento rápidamente sobre la aplicación.* UML es un lenguaje y un proceso con una notación neutral, lo que quiere decir que se puede usar para diseñar un sistema orientado a objetos completo en cualquier lenguaje de programación y con cualquier proceso de desarrollo de software.

Patrones de Diseño

Un patrón de diseño es un conjunto de clasificadores, relaciones y comportamientos parametrizados, que se pueden aplicar a múltiples situaciones enlazando elementos del modelo (clases normalmente) con los roles del patrón. Es una plantilla de colaboración.

Los patrones representan una colaboración parametrizada que se puede emplear en múltiples ocasiones dentro de uno o más sistemas. Para ser un patrón, la colaboración tiene que ser utilizable en una amplia gama de situaciones, para justificar darle un nombre. Un patrón es una solución que demostradamente funciona en un cierto número de situaciones. No es necesariamente la única solución del problema, pero es una solución que ha sido efectiva en el pasado. Casi todos los patrones tienen ventajas y desventajas, que dependen de distintos aspectos del sistema global. El creador de modelos debe considerar estas ventajas y desventajas antes de tomar la decisión de utilizar un patrón.

Todas las arquitecturas orientadas a objetos bien estructuradas están llenas de patrones. El diseño de software orientado a objetos es difícil, y el diseño de objetos reutilizables es aún más difícil.

Modelado Visual

El desarrollo de un modelo para un sistema de software, antes de su construcción o renovación, es tan esencial como tener un plano para construir un edificio. Los buenos modelos son vitales para la comunicación entre los desarrolladores de un sistema de software.

A inicios de la década de los 90, las herramientas para el modelado de software orientado a objetos, seguidas por el desarrollo del enfoque de modelado visual. "Modelado Visual" implica un proceso. Este proceso implica primeramente diseñar el sistema dibujando diagramas (los planos) y después emplear herramientas para convertir esos diagramas en código. El valor de este tipo de enfoque es que la codificación que usualmente es una tarea tediosa se realiza casi automáticamente, dando libertad al programador para enfocarse en los problemas e implicaciones de diseño, además, la transición de la fase de diseño a la de implementación es más simple y libre de problemas. Utilizar la característica de la generación de código, el desarrollador puede avanzar y regresar entre las etapas de codificación y diseño.
Hoy en día, las herramientas de modelado visual proporcionan muchas características que reemplazan algunas de las más tediosas tareas para los diseñadores, programadores y generadores de documentación.

Intercambio de Meta-Datos

El estándar XMI creado por el Object Management Group -(OMG) es uno de los recientes desarrollos más útiles para la comunidad de desarrollo de UML. XMI es un formato de intercambio con el potencial de, finalmente, permitir a las mejores herramientas de desarrollo compartir modelos fácilmente. Por ejemplo, en lugar de escribir scripts para crear informes con una herramienta de modelado UML, un usuario simplemente podría exportar el modelo en desarrollo utilizando XMI e importarlo a una herramienta especializada para la creación de informes. Como XMI utiliza XML para representar la información del modelo, un grupo de soluciones basadas en XML estarán inmediatamente disponibles, tales como las hojas de estilo (stylesheets) XSL para la presentación basada en navegadores y las herramientas de interrogación (query tools) XQL para funciones de búsqueda.

El estándar XMI es complejo y tomará tiempo reconciliar muchas cuestiones de compatibilidad que inevitablemente se suscitarán antes de que su uso pueda generalizarse. Sin embargo, dado que el XMI ha sido desarrollado por compañías destacadas en la industria, tales como IBM y Unisys entre otras, se anticipa que los productos aparecerán próximamente. Incumbe a la comunidad de usuarios exigir el soporte XMI en las herramientas UML mediante su inclusión al principio de las listas de control de funciones. Para mayor información sobre XMI, consulte la excelente página web de IBM.

El propósito principal de XMI es permitir el intercambio fácil de meta-datos entre herramientas de modelado (basadas en el UML de la OMG) y entre herramientas y repositorios de meta-datos en entornos distribuidos heterogéneos.
XMI integra tres estándares de la industria:

XML - *eXtensible Markup Language*, un estándar W3C.
UML - *Unified Modeling Language*, un estándar de modelado del OMG
MOF - Meta *Object Facility* y el estándar de repositorios de meta-datos del OMG.

La integración de estos tres estándares en XMI combina lo mejor en tecnologías de meta-datos y modelado de OMG y W3C permitiendo a los desarrolladores de sistemas distribuidos compartir modelos de objetos y otros meta-datos a través de redes (principalmente Internet).

3.3 REQUERIMIENTOS Y NECESIDADES DEL SISTEMA

Los requerimientos del sistema, se encuentran implícitos en el objetivo del trabajo terminal, definido originalmente en el Protocolo para Proyectos de Trabajo Terminal, de donde se extrae que la herramienta requiere:

- ser capaz de generar código en lenguaje JAVA.
- incorporar los patrones de diseño como unidades que puedan insertarse en los diagramas de clases.
- permitir la incorporación de nuevos patrones sin tener que modificar el código de la aplicación.
- proporcionar una IGU como medio para interactuar con los diagramas –Editor de diagramas de clases UML-.
- ser amigable y que facilite la definición de la arquitectura de clases de una aplicación.

ExempLUM se creó como una herramienta de modelado y proporciona las siguientes características:

- Una interfaz gráfica de usuario con un área de trabajo similar a una hoja de papel, que permita editar los símbolos definidos para los diagramas de clases UML.
- Generación de código basada en un diagrama creado con ExempLUM.
- Exportación de modelos estáticos (diagramas de clases) en formato XMI (*XML Metadata Interchange*) permitiendo el intercambio de información con otras herramientas compatibles con *XMI*.
- Soporte para la mayor parte de las características de UML 1.3 y la diagramación sencilla.
- Sincronización del meta-modelo. Al hacer cambios en los formularios y las vistas gráficas de métodos, atributos, clases e interfaces éstos se reflejan en el meta-modelo subyacente.
- Exportación del Diagrama. Conversión a imagen con formato GIF.
- Generación automática de patrones de la Pandilla de los Cuatro. Para uso y conocimiento de los diseñadores.
- Registro de una estructura de clases como patrón de usuario. Para facilitar diseños posteriores basados en la experiencia propia del usuario.

5. RESULTADOS

- Software
    - Herramienta para generar código (Java) a partir de un diseño orientado a objetos, utilizando como plantillas alguno(s) de los 23 patrones base de la POO u otros nuevos que también podrán ser integrados a la aplicación de acuerdo a las necesidades del usuario.
- Documentación
    - Manual de Usuario
    - Reporte Técnico

6. CONCLUSIONES

Una vez culminadas las etapas de la Ingeniería de Software correspondientes al análisis, diseño e implementación de la "Herramienta de generación de código a partir de patrones de diseño orientados a objetos" (TT0291), se concluye lo siguiente:

- Los objetivos, propósitos y metas fijados en la propuesta del proyecto (protocolo) se lograron satisfactoriamente.

- Se culminó el desarrollo de una herramienta de software (ExempLUM) capaz de generar código (lenguaje Java) basado en patrones de diseño orientados a objetos (GoF), que permite la incorporación de nuevos patrones (de usuario) sin tener que modificar su código y que dispone de una IGU que facilita la arquitectura de clases (UML) de una aplicación.

- Un manual técnico, que describe la arquitectura, implementación y función interna de la herramienta; así como un manual de usuario, que le indica a este último como operar la aplicación, complementan los productos esperados.

- La herramienta ha sido dotada de operatividad adicional que extiende la funcionalidad de un editor de diagramas de clases UML.

7. POSIBLES MEJORAS

Comparando ExempLUM con algunas herramientas CASE existentes en el mercado (Rational Rose, Magic Draw, Together), se contemplan las siguientes posibles mejoras:

- Incorporación de todos los diagramas propios del estándar UML.

Área	Vista	Diagramas	Conceptos Principales
Estructural	Vista estática	Diagrama de clases	Clase, asociación, generalización, dependencia, realización, interfaz
	Vista de casos de uso	Diagrama de casos de uso	Caso de uso, actor, asociación, extensión, inclusión, generalización de casos de uso
	Vista de implementación	Diagrama de componentes	Componente, interfaz, dependencia, realización
	Vista de despliegue	Diagrama de despliegue	Nodo, componente, dependencia, localización
Dinámica	Vista de máquina de estados	Diagrama de estados	Estado, evento, transición, acción
	Vista de actividad	Diagrama de actividad	Estado, actividad, transición de terminación, división, unión
	Vista de interacción	Diagrama de secuencia	Interacción, objeto, mensaje, activación
		Diagrama de colaboración	Colaboración, interacción, rol de colaboración, mensaje
Gestión del modelo	Vista de gestión del modelo	Diagrama de clases	Paquete, subsistema, modelo

Tabla 7.1. Diagramas UML.

- Generación de Código en diferentes lenguajes.
  - C++
  - Smalltalk
  - Eiffel
  - Visual Basic
  - Visual C++
- Generación de Código de los elementos seleccionados en el diagrama.
- Importación de Bibliotecas JAVA.
- Importación de archivos con formato XMI.
- Impresión de los diagramas.
- Función Acercar/Alejar (Zoom) a un diagrama.
- Incorporación de OR y AND entre relaciones.
- Incorporación de otros Patrones de Diseño conocidos (además de los Patrones GoF).

8. BIBLIOGRAFÍA

1. Design Patterns. Elements of Reusable Object-Oriented Software.
   Gamma, Erich et al.
   Addison-Wesley
   2000

2. Modelado y Diseño Orientado a objetos.
   Rumbaugh, James et al.
   Prentice Hall
   1996

3. Programación Orientada a Objetos.
   Luis Joyanes Aguilar
   McGraw-Hill
   1998

4. El Proceso Unificado de Desarrollo de Software.
   Jacobson, Ivar et al.
   Addison-Wesley
   2000

5. El Lenguaje Unificado de Modelado. Manual de Referencia.
   Rumbaugh, James et al.
   Addison-Wesley
   2000

6. Aprendiendo UML.
   Scmuller, Joseph
   Prentice Hall
   2000

7. Manual de XML.
   Goldfard, Charles et al.
   Prentice Hall
   2000

8. Java 2 Complete.
   Steven Holzner et al.
   SYBEX
   1999

9. Análisis y Diseño de Sistemas .
   Kendall & Kendall.
   Prentice Hall
   1997

10. Java 1.2 Al descubierto.
Jamie Jaworsky
Prentice Hall
1998

11. Modern Compiler Implementation in C.
Andrew W. Appel
Cambridge University Press
2000

12. Algoritmos en C++.
Robert Sedgewick
Addison-Wesley/Diaz de Santos
2000

13. How to program Java.
Deitel & Deitel.
Prentice Hall
1999

14. JAVADOC – The Java API Documentation Generator.
Sun Microsystems

15. The Java Beans API specification.
Sun Microsystems

16. Ingeniería del Software "Un enfoque práctico"
Roger S. Pressman
McGraw-Hill

17. El Cálculo con Geometría Analítica.
Louis Leithold
Harla
1992

18. Geometría Analítica.
Lehmann
Limusa
2000

19. Concurrent Programming in Java.
Doug Lea
Addison-Wesley
2000

20. XML Metadata Interchange (XMI) Specification.
OMG.
2000

21. Meta Object Facility (MOF) Specification.
OMG.
1999